

A Generic Software Library for Creating Multimedia Browse/Search Applications

Cole, Richard Stephen

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author

For additional information about this publication click this link.

<http://qmro.qmul.ac.uk/jspui/handle/123456789/1746>

Information about this research object was correct at the time of download; we occasionally make corrections to records, please therefore check the published record when citing. For more information contact scholarlycommunications@qmul.ac.uk

A Generic Software Library for Creating Multimedia Browse/Search Applications

Richard Stephen Cole

Submitted for the Degree of Doctor of Philosophy

**Queen Mary and Westfield College
1996**

© Philips Electronics 1996

This thesis is dedicated to the memory of my grandfather
and to my Mum and Dad without whose support it would
never have been completed

Abstract

This thesis surveys the field of browse/search interactions. The results of this study form the basis of a specification of a representation scheme and a library of access functions which facilitate the creation of information-rich multimedia applications.

Evidence is provided for the hypothesis that browsing and searching are the extreme ends of a continuum of data access methods and that many browse/search interactions contain a mixture of both with the ratio varying as the interaction proceeds. These observations motivate the integration of browsing and search facilities so that applications can be built which exhibit both types of information access.

This work is tailored to the area of consumer multimedia with a review of the constraints that this imposes on the authoring process and the applications themselves forming part of this work.

The specification of the functionality of the function library, together with its implementation and testing are described in detail. The library has been evaluated by constructing a number of prototype applications which demonstrate the utility and scope of the library.

Table of Contents

Table of Contents	4
List of Figures	7
List of Tables.....	11
1 Introduction	15
1.1 Structure of the Thesis.....	18
2 Background to the Research.....	21
2.1 What is Browsing ?	21
2.2 Basic Terminology	21
2.3 Consumer Multimedia.....	22
2.4 Enabling Technologies.....	23
2.5 CD-i - A Typical Consumer Multimedia Platform	29
2.6 The Research Context	30
2.7 Multimedia Authoring Needs.....	34
2.8 Research Opportunities	36
2.9 Research Agenda.....	38
3 Objectives, Scope and Strategy.....	39
3.1 Objectives.....	39
3.2 Scope	42
3.3 Research Strategy	45
4 Study of Browsing.....	47
4.1 Examples Studied.....	47
4.2 Observations.....	67
4.3 Summary	69
5 Characteristics of Browse and Search.....	70
5.1 The Task Perspective	71
5.2 The Cognitive Perspective	74
5.3 Integration of Browse and Search	77
5.4 The Functional Perspective	79
5.5 Generic Browser Facilities.....	82

5.6	Summary	83
6	An Architecture for Multimedia Browsing	85
6.1	Data Model	85
6.2	Representation Schemes	87
6.3	Multimedia Data	90
6.4	Models of Browsing and Search	94
6.5	Browser Information Structures	94
6.6	Navigation	107
6.7	Marking	107
6.8	History Mechanisms	108
6.9	Search Methods	110
6.10	Authoring Perspective	117
6.11	Summary	118
6.12	Functional Specification of the Generic Browser	119
7	Library Implementations	122
7.1	CD-i - The Target Platform	122
7.2	Lisp Version	122
7.3	'C' Implementation	130
7.4	A Worked Example	144
7.5	Summary	154
8	User Interfaces	156
8.1	Generic Browser Interface and Widget Set	156
8.2	Navigable Structures	158
8.3	Presentation Aspects	163
8.4	Transition effects	164
8.5	Constraints Imposed by the Target System	166
8.6	Summary	167
9	Evaluation of the Generic Browser	168
9.1	Formal Test Program	169
9.2	Prototype Applications	169

9.3	Reimplementation of Existing Browsers.....	185
9.4	Summary	193
10	Summary & Conclusions	194
10.1	Contribution	194
10.2	Conclusions	195
11	Further Work	199
11.1	Further Work on The Generic Browser.....	199
11.2	Other Research Issues	201
11.3	Agents.....	202
11.4	World Wide Web	202
11.5	Summary	203
	References	204
	Glossary.....	215
	Appendices.....	219
A	CD-i - A Typical Consumer Multimedia Platform	219
B	Generic Browser Functions.....	225

List of Figures

Figure 1.	Film Browser - access by genre and title.	32
Figure 2.	Film Browser - browsing screen.	32
Figure 3.	Components of application creation.....	37
Figure 4.	Scope of the thesis.....	44
Figure 5.	Refinement loop	46
Figure 6.	The entry screen for Tell Me Why - moving the cursor over the doorways displays the topic, How Things Began.	52
Figure 7.	How Things Began - famous people gallery	53
Figure 8.	Selecting the Word Origins option from the How Things Began Topic.....	53
Figure 9.	The Word Origins screen.	54
Figure 10.	The Features screen.....	55
Figure 11.	The backdrop browser	57
Figure 12.	The character browser	58
Figure 13.	The music browser	59
Figure 14.	The Main Hall	60
Figure 15.	The Reference option	61
Figure 16.	The technology “drawer”.	61
Figure 17.	Topics	62
Figure 18.	Guided tours.	62
Figure 19.	The “Airborne” tour	63
Figure 20.	Information from the “Airborne” section of the museum	63
Figure 21.	Related Information - a stamp with an aeronautical subject	64
Figure 22.	PhotoCD on CD-i	65
Figure 23.	Picture-by-picture browser.....	66

Figure 24. Minipic browser	66
Figure 25. The sequence editor.....	67
Figure 26. Browse-search continuum.....	70
Figure 27. Salomon's components of browsing	71
Figure 28. Cove & Walsh's components of browsing.....	72
Figure 29. [Laurel et al]'s "modes"	72
Figure 30. Marchionini's components of browsing.....	72
Figure 31. Waterworth & Chignel's browsing activities.....	73
Figure 32. Task perspective on the browse-search continuum.....	74
Figure 33. Three browsing strategies [Carmel et al 1992].	75
Figure 34. Search strategies [Garber & Grunes 1992]	76
Figure 35. Variation of cognitive load.....	77
Figure 36. Shift of activity during interaction	78
Figure 37. Browse-search continuum - functional perspective	79
Figure 38. Pictorial representation of a frame	88
Figure 39. A relation between two frames.....	89
Figure 40. Avoiding storing picture data in a frame.....	91
Figure 41. A picture object encapsulates the information about the picture of a physical object	93
Figure 42. Objects on a path	99
Figure 43. Objects connected by hyperlinks.....	100
Figure 44. A branching path	103
Figure 45. A branching path structure	104
Figure 46. A "diversion" from a path	104
Figure 47. Components of the Generic Browser	120

Figure 48. Marked items data structures (LISP version).....	126
Figure 49. Screen linked to screen.....	127
Figure 50. Hotspots linked to screens.....	128
Figure 51. Look-up table links hotspots to frames.	129
Figure 52. Use of instances and links to form a path structure.....	134
Figure 53. The History path.....	137
Figure 54. Backtracking along the History path.....	137
Figure 55. The basic class hierarchy.....	145
Figure 56. Types of accommodation	146
Figure 57. The characteristic class.....	149
Figure 58. Zermatt's characteristics.	150
Figure 59. Application interface structure	157
Figure 60. How the user's input triggers the Generic Browser.....	158
Figure 61. Layers of the Dexter Model [Halasz & Schwartz 1990].....	159
Figure 62. The top-level structure of The World of Impressionism CD-i Disc	160
Figure 63. Hierarchy with browsers at leaf nodes.	160
Figure 64. Object linked to both location and data hierarchies	162
Figure 65. Use of instances of a picture object to store display specific data	163
Figure 66. Transition information stored within link frames	164
Figure 67. Effects between objects on a path	166
Figure 68. Linked frame structure of the picture browsers	170
Figure 69. The Structure of the Guitar Browser.....	175
Figure 70. The Ski Browser control panel.....	176
Figure 71. Cruise Browser structure.....	179
Figure 72. Cruise Browser main menu.....	180

Figure 73. Area of the World menu.....	181
Figure 74. Itinerary browser	181
Figure 75. Cruise Line menu	182
Figure 76. The World of Impressionism top level structure.....	186
Figure 77. World of Impressionism main menu.....	186
Figure 78. The Windows on Impressionism menu - the Paris option is centre of bottom row	187
Figure 79. The Paris map.....	187
Figure 80. Selecting an option on the Paris map.	188
Figure 81. Structure of the College Browser disc.....	190
Figure 82. The Location option	190
Figure 83. The State menu.....	191
Figure 84. Selecting Texas - note the reduced number of universities (82).	191
Figure 85. University Information menu	192
Figure 86. An Inheritance Hierarchy	216
Figure 87. Interleaved data	220
Figure 88. The functional units comprising a CD-i player	221
Figure 89. CD-i Plane Structure	222

List of Tables

Table 1.	Basic terminology.	22
Table 2.	CD formats.	24
Table 3.	Dimensions of browsing [Waterworth & Chignel 1991].	73
Table 4.	Resort Characteristics.....	148
Table 5.	CD-i screen sizes.....	224
Table 6.	CD-i audio formats.....	225

Context

This PhD work has been conducted part-time whilst the author has been employed by Philips Research Laboratories, Redhill (PRL). This thesis documents work which has formed a self contained part¹ of a PRL project the aim of which has been to make use of techniques from the artificial intelligence (AI) world to add value to informative CD-i² applications. The PRL Project also contributed to the Homestead Esprit project³ where an implementation of the Generic Browser work formed part of the data representation and access software which was used for trials of CD-i-based home shopping.

¹Due attribution is given wherever work not performed by the author, such as related work within the project, is described.

²Compact Disc Interactive (CD-i) is a consumer multimedia system with application software sold on compact disc.

³Esprit III Project 6789 - Homestead

Acknowledgements

The author wishes to express his sincere thanks to those who have helped and supported him during this research and its subsequent writing up.

Firstly to Professor Peter Johnson my supervisor for his advice and encouragement. Secondly to Philips Research Laboratories who have sponsored this research and especially Jenny Jerrams-Smith and Tony Weaver who got me started and Simon Turner who allowed me to continue even when it did seem to be taking forever!

To my friends and colleagues at PRL and in particular to Phil Lloyd.

To Dr Peter Kyberd I owe a particular debt of gratitude for his timely and insightful encouragement which is only possible from one who has already “been there”.

Last but by no means least a special thank you to Mum and Dad and the rest of my family without whose unstinting support and encouragement I would never have got here.

Trademarks

Apple, HyperCard, HyperTalk, QuickTime, Apple Media Tool and Macintosh are trademarks of Apple Computer, Inc.

CD-i is a trademark of Philips Electronics, B.V.

dBase is a trademark of Borland International, Inc.

INSPEC is a trademark of Hoechst Ceanese Corporation.

Java is a trademark of Sun Microsystems, Inc.

KnowledgeCraft is a trademark of Carnegie Group.

Kodak and PhotoCD are registered trademarks of the Eastman Kodak Company.

Macromedia and Macromedia Director are trademarks of Macromedia, Inc.

Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation.

Mosaic is a trademark of the University of Illonois.

Netscape is a trademark of Netscape Communications Corporation.

Nexpert Object is a trademark of Neuron Data, Inc.

OS-9 is a trademark of Microware Systems Corporation.

Philips is a registered trademark of Philips Electronics, BV.

Post-it is a trademark of the Minnesota Mining & Manufacturing Co.

Sony is a registered trademark of The Sony Corporation.

Sun is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of Novell, Inc.

Other brand or product names are trademarks or registered trademarks of their respective holders.

1 Introduction

People interact with information in many circumstances. They browse magazines, read books and search for information in encyclopædias. They go to libraries to browse for inspiration or search for information. These interactions can be loosely grouped as browse and search processes. These same processes form the basis of information-rich applications on computers. Users access information held on CD-ROMs and “surf” the World Wide Web⁴ sometimes in the hope of finding something interesting (browsing) and sometimes with a specific goal to find particular information (searching). Though research into specific aspects of these processes has been done within the fields of hypertext, hypermedia and databases, little work has been done on the theoretical underpinnings of integrating browsing and search within single applications. Perhaps this is because hypertext and hypermedia have concentrated on browsing and databases on search as these are the primary types of information access for these types of application. However there is a trend for hypertext, hypermedia and databases to converge on similar representation schemes which would make such integration possible.

The field of multimedia systems covers a broad range of applications which combine text, pictures, audio and video. There are three factors which determine the state of the art. These are the technology available, the products which exploit the technology and the research contribution. In an ideal world the research would drive the technology which would be realised as products. And indeed this is, to some extent what happens. However, technology also progresses by development (evolution if you will) and in this process the link with the foundations of research can become tenuous. Technology can advance without any more purpose than that the next step is possible. The ramifications of that advance are discovered after the event rather than predicted beforehand.

In the multimedia area this technology-led progress (if it can be so termed) is taking place in both hardware and the software. Hardware is evolving at a frantic pace. Users “need” more performance, storage, screen resolution and so on. Ever more complex applications push the limits of what is possible with the existing technology and, as a by-product, give a justification for increasing the power of the hardware - a vicious circle. It is interesting to observe the ever increasing storage and processing requirements of the operating systems themselves - whose major function is to allow the user access to his applications and files. Improved usability is often cited as the reason for the expanded facilities offered by each

⁴The World Wide Web (WWW or W³) is a global hypermedia system accessible via the Internet. Browsing information on the web is also known as surfing.

new release. However the research which could justify this expansion lags behind the drive to outdo competitor's offerings on number of features alone [Oren 1990].

It is into this maelstrom that the area of consumer multimedia has been born. Price has also been added to the equation of steering factors and has a very strong influence. Technology must be exploited to the full to support more eye catching products than are offered by the competition. Indeed the home computer segment of the PC market now leads in the adoption of advanced technology such as multimedia upgrades and even the Pentium processor [Kehoe 1995]. Software too must be produced within a limited budget. Re-use of code can help to offset the development costs and re-use of the assets offsets their purchase or production costs. In an article analysing the problems facing multimedia publishers in the consumer market and, in particular, the competition with tv, books, newspapers, and radio, [Carrigan 1991] writes

"What is required are techniques that make information browsing as compelling as watching a film or viewing a TV documentary, [...]."

Achieving the right blend of features (to sell the product) and cost effectiveness is the tightrope being walked by software developers in this technology-led field.

The aim of this thesis is to redress the balance a little by *researching* the area of multimedia browse and search with particular emphasis on the needs of consumer applications. Providing a research basis for the development of informative applications will mean that they can more readily provide the user with what he or she wants, namely information, education and entertainment.

With consumer multimedia systems a reality and considerable commercial interest in the field there is a pressing need for improvements to the multimedia authoring process. Authoring is the process of creating the application from its components (text, pictures, audio and video) and is currently very time consuming and therefore costly. In particular, the facilities for building interesting, stimulating applications lags behind the support given to the process of creating visually exciting screen designs. Support for authors is largely non-existent - the majority of current multimedia information-rich applications are developed as one-off applications using bespoke software, often programmed in a low level language such as 'C' or even assembler.

Generally available tools for creating multimedia applications such as HyperCard, Macromedia Director and Apple Media Tool have been developed with the needs of multimedia as the primary design criteria. All three allow the creation of multimedia objects (screens) containing pictures, text, audio and video which can then be displayed. Hotspots allow the user to navigate between screens. Information access is thus limited to encoding information as screens and then displaying them. Separation of the data from its screen

presentation is not provided for except by resorting to the tool's programming language (which rather defeats the object of using the tool in the first place). The particular requirements of consumer (as opposed to professional) applications have also not been addressed.

As an example consider MacroMedia Director. This tool was originally designed as an animation tool. It has a basic concept of a "stage" where "cast members" (pictures, graphics items, text, audio and video) can be displayed. This process is under the control of a "score" which choreographs the presentation. For further control including responding to mouse or keyboard events the author must write "scripts" written in Lingo, a programming language. This tool is particularly good for presentation-type applications or those with simple navigation. However the time-line used for the score becomes unwieldy for complex branching structures particularly as the relationships between items are stored in the score and not the objects themselves.

Apple Media Tool takes a different viewpoint concentrating on a structure of linked multimedia nodes. The links originate from hotspots and can embody transition information such as dissolves from one picture to another. This tool makes it simple to construct applications with one navigable structure but adding additional structures such as guided tours is more difficult. Once the basic drag-and-drop positioning of nodes and links is exceeded the author must resort to 'C' programming to accomplish other features.

The problem can be summarised as the lack of research into the functionality required to support the efficient creation of information-rich applications in the consumer domain. The creation of a generic function library for this type of application will reduce the need for expensive tailor-made software; generic in the sense that the functionality would be independent of target platform and application.

The general form of the solution to this problem can be outlined as follows:

- finding the generic primitives of browsing and search,
- investigating how browsing and search should be integrated,
- considering how the data representation and structuring needs in a multimedia browse/search application can be met.

The consumer multimedia area is interesting because of the necessity to satisfy the needs of users by providing authors with a data representation scheme and function library, based on a foundation of research (addressing the problem of technology driving multimedia software development), which can be utilised on the existing hardware that is available to

consumers for use in the home. This constrains the solution to be realistic thus capable of being exploited in the application development process reasonably quickly.

1.1 Structure of the Thesis

The structure of this thesis is, perhaps, slightly unusual in that it does not begin with a survey of the relevant research literature before homing in onto the problem to be solved. Unfortunately not much existing research (on consumer multimedia per se) was available to guide and influence this work and so in terms of defining a problem space an alternative approach has been followed. Since the technology of multimedia systems is currently driving application development it is important to look at this technology first. A problem in the technological space forms the basis for the research problem. Accordingly consumer multimedia is looked at from the technological and product perspectives first. This allows a problem area to be defined (in terms of those two spaces). That problem space is then addressed in terms of related and relevant research which allows the extraction of the research issues to be tackled.

There are two basic approaches to solving problems in this field of consumer multimedia applications. Firstly existing software is evolved to suit the new requirements. This creates many difficulties as old designs are adapted and often means that unforeseen side effects plague the development process. The second approach is to start from scratch. However there is little or no theoretical or experimental background to support this process so that it becomes basically trial and error, often a time consuming and therefore expensive process.

The rationale for using research to solve the problem, rather than the approaches just described, is that research provides a reasoned basis for the solution giving confidence that it is the right solution by knowing why it is the right solution. It is theory led and the theory can be tested. If the research-derived solution is found not to be correct when tested the model and assumptions can be challenged in a reasoned way to solve the problem. This avoids much of the error in the trial and error approach and the problems of historical decisions inherent in the evolutionary approach. The existence of research in this area will provide an alternative to the technology-led evolution of applications.

This thesis commences with some background, starting by defining some basic terminology to be used. This is particularly important as this work touches on a number of disciplines and includes terminology from the consumer area as well. An examination of the consumer multimedia domain and the opportunities within it follows. This overview sets the scene for a detailed examination of the technologies and products. The section on technologies looks in detail at the key elements available or soon to become available for

systems that will support multimedia applications. In particular the use of CD's for data storage has acted as a catalyst to consumer multimedia, exploiting the colour displays and reasonable quality audio that have also become increasingly available on modest hardware. The embodiment of this technology in actual systems exposes the constraints that are imposed by the marketplace on such systems and highlights the technology-led evolution described above. The consumer system chosen as the target platform for the implementation component of the work, Compact Disc Interactive (CD-i), is described in detail as an example of what such systems have to offer.

Having identified the information-rich applications in the consumer domain as an area lacking in research chapter 3 outlines the problem space and sets the main objectives of this thesis. This is to show that information-rich consumer multimedia applications can be built effectively by using a core function set accessing a flexible data model and integrating browse and search. The key steps to this are the definition of the data model, the devising of a method to integrate browse and search, definition of the core function set and the implementation of a prototype and its testing and evaluation. The scope of the research is also defined here to contain the research task to manageable proportions. A strategy for tackling the problems is then outlined.

The next two chapters, 4 and 5, develop the research problem by looking at browsing and search in two ways, empirically and theoretically. In chapter 4 a number of currently existing examples of consumer multimedia applications and other browsing situations are examined with a view to extracting common features. The examples studied range from printed documents to multimedia applications. In chapter 5 the fields of browse and search are studied with a literature survey which seeks to characterise them from a theoretical point of view. Various existing models are examined from three perspectives: task, cognitive and functional. After this preliminary literature review and observational studies the rest of this work has been conducted as a refinement process with a cycle of devising a model, implementing a function set, building prototype applications to evaluate the function set and suggest changes to the model or implementation. These are made and the cycle repeated.

Chapters 6 and 7 describe the solution to the problem set in the preceding chapters. Chapter 6 develops a model of an integrated browsing and search process and defines a system architecture to support it. This model draws on the empirical data derived in chapter 4 and the theoretical work studied in chapter 5 and distils this into a model suited to the consumer multimedia domain.

With the architecture and data structures defined, chapter 7 describes the development of a library of functions, based around a frame-representation, that allows information-rich, multimedia applications to be constructed. An early LISP prototype demonstrated the

feasibility of the structures and functionality defined in the model. The LISP programming environment was found to be well suited to the rapid prototyping which characterised this phase of the work. Unfortunately the very reasons for this proved its downfall as running an interpreted language on such a modest performing hardware platform as CD-i proved not to be viable. A switch was made to developing the prototype browser library, and the applications constructed to test it, in 'C'. As this new version of the library was developed, new functionality was added and assessments done. A formal testing process has also been used to test the actual functionality of the Generic Browser to ensure that each function meets its specification.

In chapter 8 the important topic of how interface issues affect the Generic Browser is tackled. Though not the main thrust of the work it is nevertheless necessary to consider issues such as the feasibility of representing the user interface in the same data structures as the browsable information and how transition effects could be associated with the links between data items.

A significant amount of work has been done building prototype applications to demonstrate the applicability of the work to real world situations. These prototypes have formed part of the evaluation process for the Generic Browser. Chapter 9 describes the prototypes and what modifications were made to the function library as a consequence of building them.

To avoid the problem of unconsciously biasing the prototypes to suit the Generic Browser, a paper study has been carried out to show how two existing CD-i applications could have been built using the Generic Browser.

These chapters form the body of the thesis. A summary chapter draws conclusions from the work done. Finally a chapter is devoted to further work that could be done based on the work reported in this thesis. This is divided into two parts, the first looks at the short-term developments possible with the Generic Browser. The second looks at more major, longer-term possibilities such as the integration of video (which is becoming increasingly important in the consumer domain).

2 Background to the Research

This chapter sets the scene for this research into providing browsing and search utilities for consumer multimedia systems. It starts by defining some of the essential terminology used in this field, a full glossary can be found at the end of this thesis. The motivation for targeting consumer multimedia is then explained and the consumer domain examined from the perspectives of the types of activity that need to be supported and the hardware and available to do them. The shortcomings of existing software and the new technologies coming to fruition suggest an agenda for the areas of research. These are further examined and developed in the next chapter where the selection of the research area that comprises this work is chosen.

2.1 What is Browsing ?

The term browsing is at once familiar and yet hard to define. Dictionary definitions emphasise a leisurely, unfocused search for information. In the computing domain browsing has come to mean the type of exploratory access used in hypertext systems, often contrasted with search as used in database retrieval. [Liebscher & Marchionini 1988] define browsing as the opposite of search. However [Palay & Fox 1981] describe browsing used as a search activity. The problems of defining browsing will be explored further in chapter 5. Suffice to say here that this thesis will show that browsing and search are better defined as the extreme ends of a continuum rather than as two separate, individual access mechanisms. As one moves from browsing to search, the focus becomes more directed and the activity, perhaps, less leisurely. To avoid the confusion that exists because of the ambiguity of the terms browse and search, within this thesis the term browsing is used to describe the activities that lie on that continuum.

2.2 Basic Terminology

At this point it is necessary to define the basic terminology to be used in the remainder of this thesis. The key terms are defined below. Other terms are defined in the glossary.

asset	the multimedia objects such as pictures, audio material, text and video which are the basis of a multimedia application.
browser	a convenient shorthand for “browse and search mechanism” as in “Generic Browser”.

digital video (DV)	full-screen, full motion video, usually MPEG (see below) encoded to VideoCD (white book) specification. Sometimes also referred to as fmv.
Generic Browser	the function library described in this thesis is known as the Generic Browser. The reasons for this name are largely historical - like LISP's CAR and CDR ⁵ . The usage of the term browser should not be taken as a true description of the total functionality of the library which comprises aspects of data representation, browsing and search.
link	a directed arc joining two information nodes; often used to represent a relationship between concepts.
MPEG	Motion Picture Expert Group, the ISO standardisation committee that has defined the MPEG 1 digital video encoding/decoding standard [ISO 1993a, ISO 1993b, ISO 1993c].
multimedia application	one which combines more than one of the media types text, pictures, audio and video
navigation	the process of moving through an information space in a non-random way.
node	a chunk or unit of information; often used to represent a concept.

Table 1. Basic terminology.

2.3 Consumer Multimedia

Multimedia has been around in the professional arena for a number of years. It did not take off as fast as some of its most ardent protagonists such as Apple Computer might have hoped, but CD-ROM drives are becoming standard features of PCs and so the

⁵These two functions which return the head and tail of a list respectively, have names derived from the architecture of the machine on which they were first implemented viz. CAR - contents of address register and CDR - contents of decrement register.

necessary high capacity storage devices have become common and support for audio and video is increasingly becoming a standard feature of mainstream workstations.

Multimedia in the home is another matter entirely. There are a number of key issues which differentiate this market from the professional one:

Constrained hardware environment - the consumer market is much more sensitive to price than the professional one. As a consequence computer based systems for the consumer market are generally of more modest performance (in terms of processing power and memory capacity). The lower purchase price of consumer systems requires a commensurate higher volume of sales and reduced cost of production.

Low computer literacy - consumer multimedia systems must avoid alienating those people who are not computer literate and for whom a computer is regarded as difficult to use and unfriendly. This is often achieved by avoiding calling the system a computer and eliminating the keyboard. Interaction is then performed with joysticks, trackballs, buttons and similar devices.

Low tolerance to poor response times - because the majority of consumers don't know (and don't want to know) how a multimedia system works they are more critical of poor response times.

No requirement to use - consumers do not *have* to use their multimedia systems. These systems must give their users pleasure or fulfil some other need. (Professional users generally do not have the choice, use of the system is part of their job or the only way they can find information). Consumer systems are in competition with existing uses for a person's leisure time. Possibly because use is optional, consumers do not tolerate uninteresting or difficult to use software.

Because of the price sensitivity of the consumer market, it is not as easy to pass on high application development costs to the purchaser so there is a need to make application production as cheap as possible hence tools which assist in the authoring process or which allow re-use of code or data are in demand.

2.4 Enabling Technologies

Critical to the success of multimedia as a "new publishing medium" are the technologies which are allowing it to expand out of the mainstream computing, data storage and retrieval environments and into the mass marketplace.

2.4.1 Compact Discs for Data

Although video disc players have been connected to PCs for interactive video (training) applications, the total cost of these combined systems is high, as is the cost of mastering the discs. These systems have therefore remained in the professional market.

Compact disc was originally developed as a cheap medium for storing digitally encoded audio but it did not take long for its potential as a storage medium for any digital data to be realised. CD-ROMs were the first use in the computing domain, holding 600Mb of data including programs. Extensions and alternatives to the way the data is laid out on the disc have lead to a number of alternative formats.

Acronym	Standard	Description	Date
CD-DA	Red Book	Audio CD standard	1980
CD-ROM	Yellow Book	Computer Data	1984
CD-i	Green Book ⁶	Multimedia data (interleaved)	1987
CD-R	Orange Book	Multi-session recording to disc (Photo-CD)	1992
VideoCD	White Book	MPEG 1 coded linear movies	1993

Table 2. CD formats.

However the basis of all these standards is the same - the CD provides a cheap storage medium for about 650Mb of digitally encoded data. Riding on the back of the music CD industry's economies of scale means that the infrastructure for CD publishing is in place.

2.4.2 Cheap Platforms

As has already been observed the consumer is very conscious of the cost of new "gadgets" and so it is only with the advent of *cheap* multimedia platforms that this market has opened up. Some of the contenders in this field are briefly examined below to give the reader an idea of the capabilities of the target platforms of this work.

⁶The Green Book CD-i standard defines not only the disc format but the hardware and operating system as well.

CD-ROM - not really a platform in its own right, rather an add-on to existing platforms (Macintosh, PC etc.). This is the basic building block on which multimedia is based - cheap, removable, robust, high capacity storage. Though initially only available on high-end machines there is a trend to incorporate a CD-ROM drive in machines aimed at the home market. CD-ROM provides the minimum for multimedia in the sense that whilst it does give 600Mb of storage there is no interleaving of data as there is on CD-ROM XA and CD-i disc. This means that, for example, music data cannot be played off the disc whilst a picture is loading.

Because personal computers have been designed as just that, computers, their multimedia capabilities are generally barely adequate. They usually have poor internal audio capabilities (in the case of PCs requiring an add-on card), modest video capabilities with only one plane and often only 8-bit CLUT graphics. But the unrelenting progress in the pc industry means more memory and processing power are available every year so a brute force approach to, say, screen handling has overcome most of the disadvantages of not having an architecture specifically designed for multimedia with add-on cards supplying the missing functionality.

The large computer manufacturers such as Apple and IBM are always *nearly* in the consumer market. But with CD-ROM equipped machinery costing £1500 and more it is difficult to see multimedia versions of personal computers penetrating outside the traditional home computer market. However that market is an ever expanding one. About 7 million multimedia equipped PCs were sold for home use in 1994 [Cortese 1995] and CD-ROMs are now sold like computer games or audio CDs and even have a "top ten".

CDTV - essentially a Commodore Amiga with a CD-ROM drive, no keyboard or other disks. Released in 1991, about the same time as CD-i, but with much less development. A lack of innovative multimedia software saw this machine fail to penetrate further than the home computer/games machine market where it has become a peripheral for existing Amiga users. Indeed, Commodore have now (in 1994) released a successor to CDTV the CD-32 (a games console).

CD-i - a consumer multimedia machine designed with this market as its target. CD-i is built around the "Green Book" standard [Philips & Sony 1990]. Modest memory and processing power betray its mid '80s origins. CD-i offers interleaved multimedia data so that, effectively, more than one file can be read concurrently from the disc. This allows music to be played from the disc whilst a picture is being loaded for example. There are two video planes allowing a range

of video effects, a variety of picture coding formats from CLUT to tv-quality allows the use of photographic (natural) images as well as “computer graphics” and finally high quality stereo audio.

Nintendo and Sega - although producers of computer games machines, both these manufactures have recently produced CD-drives as add-ons to their top of the range games machines (these use CD-ROM format and are consequently slow). It will be interesting to see if they branch out from their games market niche and into the multimedia arena proper.

3D0 - this machine is aimed much more at the games market than CD-i (which was developed before the dramatic rise in the fortunes of Nintendo and Sega made computer games a “legitimate” market to be in). Custom chips allow fast video processing promising excellent special effects. It is a similar story for the other recently (1995) released platforms such as the **Sony Playstation, Atari Jaguar, Sega Saturn** and ... - more power, more realistic graphics but a concentration on games and a dearth of other software.

Though not an exhaustive list this gives a good impression of the type of hardware available for consumer multimedia. To sum up, this market has shown no signs of adopting a single platform. Some companies have stayed with a standardised design and concentrated on the addition of digital video and the production of a large software library. Others, keen to cash in on the games machine bonanza are producing ever more powerful games consoles which now include CD drives as part of the specification. In all cases these machines undercut the prices of multimedia equipped PCs but there is now a fast growing market for consumer software for PCs as well.

In spite of the increasing processing power of current consumer multimedia machines, with the exception of personal computers these improvements are largely aimed at better graphics processing rather than information processing. The computing environment remains constrained with modest amounts of memory and slow CD drives for mass storage. (Only very small amounts of non-volatile storage are available, principally for the storage of game high scores and saved games.)

One key observation is that in spite of the volatility of the hardware market there is an ever increasing interest in producing multimedia software. Cross-platform code development is a must for software houses so that they can “hedge their bets” while the hardware manufacturers try to carve out a niche for their particular combination of the basic multimedia components.

2.4.3 Digital Video

As the whole concept of multimedia developed in the late '80s it became increasingly apparent that multimedia which consisted of just static pictures and simple animations was in a sense incomplete. There was a need for such systems to display moving pictures i.e. video. Hooking-up laser disc players or video tape machines was not the real answer. First on the scene was DVI (digital video interactive) from Intel. This system offered real-time decoding and either low quality "edit level" video for prototyping which can be coded on the user's PC or higher quality material coded by Intel. DVI also required an add-on board for the PC. The relatively high cost of a DVI equipped PC has meant that this system did not really take off outside limited professional uses.

Apple Computer, long time proponents of multimedia, have developed QuickTime [Apple 1993a, Apple 1993b], an operating system extension to integrate video at the system level on Macintosh and also PC. This can use a variety of compression/decompression systems. Apple supply a number of software codecs with QuickTime which have the advantage is that they do not require an add-on board to display QuickTime movies. However, needless to say, a powerful Macintosh is needed to display even a small (i.e. part screen) image at a reasonable frame rate.

Meanwhile ISO's Motion Picture Expert Group (MPEG) were working to produce their standard for digital video compression/decompression [ISO 1993a, ISO 1993b, ISO 1993c]. Philips backed MPEG for CD-i and in late 1993, after much delay, released MPEG decoders for CD-i. Decoders have also been promised for other platforms such as 3D0 and the Atari Jaguar. Apple have specified that QuickTime 2 will support MPEG via add-on hardware [Apple 1995 pp. 9-10].

One target area for digital video is the home video market. As well as being playable on CD-i systems, MPEG encoded discs with feature films on will be playable on dedicated digital video movie players. Because the cost of producing CDs is much less than that of duplicating video tapes (by a factor of three) it has been suggested that the major home video suppliers will kill-off VHS pre-recorded movies in favour of digital video CDs. This would reinforce MPEG as the standard for digital video coding and reduce the costs as volume forces the encoding price down. Integration of digital video with "static" multimedia will become expected by consumers.

2.4.4 Consumer Applications of Multimedia Technology

As well as the "traditional" informative applications such as encyclopædia discs or museum guides there are several new avenues opened by a "mass market" for information-rich multimedia.

Home Shopping

There is a growing interest in the possibilities offered by multimedia systems for shopping from home. Major catalogue companies spend large sums of money on printing and distributing bulky catalogues. Putting such a catalogue on a CD would reduce these overheads. So the first step in home shopping is the catalogue on a disc. The real challenge is to explore how multimedia can be exploited as a sales vehicle which was the area being investigated by, for example, the Esprit Homestead Project⁷. A catalogue with digital video might be more like tv advertising. Search facilities would make finding the right item to buy more easy. A modem connection would allow orders to be placed on the retailer's computer. Prices and availability could be transmitted to the consumer, superseding information held on the disc.

Magazine Discs

In 1991 Computer Graphics World ran an article on Interactive Magazines [Maguire 1991]. This explored the potential for the nascent CD-based magazine market, concentrating on CD-ROM as a vehicle for distribution. Maguire looked at three magazine discs: Macworld Interactive, Nautilus and Verbum Interactive. He was impressed with the quality of these three offerings but felt that the bottom-line was (in 1991) that the home market was "a high stakes long shot" though there was potential in the professional market with for example magazine CD-ROMs for doctors extolling the virtues of new drugs and treatments. The main obstacle to take-up was the lack of CD-ROM drives. Most PCs sold for home use now have CD-ROM drives [Kehoe 1995] so all that is required is a sufficiently cheap authoring mechanism so that magazine applications can be produced on a regular basis.

A very important characteristic of magazine applications is their transient nature - they are a throw-away commodity. This requires that they must be produced at low cost which places even greater demands on the authoring tools. For such applications common structures and interface elements are important from two perspectives; firstly, from the reader's viewpoint a "house style" allows familiarity with the layouts to be built-up as successive issues are read; secondly, from the authoring perspective new structures carry an implementation overhead - indeed most paper magazines keep to a well defined style for many issues with revisions every year or so to keep the style fresh.

CD-i World, the first CD-i magazine disc, used MediaMogul for authoring. MediaMogul is a fairly simple tool allowing the creation of screens linked by hotspots. The authoring

⁷Esprit III Project 6789 Homestead had the objective of investigating the potential of home shopping from a CD based application and conducted extensive user trials with a CD-i based system.

system consists of a spreadsheet interface to the sequencing of the screens and a hotspot editor. Screens can show pictures or video and have audio associated with them. More complex structure could be produced with a better tool. There is a need for a high level representation scheme to allow authors to create content structure and specify interaction possibilities, minimise programming and reduce cost to allow magazine discs to be cheap enough to be marketable as throw-away commodities.

Networked Multimedia

Although multimedia in the home currently revolves around a single, unconnected machine, the burgeoning high bandwidth communication paths to the home via satellite, cable and (digital) telephone links, the so-called “information superhighway”, mean that new avenues based on the idea of networked multimedia are coming to the fore in the multimedia world. The range is enormous. Home shopping might allow downloading of up-to-date prices and merchandise availability together with order placement direct from the box in the lounge, even browsing around a virtual store. Multimedia, multiplayer games (e.g. [Crawford 1993]), pay-as-you-view, video-on-demand (e.g. [Little et al 1993]) and so on. Many of these services will require data storage, structuring and access tools. For instance video on demand will need a database of movies available and may offer information such as reviews, plot descriptions and cast details about these films.

Although in networked multimedia the servers will be powerful machines and so not subject to the tight performance constraints of machines in the home they still share the need for the structuring and access facilities for multimedia information. Thus research directed at domestic multimedia platforms is applicable to this area as well.

2.5 CD-i - A Typical Consumer Multimedia Platform

An important objective of this work is to demonstrate its suitability for implementation on the types of multimedia machine sold for use in the home. To this end all the implementation work has been carried out on CD-i hardware. To give the reader a better appreciation of the capabilities of such platforms the key features of the CD-i player are listed below. The CD-i hardware, disc format and system software are defined in a standard document known as the “Green Book” [Philips & Sony 1990].⁸

⁸For further information on the CD-i hardware and the CD-i authoring process see also [Philips IMS 1992a, Philips IMS 1992b & Philips IMS 1992c].

Key features of CD-i

650 Mbytes of data on CD disc. Interleaved format allows more than one stream of data can be read concurrently.

10MHz 68000-based CPU, 1MB RAM.

2 video planes for CLUT graphics or DYUV coded natural images, a background plane for digital video and a cursor plane.

Digital video option decodes MPEG 1 coded video onto the background plane. An additional 1Mbyte of RAM is also available.

High quality stereo audio.

Real-time, multi-tasking CD-RTOS operating system based on OS-9.

A more detailed specification of the CD-i system is given in appendix A.

2.6 The Research Context

This PhD work has been conducted part-time whilst the author has been employed by Philips Research Laboratories, Redhill (PRL). This thesis documents work which has formed a self contained part⁹ of a PRL project the aim of which has been to make use of techniques from the artificial intelligence (AI) world to add value to informative CD-i applications.

This environment has imposed some constraints. Firstly there was a Project need to build prototype applications early in the work. Accordingly this was done in parallel with the theoretical investigations rather than delaying the prototyping activity until after the problem space had been fully defined. However the fact that the problem space was poorly defined initially meant that this prototyping helped in the initial exploration that characterised the problem space. The choice of the CD-i platform (see below) was clearly influenced by the Philips connection. Similarly total freedom of choice for implementation language was not possible because of the requirements of the PRL Project. However where constrained choices have been made the limitations of the choice are recorded.

⁹Due attribution is given wherever work not performed by the author such as related work within the project is described.

2.6.1 The Choice of CD-i

At the time that this research was started the CD-i platform was the only multimedia system specifically designed for use in the home (not counting game consoles that were dedicated to game use). CD-i was chosen because it was felt that it was important to address the constraints such a system (as opposed to a PC or Macintosh) would place on the work. As the work has progressed this choice has become less clear cut because PC's are now very common in homes in the US and could be regarded as consumer multimedia. Such platforms pose slightly different problems to CD-i. The display and audio architectures are not "tailor made" for multimedia and pure CD-ROM requires very careful use of the CD filesystem to avoid unacceptable delays. However more memory and a writable disk coupled with sophisticated programming environments might make such machines a good choice for this type of research if it were to be started now.

2.6.2 The Film Browser

The Generic Browser concept grew from experiences gained during the development of a browser implemented on the CD-i platform. This browser, The Film Browser, was derived from an existing browser, the Cookery Browser. The Cookery Browser was designed and developed in the PRL project, the application code being written by David Walker [Walker 1990].

The Cookery Browser allowed the user to view recipes for the four courses of a meal. The user browsed with next and previous buttons, could mark¹⁰ items of interest, heard a short audio commentary for each dish and could select whether he saw a picture of the dish, the recipe, a description or the ingredients. An index was also provided for rapid access to the dishes arranged by type e.g. fish dishes, meat dishes and so on.

The Film Browser¹¹ [Cole 1991c] gave similar functionality but in the domain of feature films. In this browser, the user could see a still picture from the film, a film clip, a review and a cast list. The index allowed access by genre. Again marking was available with either all the films, all the films in a genre or all the marked films being available to browse.

¹⁰Marking is the attachment of annotations to an information item.

¹¹The author's contribution to the Film Browser was the modification of the Cookery Browser code to the film domain and the building of the actual application. The screen designs, asset production and voice-overs were contributed by other members of the project

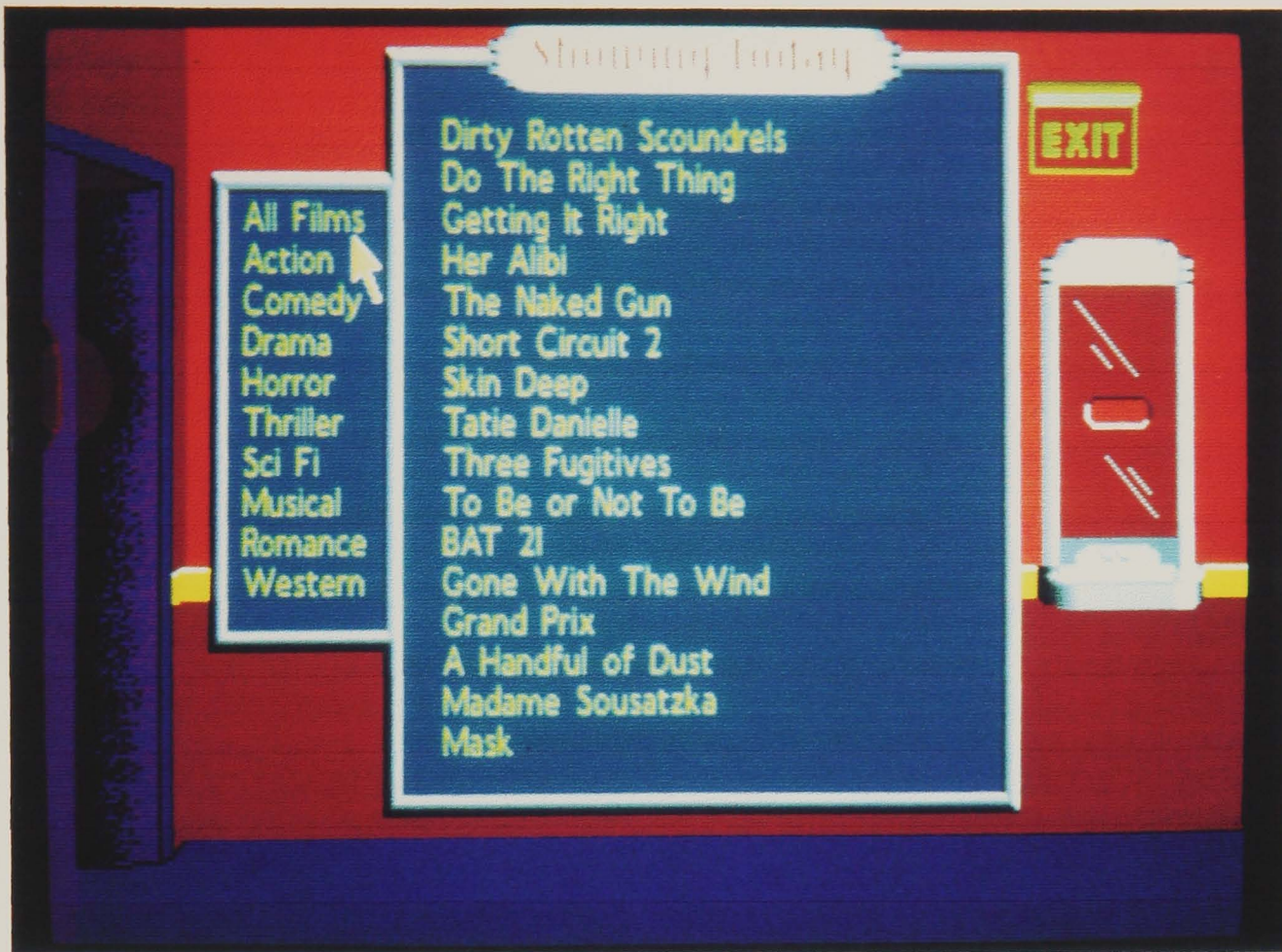


Figure 1. Film Browser - access by genre and title.



Figure 2. Film Browser - browsing screen.

Four key features of the Film Browser are listed below:

- browsing (navigation)
- marking
- multimedia data
- subsets

Functionally these two browsers are very similar and it was proposed to simply change the data being used by the Cookery Browser together with the graphics in the interface to achieve the desired functionality in the Film Browser.

Following the completion of the Film Browser the following observations were made:

- the same basic browsing functionality was used in two different applications and further thought suggested that these applications were by no means unique in their requirements for browsing facilities

- modification of an existing 'C' program, which had not been written with such modification in mind, was a time consuming task. The resulting new application used the same data structuring and access functions with a different look and feel to the user interface

- extending the Film Browser to allow more genres was constrained by the representation used (in this case the bits in an integer). A more general representation scheme would allow extensions without major modifications to the code.

Two approaches which avoid some or all of these problems suggested themselves.

Firstly a browser "shell" application could be written with reusability as a design requirement. Many of the code modifications to the Cookery Browser were as a direct consequence of reuse not being considered. Examples range from application specific variable naming to inflexible data structures.

Whilst this approach works well when the applications are all very similar the widespread use of browsing interactions in multimedia software (as demonstrated later in this thesis) suggests a more general approach.

The predicted widespread use of browsing in the context of the growth of consumer multimedia described in this chapter further adds weight to the decision to pursue this concept in preference to that of browser shells.

One issue raised but not addressed by the work on the Film Browser was that of the use of search. For example in a Film Browser-type application it would not be unreasonable for the user to request to see information on all films by a certain actor. This would require the application to search for these films. Hence the position of search with respect to browsing is an issue for information-rich application development.

The Film Browser provided the trigger to investigate browsing interactions with the aim of specifying a generic framework for multimedia browsing. Such a framework consists of both a representation scheme for the data to be browsed and the functions required to access and manipulate the data within the context of the browser application. As this investigation progressed the importance of search to multimedia applications became apparent. Thus search was added to the feature space to be addressed by the generic software.

2.7 Multimedia Authoring Needs

Authoring is the process of turning raw assets into an application. Whilst multimedia is a relatively young field it is growing fast (as the number of CD-ROM applications testifies). As the production of applications moves out of the professional marketplace the need for design and creativity in the authoring process increases. As all the entrants into the multimedia market have found, it doesn't matter how gee-whiz the hardware is, it is the software that makes the money. Consumers do not *have* to buy multimedia software. Richard A. Bowers, executive director of the Optical Publishers Association said in 1991 "What's going to make multimedia - or CD-ROM go - will be people who have the content, and the creative and business chutzpah to pull it all together and really make this new medium" [Maguire 1991 p. 40]. The major players in the multimedia market are buying asset owners such as film and record companies e.g. Sony owns CBS records, Columbia and Tristar Hollywood studios, Philips owns 70% of PolyGram which owns, among others, Propaganda Films and Matsushita owns MCA (Universal Studios).

When CD-i was being developed there was a critical shortage of suitable authoring software and so applications were developed by teams which included programmers skilled in 'C' or even assembler. This meant that application build costs were very high. Considerable effort was (and is) being expended to provide tools that increase productivity by reducing the amount of hand coding that is necessary and as a consequence the level of testing required to prove the application before release. This type of bespoke software development is very costly. At first sight it might seem that code re-use and portability are alien to multimedia and to a large extent this is the case at present, particularly on dedicated home machines. The alternative is to make use of application development tools. However

there are only a few multimedia authoring tools available for the main platforms (Macintosh and PC).

Those multimedia authoring tools that do exist concentrate on tying together assets in the form of screens. This is essentially based around a slideshow presentation type of mechanism. Interaction is often limited to switching between presentations. Examples of this type of tool are:

MediaMogul for CD-i - in MediaMogul the fundamental objects are scripts which control the presentation of assets and menus which allow branching between scripts. There are no data handling facilities at all. Within this rather constrained environment a range of interesting applications can be built. Good examples are the CD-i World magazine discs¹². The lack of any data handling facilities within MediaMogul *is* restrictive however and a number of authors have requested such functionality to be added. One route for providing these features is to use the MediaMogul "plug-in" which allows extensions to the basic MediaMogul system¹³.

The next step up are Hypercard and Macromedia Director (similar products are also available for PCs) which allow multimedia, interactive presentations to be assembled. These allow complex program structure but require the use of programming languages to achieve this. However, even in Macromedia Director there are no database facilities for structuring and accessing data.

There are not a great many tools available to assist the author of a multimedia application which contains a large amount of information i.e. where accessing the information is the purpose of the application. There are database and spreadsheet packages for PCs (though few, if any, have been ported to dedicated consumer multimedia boxes such as CD-i) which allow raw information to be stored, retrieved, and to a very limited extent displayed. The principal problem with the use of databases is their inflexible representation.

What is lacking is the ability to put data at the heart of the authoring process rather than presentation. The research issues here are: what does it mean to have data-centred authoring? What are its advantages and disadvantages compared to the location/presentation model currently in use? Can the two coexist or be integrated?

¹²The CD-i World discs are published by Parker Taylor and Company to accompany the CD-i World Journal.

¹³A MediaMogul database search plug-in has been developed using the Generic Browser search functionality described in this thesis.

2.8 Research Opportunities

The prerequisites for consumers to have access to multimedia are now in place. There are a number of competing platforms in the marketplace. Of course the success of these ventures are dependent on application producers giving the public applications which will interest, amuse and inform them. To do this economically they will need tools to streamline the production process and before tools can be produced the representational and functional underpinnings must be developed.

The domain of consumer multimedia has been chosen as the target for this research because it offers a number of interesting challenges. Firstly it is not an area that has been as heavily researched as, for example multimedia for teaching purposes where entertainment is an aid to learning rather than the primary purpose of the application. Secondly there are a number of constraints that this market places on the hardware. This in turn impinges on the design of the software components from which the application is constructed. The challenge here is to provide the *right* support to enable good application software to be written. In a relatively unconstrained environment many alternatives can be offered to an author without seriously prejudicing the performance of the application. In a constrained environment a more careful choice of options is required. It is also an area which is fast evolving - de-facto standards last only a year or so. This means that to be useful software must not be tied to a particular platform. The consumer industry's preoccupation with "the bottom line" i.e. minimising costs is also very relevant because this provides a driving force for more efficient and productive software and tools. The research issue here is the production of techniques and methodologies which can be implemented within these constraints. In particular the area of informative applications (so-called edutainment) is poorly served from the authoring standpoint with little or no support for information structuring for example.

The Film Browser has shown key characteristics which are also shared by its predecessor the Cookery Browser. If the hypothesis that these have a wider applicability to a range of multimedia applications is proved then providing these as a function library for the application developer avoids re-inventing this particular wheel each time such an application is built. Preliminary investigation showed that a range of multimedia applications share common information access primitives such as browsing and marking items of interest and searching for particular information. Together browsing and searching form an important part of the field of information access. In the world of CD-i that information is multimedia in nature with textual information forming only part of the range which also includes audio, video and still pictures. The additional requirements that multimedia data impose on the representation and processing functions of a system need to be explored in detail.

The essentials of application creation can be summarised as follows:

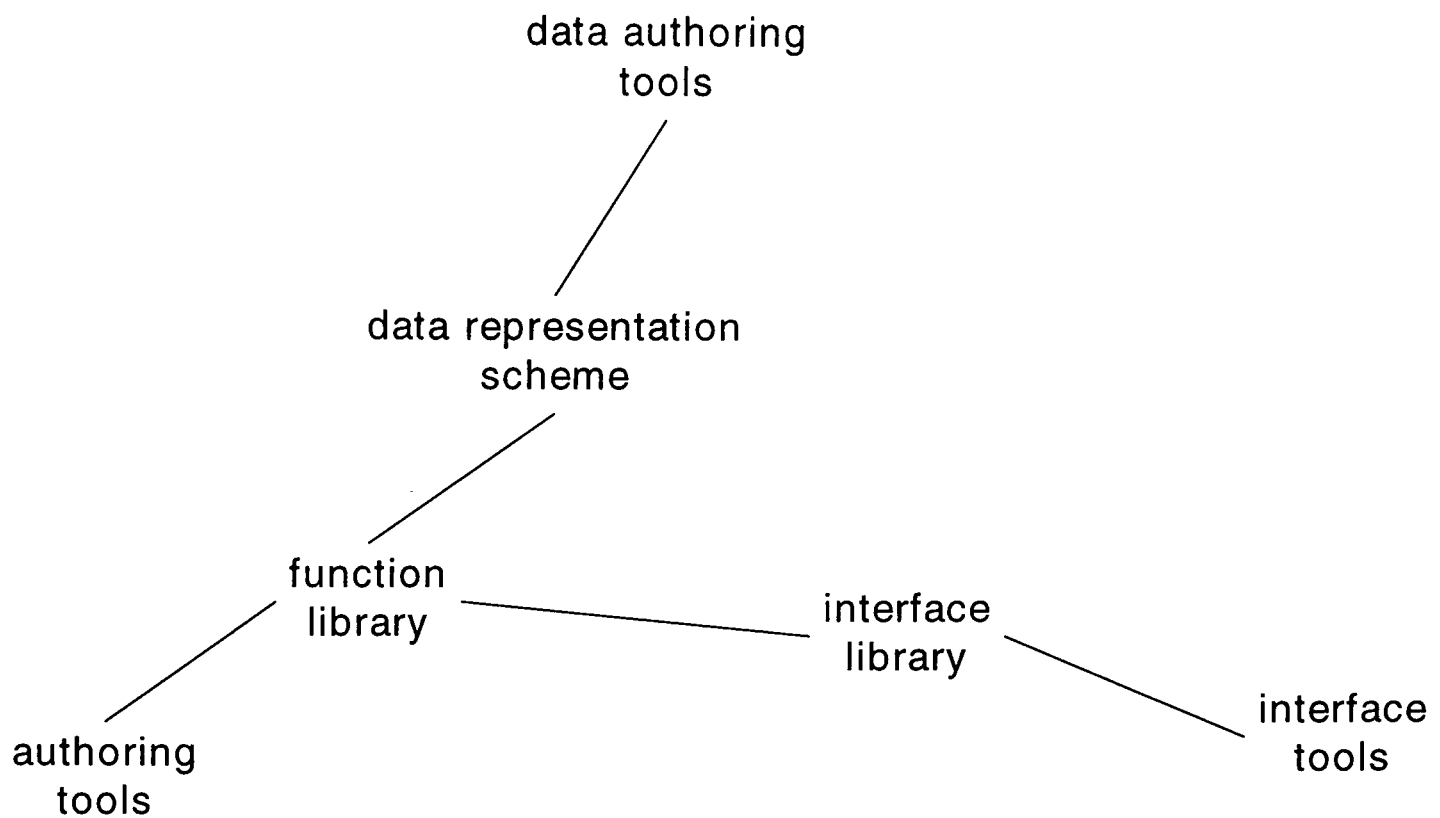


Figure 3. Components of application creation.

Most current applications have very little data other than screens of information so that the data representation is simply a means of storing these screens and the data authoring tools are paint and draw packages. The function library allows screens to be presented and the interface library translates user input from buttons and menus into screen selections. Authoring tools like MacroMedia Director include interface creation.

In information-rich applications the data may not be stored as screens but as pictures and text and audio fragments. Structural information such as hyperlinks and other forms of browsable structure form part of this data. So the data authoring process now adds data structuring to asset creation, manipulation and storage. The interface handles the presentation of this data on the physical screen as well as user input and the function library gives access to the data objects and traverses the structures.

For information-rich applications, a combination of an effective interaction layer on a rich data representation should enable interesting and stimulating applications to be created. The key is to address the problem of making the information easily and enjoyably available to the user via a mechanism that makes application creation straightforward and quick for the author. The first step towards this goal is to find the functionality that consumer software requires so that tools can be built.

2.9 Research Agenda

The first area in need of research is the provision of generalised browsing and search facilities on the “modest” hardware of consumer multimedia machines. This requires small, efficient codes to be developed with the variety of platforms causing support for cross-platform application development to be of increasing importance.

Underpinning authoring tools for such hardware will be data structures and access software. The need for an emphasis on browsing access to information, but with search facilities integrated at this level makes the needs of this environment stretch most existing models developed for the professional database application area. Accordingly there is a need to develop a suitable data model for the multimedia information that is at the heart of a wide range of potential consumer applications and to extract a rich set of access functions.

With the designer playing a key rôle in the development of consumer multimedia applications and the current bias towards a “location centred” view¹⁴ of the structure of an application there is a need to examine how information-rich applications can be supported by integrating an information centred viewpoint.

Of growing importance to the multimedia world is the integration of video with other multimedia assets. This means that the problems of representation of the video content must also be addressed together with the integration of presentation and how the inclusion of video affects the current application models and the authoring process.

Another area gaining prominence with the advent of networked multimedia and the “information superhighway” is coping with large volumes of information. Structuring and access methodologies are important here.

These are the research areas closest to the domain of multimedia browsing and search. In the next chapter the topics chosen for research in this thesis are described in more detail.

¹⁴A location centred view regards an application to be composed of a number of locations (screens of information), for example HyperCard’s cards. A user navigates by following links between these locations. Structure in the data is secondary to the navigation structure.

3 Objectives, Scope and Strategy

In this chapter the objectives of this thesis work are laid out in detail addressing research issues raised in the previous chapter. Following the objectives the scope of the work is presented, differentiating between the core topics of the work and those areas that are addressed only as far as is necessary to support the main thrust. The chapter concludes by explaining the strategy employed to do this research.

3.1 Objectives

The primary objective of this thesis is to assess the feasibility of providing a core set of information access functions operating on a flexible data model as a basis for the construction of information-rich multimedia applications in the consumer domain. Such a set of functions should integrate both browse and search activities to allow both access strategies to be provided in one application. To assess the feasibility a generic function library for creating information-rich multimedia applications that include browsing and searching access to the information is developed. This addresses the observed lack of information-based authoring facilities for multimedia applications reported in the preceding chapter.

In the context of this work the term generic is used to indicate the generality of the software from two perspectives; generic with respect to platform and secondly with application.

Platform independence means that the software is not tied to a specific platform, it does not depend on specific platform features or is written in such a way that such features are accessed through drivers which isolate them from the rest of the code. These drivers can then be re-written for different platforms without the main portion of the code needing to be modified. Chapter two has shown the rapid evolution of multimedia hardware and this necessitates a machine independent approach. However the constraints of the current generation of hardware have been used to ensure that the resulting software is realistic in the demands that it places on its host system and hence that the specification is implementable.

Application independence means that the browse/search software is not specific to any particular application. There is a trade-off between bespoke software solutions and the generalised approach preferred here. Bespoke software allows maximum efficiency as the data structures and functional code are all specific only to the tasks required by one application. However the needs of the multimedia publication industry namely speed of application creation and maximum code re-use (to spread the costs over a number of discs and allow sequel/related discs to be produced easily) has led to the use of “engines” for particular classes of disc. An example is the series of art discs for CD-i such as “The

World of Impressionism” disc described in detail in chapter 9. These all share a basic browsing engine. The natural evolution of the engine approach is to make the engine general enough to be used for a number of different types of application (but all within the same basic class of application). This is the direction taken by the Generic Browser.

There is a conflict between the goal of generality and that of efficiency and implementability. The more specific the solution the more tailored it can be, incorporating only what is necessary to solve the problem and no more. Experience in the AI world highlights this conflict. Early (1980’s), general-purpose AI toolkits such as KnowledgeCraft and KEE were extremely flexible allowing researchers to concentrate on the AI research issues. The price of this flexibility was that the toolkits were very large and inefficient. As AI techniques have been absorbed into mainstream computing the toolkits have been pruned down reducing the choice of knowledge representation methodologies for example, enabling them to be implemented on PC-sized machines. An example of this process is Nexpert Object [Neuron Data 1990].

Within the overall goal of generality there are a number of key subgoals:

3.1.1 Development of Multimedia Data Model

A data model is important because it provides a secure foundation for the function library. The model should be common to both the browsing and search functions. This allows both processes to operate on the same data structures with integration a fundamental property. The alternative where separate browsing and searching sub-systems are used can require data to be passed back and forth and overlapping or duplicated representations of the information.

In the following chapters the processes of browsing and search are investigated together with their data representation and structuring requirements. A suitable data model is then proposed. This model underpins the development of an architecture for browse/search applications. It has the following features:

The data representation is object based and is suitable for multimedia data.

The data model handles browsable structures of the following types :

linked structures typified by “hyperlinks”,

paths - for representing sequences or narrative,

subsets.

It demonstrates a uniformity of representation between the basic building blocks of multimedia applications:

data items

search queries

search results

histories

paths

The model allows the richly interlinked structures present in a wide variety of multimedia data and provides additional structuring for sequence or narrative.

3.1.2 Integration of Browsing and Search

Evidence is provided for the hypothesis that browsing and searching are the extreme ends of a continuum of data access methods and that many browse/search interactions contain a mixture of both with the ratio varying as the interaction proceeds. These observations motivate the integration of browsing and search facilities so that applications can be built which exhibit both types of information access.

By exploiting the concept of marking the data items as a means of creating browsable subsets the integration of search by marking search “hits” is demonstrated.

3.1.3 Development of a Core Function Set

The selection of a core set of browsing and searching primitives which operate on the data held in the structures developed above is described.

At the browsing end of the continuum there is an emphasis on extracting the basic, simple browsing functions. Syntactic (location) and semantic (content) navigation are both considered and their requirements distilled.

For search the emphasis is on determining what variations on the information retrieval theme are needed for the consumer domain. This results in facilities for inexact specifications and partial matches being included.

As mentioned above, marking plays a critical rôle in this work. Both from the author/user perspective where it is a valuable and necessary function, but also from the perspective of its use as the means to integrate browsing and search facilities.

3.1.4 Implementation and Evaluation

The second thread of the thesis describes the construction of an implementation of this architecture and its evaluation. These two activities have formed a refinement loop where the model has been adjusted in accordance with the results of the evaluation of early prototypes of the architecture.

The implementation of the browser runs on the CD-i platform and on Sun workstations and has been constructed as a library of 'C' functions. The constraints of targeting the work at consumer multimedia helps to ground the research in a real-world environment leading to immediately applicable results.

The browser has been evaluated by the construction of new applications demonstrating features of the work documented in this thesis. An example of how the Generic Browser functions could be used to re-implement existing browsing applications adds further confirmation that the generality goals have been met. The use of the prototype applications to refine the Generic Browser specification is documented. The description of these processes serves two purposes: firstly it demonstrates how prototypes were used to refine the specification of the Generic Browser and secondly that the Browser meets its objectives.

The successful implementation and testing of the generic software described herein would make the task of developing a tool for authoring information-rich multimedia applications realistic, though for such a tool, further studies of the authoring process would be required. This tool could become the Macromedia Director of information-based application development and a core function set is an important first step.

3.2 Scope

At this point in this thesis it is necessary to define the scope of the work to be addressed. An outline of the areas to be studied will be presented and the boundary lines drawn. In a number of instances it will be necessary to "peek over the boundary wall" and examine important related areas. In these cases the depth of the examination will be restricted to that required to provide the context for the main thrust of this work.

In the preceding chapter the following agenda of research topics was formulated:

- generalising browsing and search facilities

- provision of such facilities on consumer multimedia systems

- integration of search with browsing

data models for multimedia

integration of information with the “location centred” model used by existing authoring systems

integration of video

networked multimedia

As we have just seen the objectives of the thesis tackles four of the research items on this agenda. These are generalised browse/search facilities, targeting consumer platforms and requirements, the integration of browsing and search and devising a suitable data model.

The other topics are peripheral to the main goals of the work and are not given the same depth of treatment. One possible method for integrating a data centred view of application design with the common location centred one is proposed. The use of video in information-rich applications is a field for research in its own right. This thesis treats video as a self contained asset type which allows its inclusion within the framework of the Generic Browser in the form of simple video clips. A line is drawn between video at this level and applications which are primarily video. The latter require a different model, perhaps based around television production and techniques. Networked multimedia is not addressed directly. However the main differences between networked multimedia and disc-based multimedia are the larger data capacity, faster access and processing speed of a network server and the interaction delays caused by the network. The type of data structuring and access facilities described here are therefore equally applicable to this domain.

There are two standpoints from which the area of browsing can be examined:

Firstly there is that of the underlying functions that determine the nature of browse and search possible. In this view of the browsing world, the user interface (look and feel) of browsing applications takes a back seat. The justification of this standpoint comes from the observations that firstly the user interface is primarily part of the design of the application and so opportunities for a generalised approach are limited. Secondly, that user interfaces, their toolkits and management systems have been much studied already.

The other standpoint is that of the user interface components that might support various forms of access. Here there is a concentration on the *way* in which the user interacts with the information and not on its structuring and access.

The work described in this thesis has developed from a computer science standpoint but, in view of the importance of the interface to browse/search applications, the needs of the user

interface have been considered and one way of linking the user interface to the browse/search system is described.

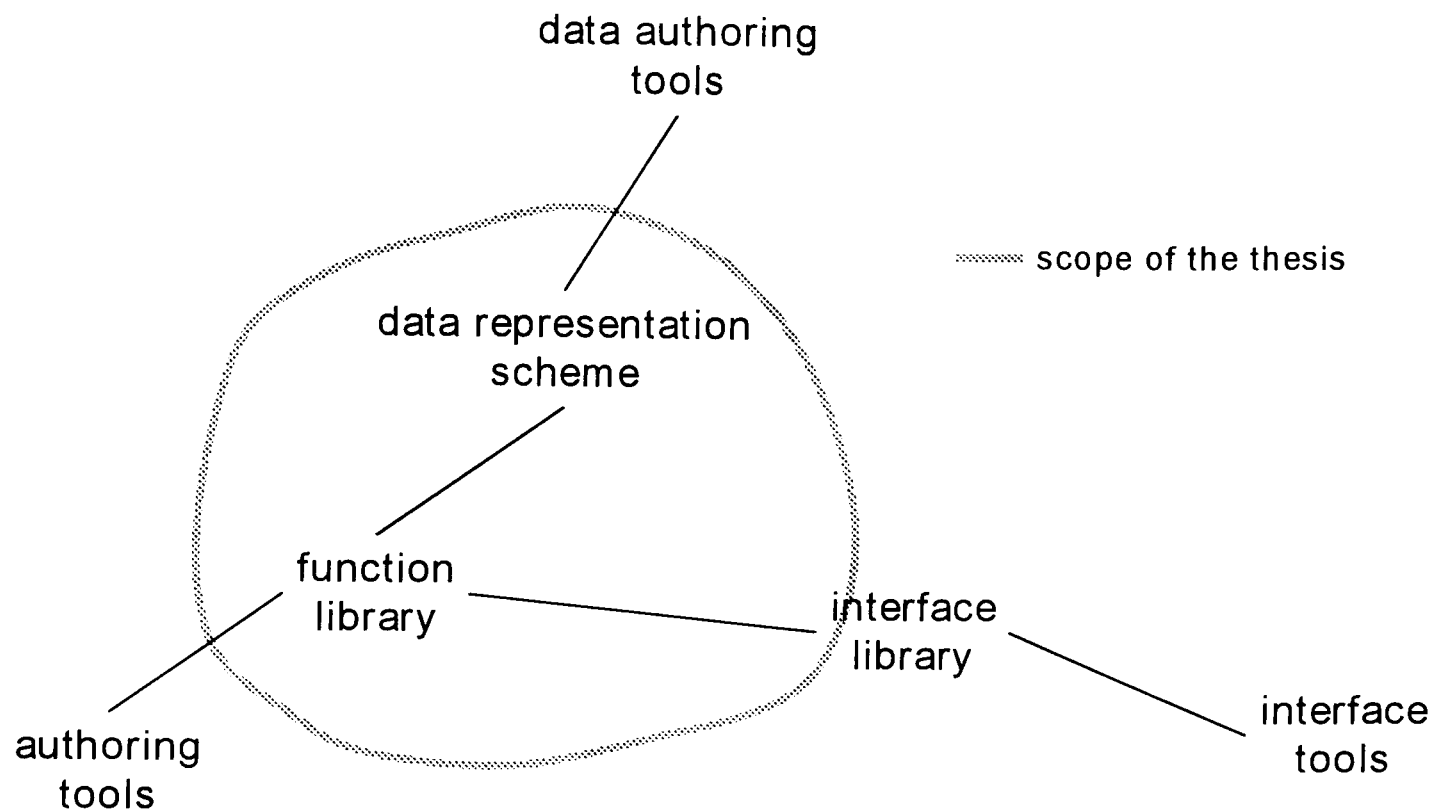


Figure 4. Scope of the thesis.

In order to design a comprehensive set of browse functions the process of browsing must be examined in some detail. Three themes are developed; the cognitive perspective, the task perspective and the functional perspective.

From the cognitive perspective it is not the aim of this thesis to devise and evaluate new cognitive models, rather the field is examined and relevant existing models considered to gain an understanding of the browsing process.

The task perspective considers types of browsing activity. This is addressed by observing browsing activities in a variety of settings.

The main thrust of this thesis is in the area of the functional design of browse/search applications and in particular what functions are required and what data model will provide the structuring necessary to support these functions. An important aspect to the approach taken in this work is that of retaining a link with the real world of implementations running on representative hardware.

The Generic Browser is at the pre-tools authoring level and offers the basic building blocks for the data representation, structuring and browse/search access to information. This thesis will describe the provision of a function library to accomplish this level of authoring. The library's advantages lie in its standardisation of browse/search access so that the author does not have to reinvent this functionality with each application he writes. As already mentioned the principal disadvantage of this approach is likely to be in terms of efficiency

either in code size or execution speed. Considerable benefit is likely to accrue if the Generic Browser facilities were to be added to existing multimedia tools but this lies outside the scope of this thesis.

3.3 Research Strategy

The Film Browser raised the preliminary problem of whether a generally applicable set of browsing primitives could be devised since this functionality seemed to be required by a number of applications. This problem space was then explored by the parallel processes of studies of real world browsing examples and the relevant literature for theoretical aspects together with prototype implementations and applications. The latter were used to both test ideas from the studies and feed back ideas themselves. This process led to the realisation of the importance of search and how its addition would increase the generality of the concept of the Generic Browser. The problem space was then enlarged to include the addition of search facilities and their integration with the browsing component already being investigated. The strategy used in both these phases has been one of a refinement process:

Firstly the technology space was defined.

The problem space was then explored to generate a set of requirements.

The solution phase consisted of forming a model and then assessing its feasibility and applicability.

The results of these assessments were then used to refine the model as shown in the diagram below.

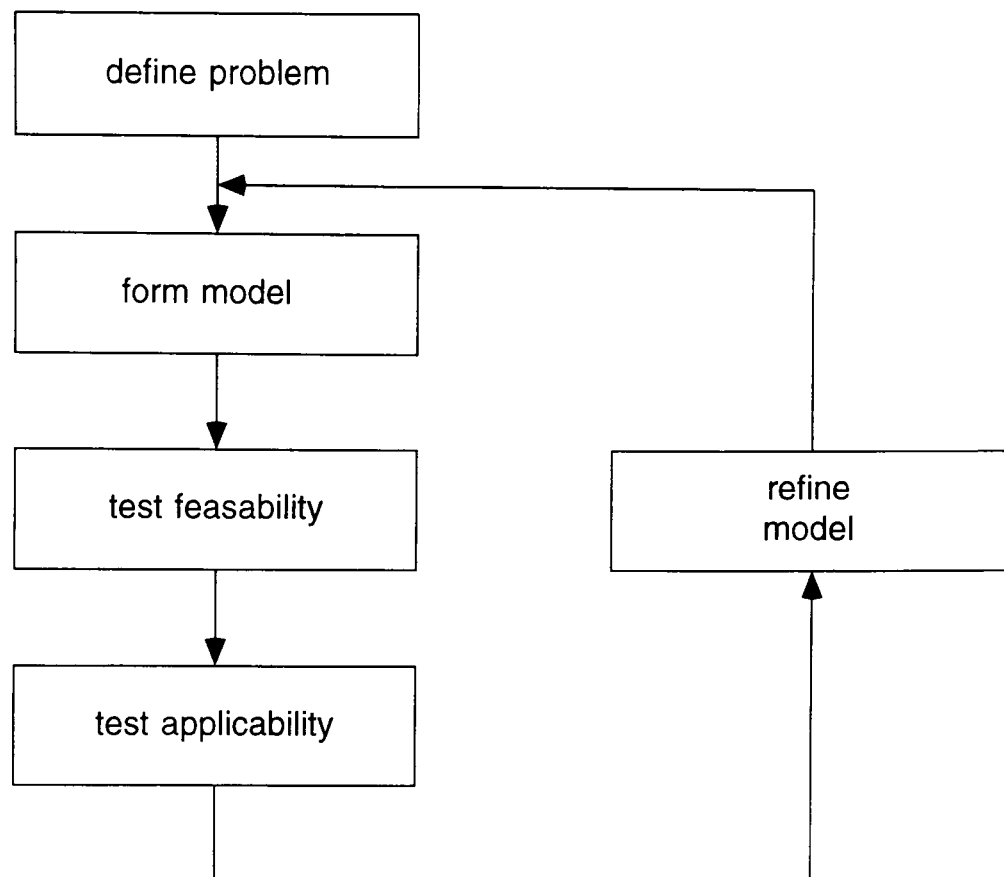


Figure 5. Refinement loop.

For clarity the study of browsers described in chapter 4 and the theoretical characterisation of browsing in chapter 5 together with the specification developed in chapter 6 refer to the final version of the Generic Browser in which search had been included. Similarly, apart from the descriptions of the individual prototypes, the evaluations described in chapter 9 use the final version of the Generic Browser.

In the next chapter the characterisation of the problem space is continued with the description of a study of real-world examples of browsing activity.

4 Study of Browsing

This chapter describes a conceptual analysis of browsing in different media by looking at real-world examples of this process. By doing so common attributes between differing instances of browsing behaviour can be extracted. These (and the differences) can then be used as a basis for a set of requirements for the functionality of the Generic Browser. This will provide essential real-world empirical data to back-up the theoretical model developed in the next chapters.

4.1 Examples Studied

This study opens with a look at the way in which people use printed material - the backbone of everyday information use. This is followed by consideration of computer-based information access and a look at existing CD-i applications to see how browsing and search are used in them.

4.1.1 Printed Material

Browsing printed material is the type of browsing with which people are most familiar. It is such an everyday activity that most do not even think about what they are doing. Yet it is this very familiarity which makes it so important to consider how such materials are used. Providing that functionality in the Generic Browser will minimise the need for learning and thus allow casual use and the sort of approachability that is a requirement in consumer multimedia.

In books and magazines browsing generally consists of flicking through the pages waiting for something to “catch one’s eye”. Their physical construction makes this easy. Additionally, most printed material has some form of internal content structure, for example the chapter structure in books, references, footnotes and so on. By virtue of their familiarity, which is based on the general conventions of the structure of such printed material, the user is able to make use of cues in the layout of the material to assist in navigation.

By their very nature, printed works also allow navigation and place marking by physical means - the thickness of the two sides of an open book is a rough guide to where one is in the book. Bookmarks, “post-it” notes, the folding over (or even removal) of the page corner allows the reader to return to pages of interest. The use of bookmarks of one form or another allows rapid navigation to known places within the book. It is possible to flip from one marked place to another and back so that information can be compared or cross referenced - this is like having both pages visible at the same time. Tables of contents act as a sort of map to the structure of the content whilst indexes provide an alternative route to

the information. Information is “chunked” into sections and chapters but it is also possible to view the pages as an approximate chunking of the information so that page turning becomes a kind of previous/next navigation from one chunk of information to another. It does not seem to matter that this is not an exact model, the whole process has an element of uncertainty built in which the human mind is quite comfortable with.

Searching in printed materials can take place using the ordering of the information in the work (for example an encyclopædia or dictionary), or via indexes or tables of contents. Here the table of contents acts a summary of the information, searching can take place at this level to find an appropriate area in the book for further searching or browsing. Searches may also rely on previous use of a book, for example using knowledge gained of the structure of the information “it’s in chapter 2” and physical cues “about 1cm from the front” (by thickness). Other visual cues are the layout of the material on the page, the presence of pictures and the use of colour. Search is frequently combined with browsing with the bias varying throughout the interaction.

Books and magazines provide a benchmark for information-rich multimedia. If computer-based information is to become as ubiquitous as the book it must take on board those aspects of books and other printed materials that make them so attractive to use, modifying them to suit the computer medium and augmenting them with facilities which are unique to computerised information access.

The browse/search process often extends outside the actual book or magazine. Looking for a book in a bookshop or library for example. Again there is structure to help; sectioning by topic, accession codes, subject and author indexes and so on.

The key feature of browsing printed materials are:

- familiarity of the form based on common features

- combination of browsing and search

- physical navigation cues

- bookmarks

- tables of contents and indexes

4.1.2 Window Shopping & Browsing Inside a Shop

Another form of browsing activity which most if not all people engage in is window shopping and browsing inside a shop. In the former the aim of the shop is to entice the browser inside so that windows are dressed to attract passers by. Viewed from an

information browsing perspective, the window contents are deliberately brief, eye catching, hinting of more detail within. Certain key aspects may be emphasised, the price, for example or the quality, whatever the main selling point of the items are. These are the search criteria that the shopper is using.

Once inside the shop, the customer may be more focused. He may still have not decided what to buy but is able to see a more complete range of the goods for sale. Assistance from the staff may provide useful information, as may the packaging. Stores are frequently laid out in a logical fashion so that the goods are grouped together by, say, function or type. This allows the search to be narrowed down to a subset of all the goods on sale. Related items often form part of displays, accessories adorn the mannequins in a clothing store. These, it is hoped, suggest that the customer should look at these items to complement his clothing purchases. This is an example of linking related items, another concept of importance to a general model of browse/search.

The key features of these browsing activities are:

- maps (e.g. store guide)

- summaries (e.g. the window displays)

- subsets

- linked items (such as accessories)

4.1.3 TV/radio Channel Hopping

This is an interesting browsing-type of activity in which the user “timeshares” his attention between a number of potential tv channels, either to get the gist of what is going on in a number of different programmes being shown at the same time or to monitor a selection of channels, waiting for something of interest to come on. This is an example of browsing in time, of time-varying information. Another example of browsing in time is the monitoring activity observed by [Ellis 1989] where users of scientific journals monitor them, skimming each issue when it is released, looking for items of interest. In both cases the user must wait for new/interesting information to arrive, no activity on his part will make it appear quicker.

4.1.4 Computer-based Information Systems

The use of computers to store and retrieve information is nearly as old as computers themselves. Specialised and highly efficient systems have been developed in the database field with the emphasis on professional use of information. The growth of the PC culture over the past ten years has seen the rise of database software for these machines, designed

so that non-database experts can construct and use database facilities. Systems such as dBase are widespread. They offer “traditional” query-based search on a record/field structure [Parsaye et al 1989]. They make extensive use of indexes¹⁵ to speed up the search but such indexes must be re-calculated when records are added or deleted. Such databases are therefore regarded as essentially static, with new information being added in batches and the indexes re-calculated. Browsing is restricted to previous/next operations on retrieved records (which form a subset of the database). Such subsets can be combined with the results of other searches.

Many information systems have some kind of query interface. Take, for example, the INSPEC systems which gives access to abstracts of scientific papers. Here the user must specify search terms (from a thesaurus of allowed ones) and “home in” on his area of interest by successive refinement of his query. There are several pitfalls in this style of interface. The user may have little knowledge of the structure or extent of the database. His first queries may well yield either far too many potential hits or too few. The use of Boolean connectives and special query languages mean that only those trained to use such systems can use them (or get the best out of them). However one advantage of the query-based approach is that the database can be narrowed down to a usable (or browsable) subset relatively easily. Another, perhaps more generally accessible example of such a system is the computerised catalogue to be found in most main public libraries.

The alternative to query-based systems is some form of hypertext or hypermedia. As has been noted earlier, hypertext has not taken off as fast as some expected though the “hot word” concept is regularly found as part of other systems such as on-line manuals and also encyclopædia CD-ROMs such as Microsoft’s Encarta or Compton’s Encyclopædia on CD-i. With hypertext systems the emphasis is on exploration rather than retrieval. A good example of hypertext in this rôle is World Wide Web. This is a distributed hypertext system which uses TCP/IP protocols to allow access to hyperdocuments on sites all over the world. The documents use a very simple mark-up language (HTML) which allows hot words to be linked to other documents or pictures anywhere on the web. Users of the system browse the networks of connected nodes in the same way that any hypertext is used (albeit rather slowly when the new destination node is held on a computer on another continent). The growing popularity of the World Wide Web suggests that this type of hypertext interface is particularly suited to this type of exploratory information access.

¹⁵In this context an index is a table or file containing one or more record keys and pointers to the corresponding record in the main database.

Key features of accessing computer-based information are:

- query interface
- subsets
- hypertext/hypermedia (associative links)

4.1.5 CD-i Applications

To demonstrate the state of the art in applications for consumer platforms this section surveys a number of CD-i applications. It looks in detail at the techniques employed by the authors to give the user access to the information on the disc. In a companion study [Lloyd 1993] looked at the whole CD-i application catalogue then available to see what proportion were based on information or used browsing as part of their interaction. He found that “at least 30% of the applications surveyed have an identifiable database requirement and nearly 40% make significant use of browse facilities”.

These applications highlight the superior still image quality of DYUV images when compared with, say, the 256 colour images of low-end personal computers, how the modest processor speed of CD-i is still able to deliver entertainment and information, the advantage of interleaved data masking the time taken to load data from the CD and a variety of interaction styles to what is basically just information whether it is sound, video, pictures or text. In each example the description of the application from the packaging is quoted to give an impression of the content.

Tell Me Why

“Tell Me Why answers over 175 questions that kids ask in five different subject areas: Our World, How Things Work, The Zoo, How Things Began, and The Human Body. The program features over three hours of fact-filled presentations, including interactive exhibits, an alphabetical index, a subject directory, guided tours and secret passageways”

This application is aimed at a younger audience. It is a disc version of the popular “Tell Me Why” books. These aim to provide informative answers to a wide variety of questions about topics such as how things work or the human body. Like the Smithsonian disc described shortly, Tell Me Why uses a pseudo 3D room metaphor with the user going through doorways to select avenues of exploration (an hierarchical contents list).

There are always only a few choices to be made at each level and although the textual names of the choices are only visible when the cursor is over the hotspot, the small number allows easy memory retention. The main hall (the main menu) allows selection from How Things Work, Our World, Zoo, How Things Began, Human Body.

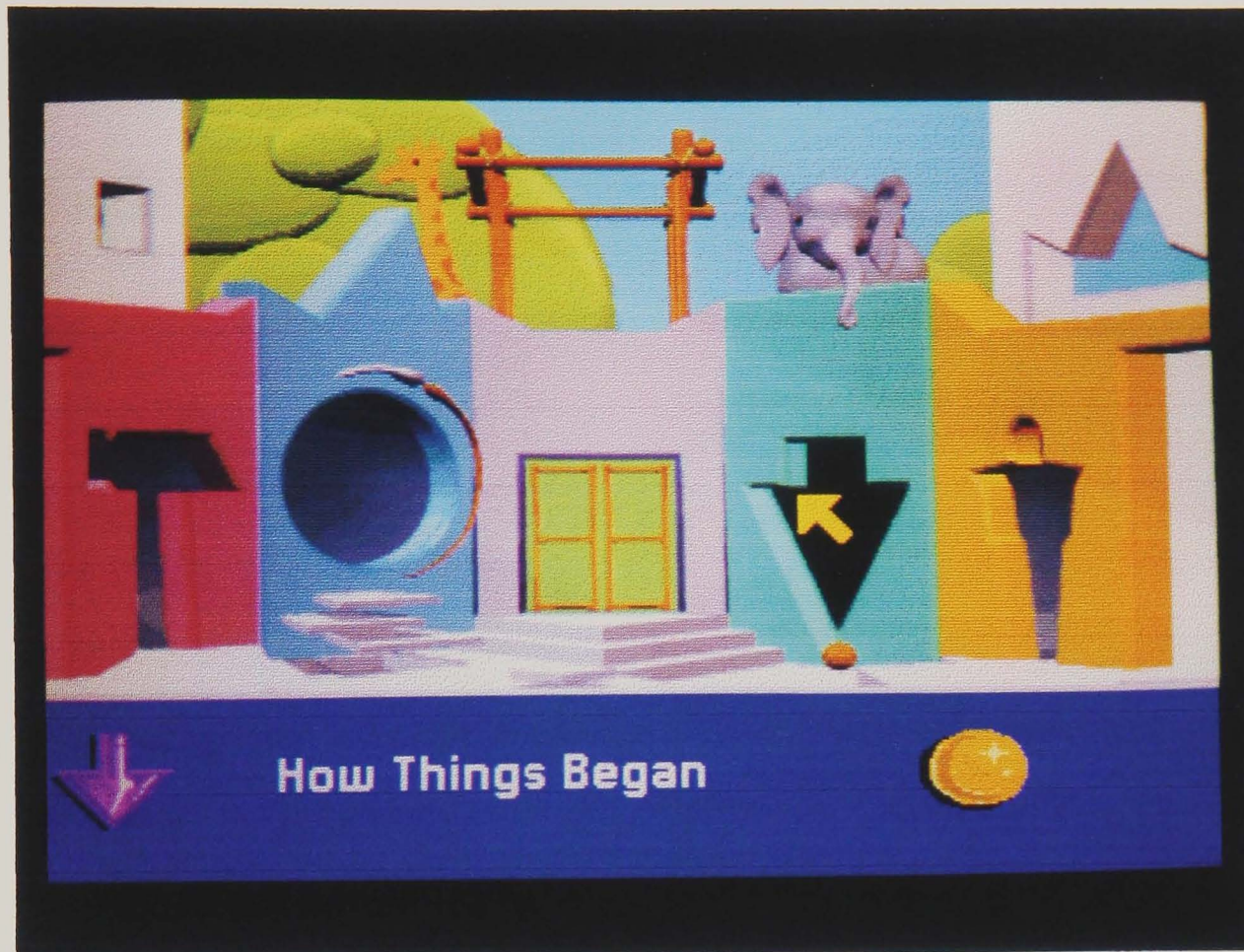


Figure 6. The entry screen for Tell Me Why - moving the cursor over the doorways displays the topic, How Things Began.

Moving through the doorway, past pictures of noteworthy people in the chosen topic area (which give a brief message about themselves if clicked upon) the user arrives at the next "menu".

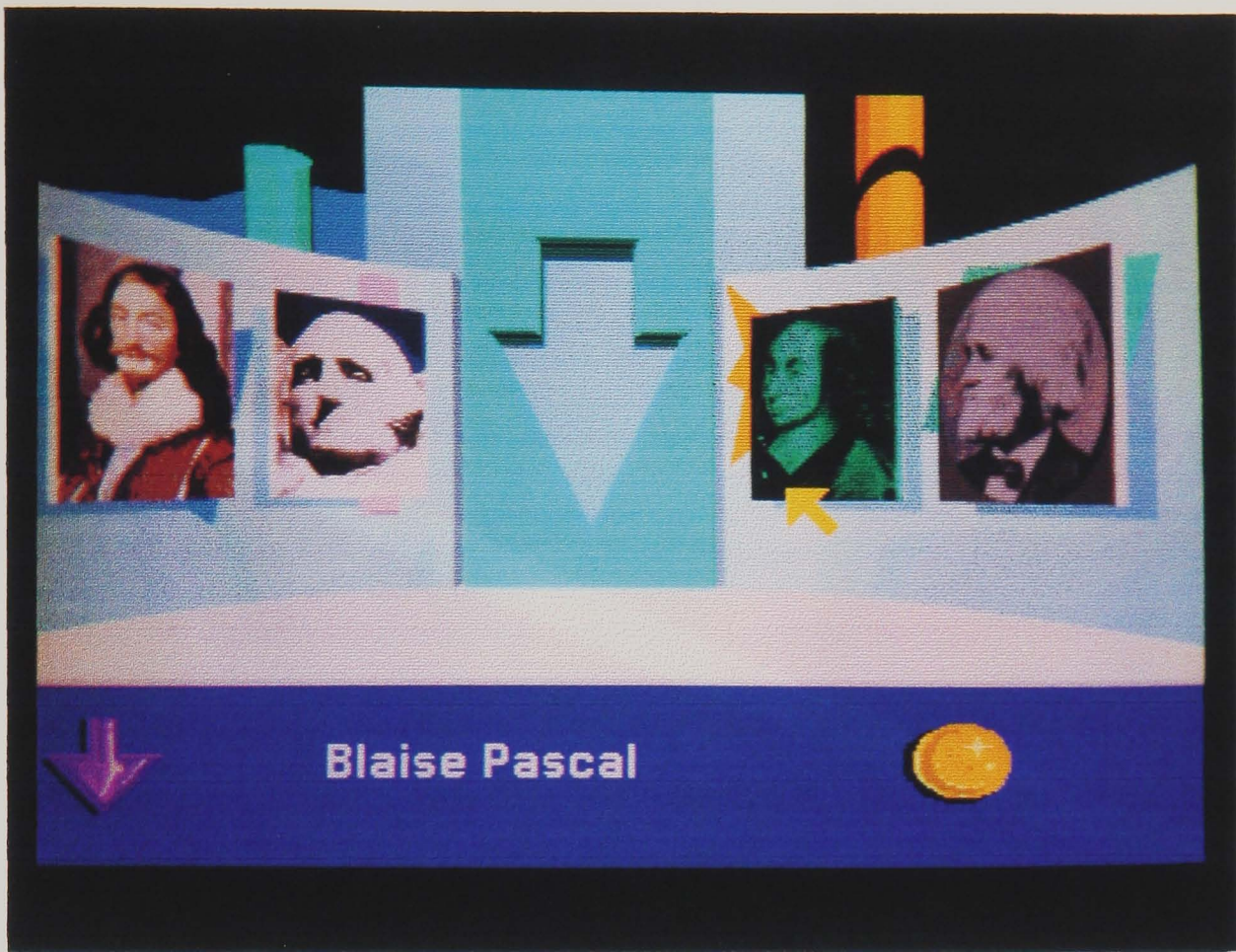


Figure 7. How Things Began - famous people gallery.

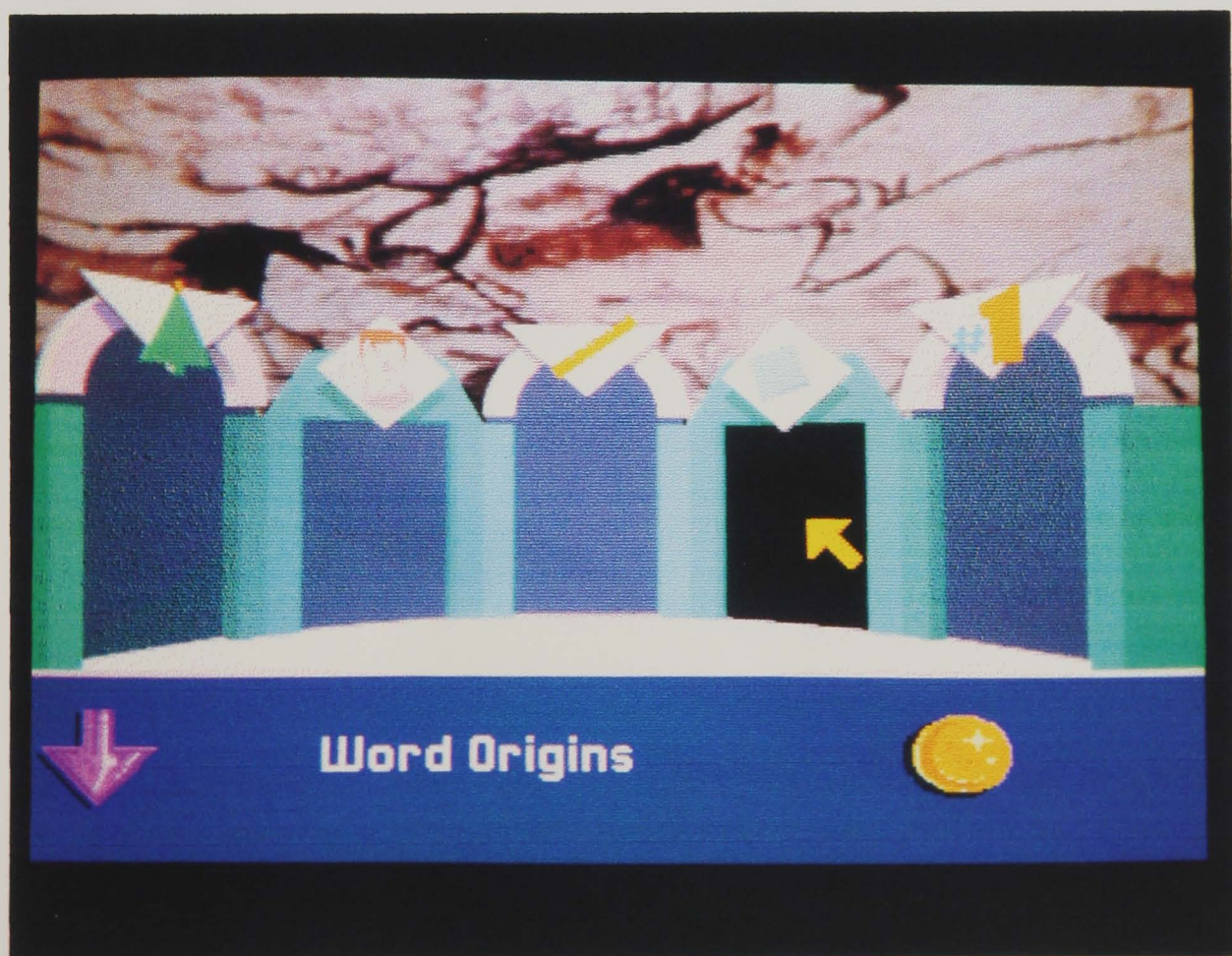


Figure 8. Selecting the Word Origins option from the How Things Began Topic.

From the Word Origins doorway the user is transported to the screen shown below. Clicking on Tom gives "How did animal get their names?" - a slideshow with a commentary about animal names followed by a menu of animals each of which gives a brief account of where its name originates from. Clicking on Woody gives "How did some trees get their names?" and on Goldy gives "How did some fish get their names?". Going through the right-hand doorway gives Famous Firsts and the left Special Exhibit.

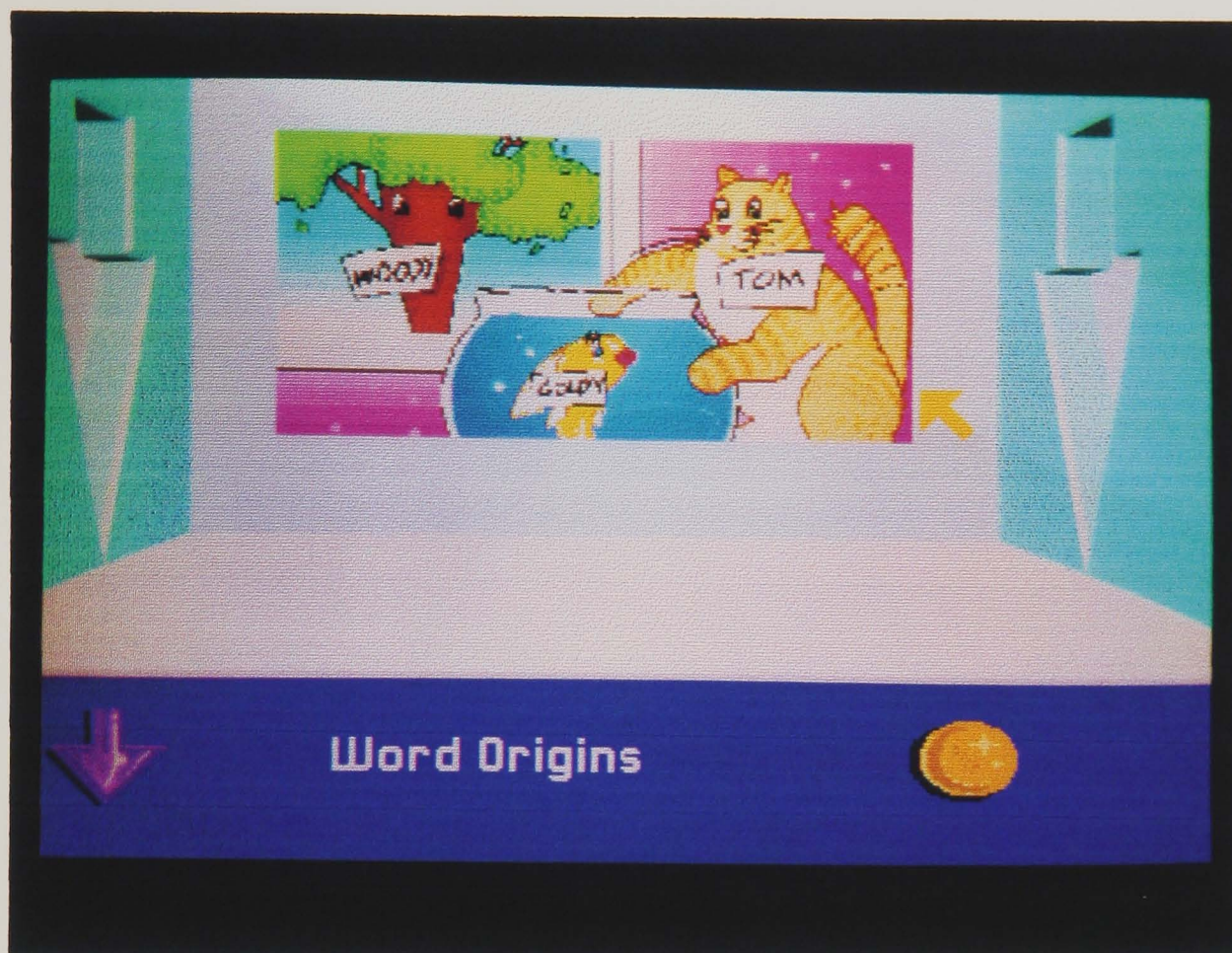


Figure 9. The Word Origins screen.

This simple hierarchy is only one way of accessing the information held on this disc. The Features screen offers the following alternatives (clockwise from the top left):

Credits and exit

The "doorways menus" already described

Guided Tours

An alphabetic index

A textual menu with the same options as the doorways

A random jump to an information screen

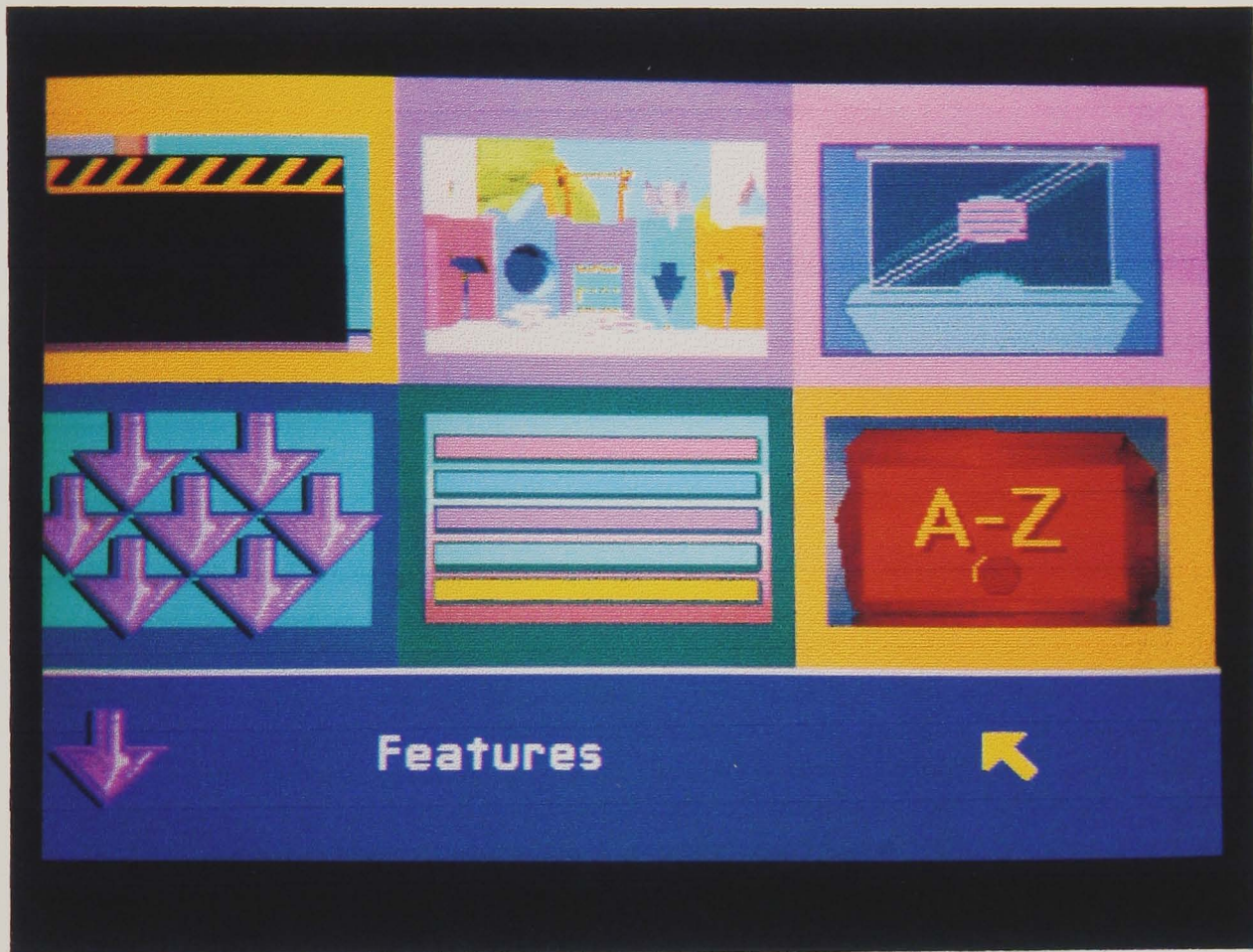


Figure 10. The Features screen.

At any time the down arrow allows the user to go back and the yellow button takes them to the features screen.

The key features of this disc are:

a variety of structures linking chunks of information such as the topic hierarchy and the index

simple point and click user interface

browsing and search within one application

Time-Life Photography

*“Your television screen becomes a simulated camera when you experience **TIME-LIFE**’s new interactive course in 35mm photography. In 25 step-by-step workshops, you’ll learn all the basic techniques of autofocus photography - from focus lock and shutter-aperture control, to depth of field. Then take practice pictures and see the results instantly on screen, without wasting film. See how to take better pictures in a variety of situations, from familiar shots of family and friends, to one-of-a-kind portraits by the greats. This program is based on **TIME-LIFE**’s classic Library of Photography and includes more than 1,000 great photos and two hours of narrated instruction.”*

Time Life Photography is an example of a tutorial disc. The content is arranged hierarchically with an hierarchical menu structure. This offers a quick way of browsing the content of the disc and navigating to items of interest in the same way that a table of contents can be used in a book. Each item, which takes the form of a commentary with illustrative pictures, can be stopped, repeated or skipped and there are diversions to related items after which the user can resume where he left off.

The key features of this disc are:

- hierarchical structure

- tutorial sequences

- links to related topics

Story Machine-Magic Tales

*“Everyone loves stories with fair maidens, handsome princes, fiendish villains and fantastic settings. Here’s a way to create stories of your very own, whether they’re fairy tales with happy endings or thrilling adventure stories. We’ve supplied all the tools you need to devise stories from scratch: dozens of scenes, hundreds of interesting characters, a complete alpha-numeric keyboard for narration and dialogue ... even musical accompaniment to suit every plot twist! This interactive experience also comes with three stories already on disc which you can embellish or just watch and enjoy. **Story Machine** will bring out the imaginative storytelling streak in everyone!”*

This disc uses a multitude of browsers to make the interface to story creation simple for young people. The basic idea is the creation of a story consisting of characters placed on backgrounds with music and sound effects and an optional text box associated with each screen.

For each of these components (excepting the text box) the user selects from browsers the item he wants. So, as an example, the user goes to the background browser to find a suitable backdrop for the next scene in his story. This browser presents 54 possible backgrounds three at a time as shown below.



Figure 11. The backdrop browser.

Once one is selected the user needs to populate the scene with characters. These are selected from a rather complex, linked set of browsers where a character group is selected from a set of eight, as an example perhaps the user selects the king.



Figure 12. The character browser.

The up/down buttons cycle through the nine varieties of king whose pose can be altered using the left/right buttons. The user can thus browse the alternatives before making his selection. Once selected the character is placed on the background and others added as required.

A “music browser” which is a 2D grid of icons each of which has a number of sound effect or music clips accessed by repeated button presses. This tool shows an example of how audio can be presented for browsing. Once selected the audio can be attached to the background to be played when the scene is displayed.



Figure 13. The music browser.

Once all the screens in the story have been assembled they can be played back in sequence.

The key feature of this disc is the simplicity of the underlying browsers with a fairly complex interface. In each case the browser is ultimately a linear sequence of objects (backdrops, actors and music).

Treasures of the Smithsonian

“Explore the highlights of a dozen museums in this unique interactive compact disc! From the AIR AND SPACE MUSEUM to the NATIONAL ZOO, treasures range from the World War I ALBATROSS FIGHTER PLANE to a ZANDE HARP from Africa. Along the way you’ll find everything from the HOPE DIAMOND to HOWDY DOODY. From LINCOLN’S LAST PHOTOGRAPH, to the famous PANDAS and artworks by PICASSO and POLLACK. You can browse through the treasures by museum, category, date, or theme. You can read detailed notes (who, what, where, when) and explore links between objects. Special features let you walk around an object, play its sounds, or zoom in on it.”

The interactive museum has always seemed a natural subject for multimedia applications and the Treasures of the Smithsonian was the first of this type of disc for CD-i. The opening screen gives the impression of actually walking into the museum. (See [Miller et al 1992] for a discussion of this type of pseudo 3D virtual walk type interface).

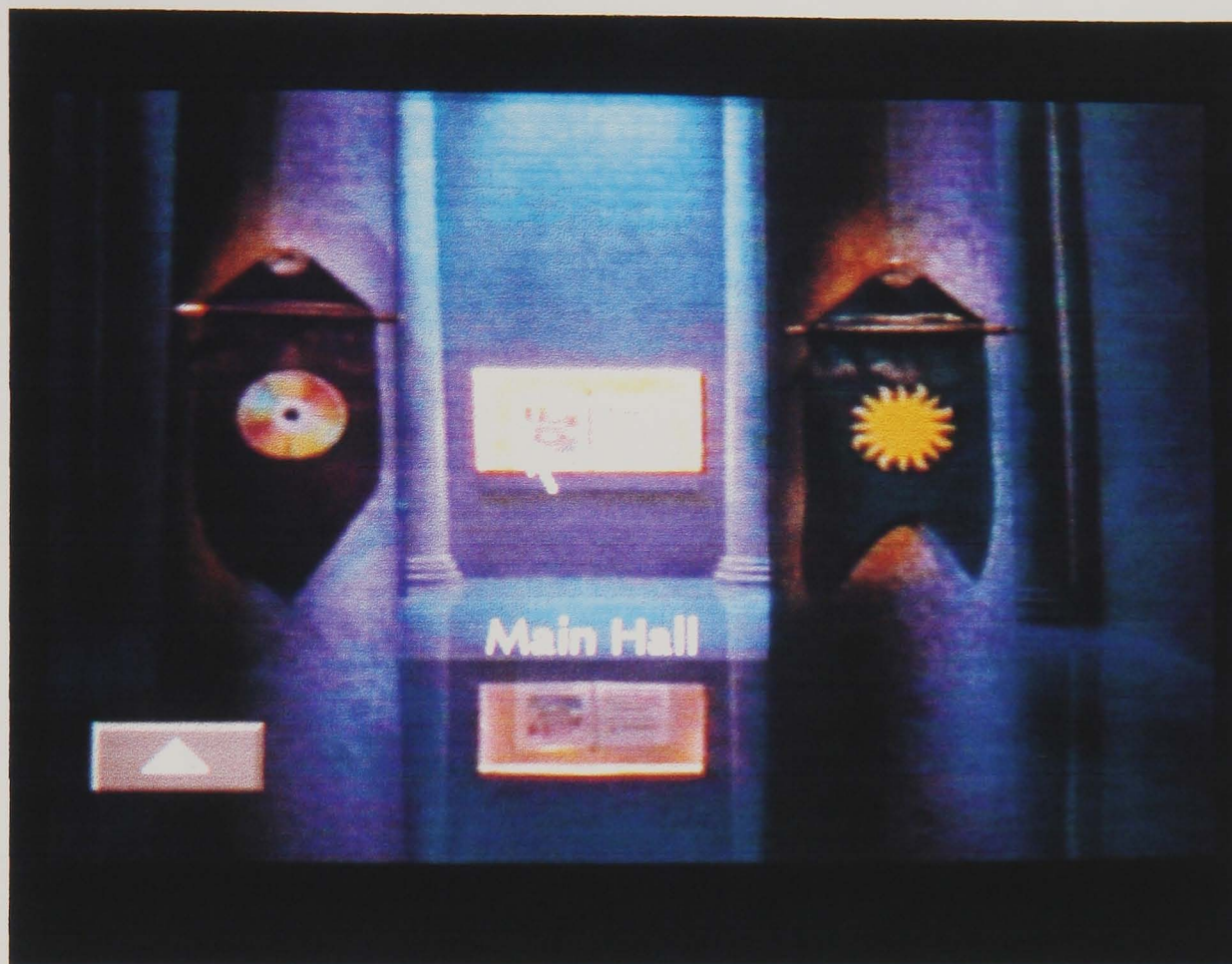


Figure 14. The Main Hall.

In the "main hall" the user selects from a number a different ways of accessing the information such as a time-line or alphabetic subject catalogue for example as shown below.

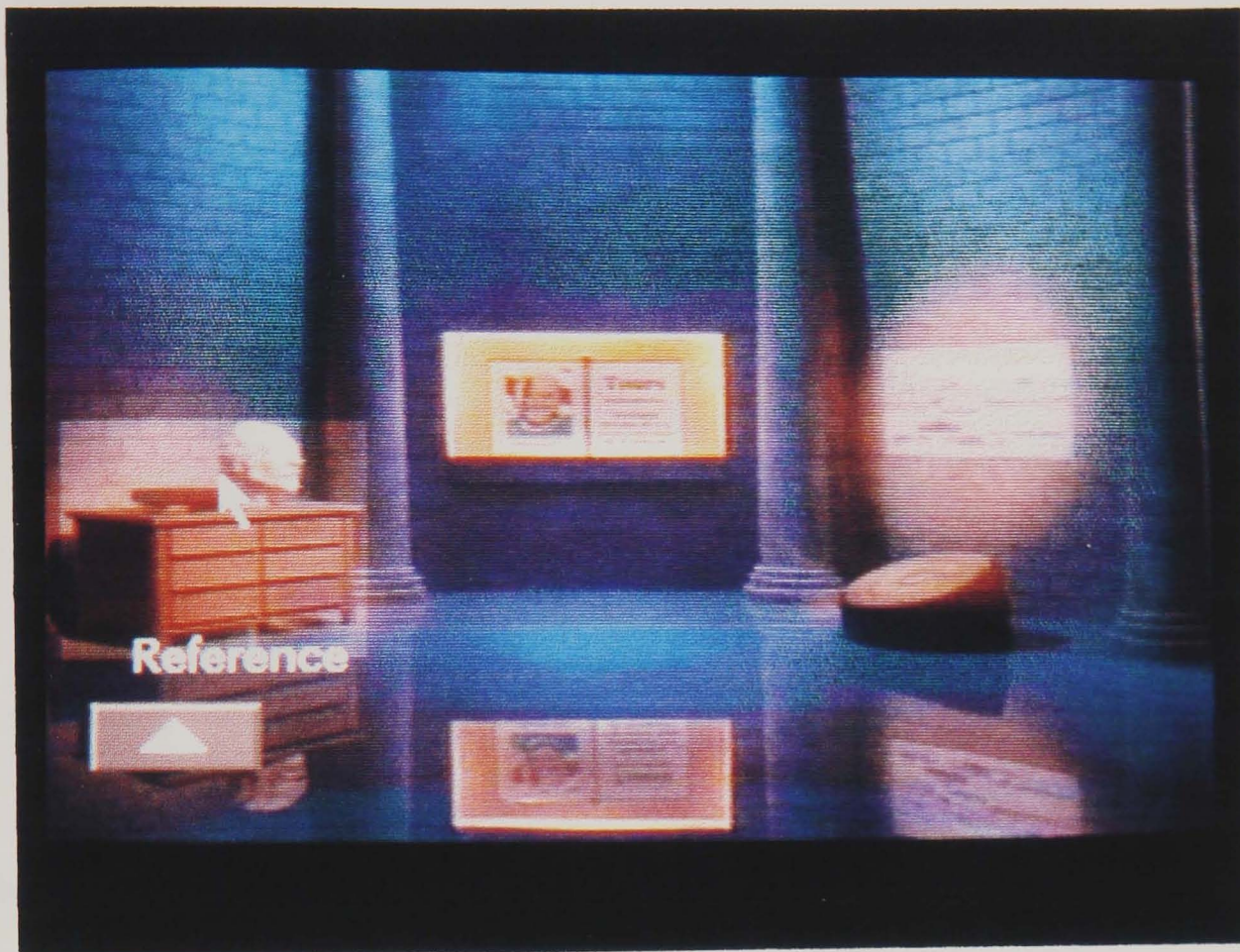


Figure 15. The Reference option.

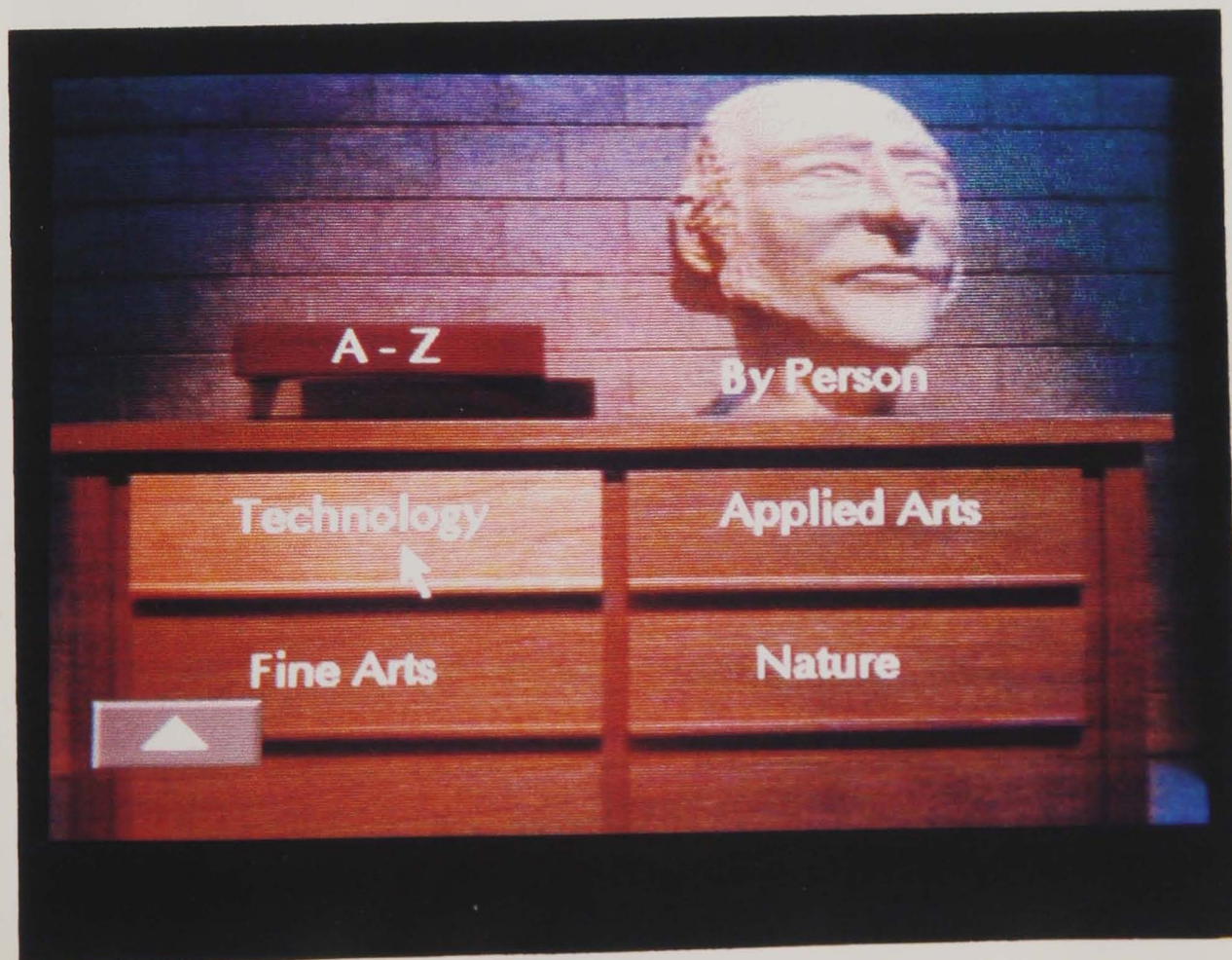


Figure 16. The technology "drawer".

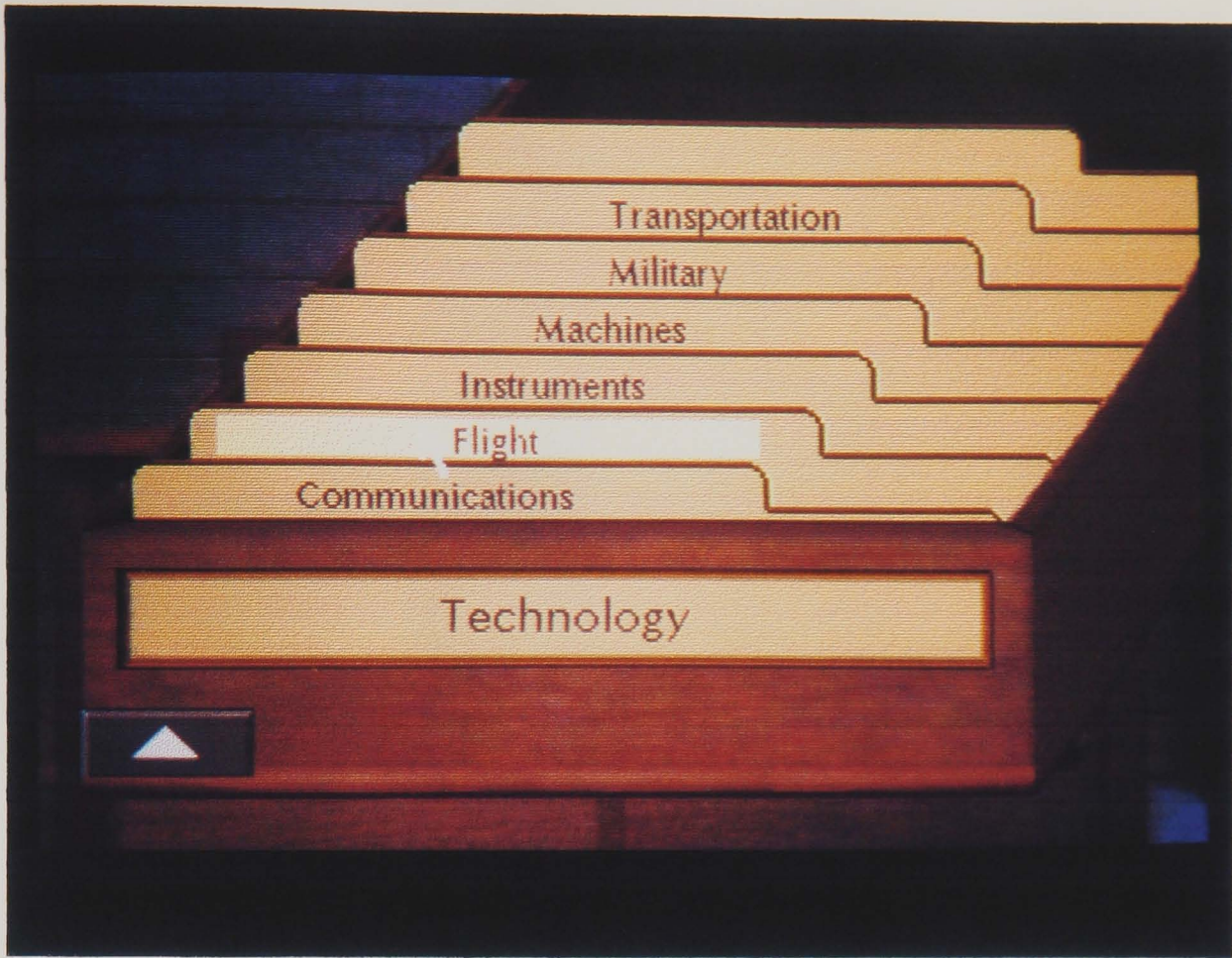


Figure 17. Topics.

Another option is to select from a range of guided tours each with its own theme.

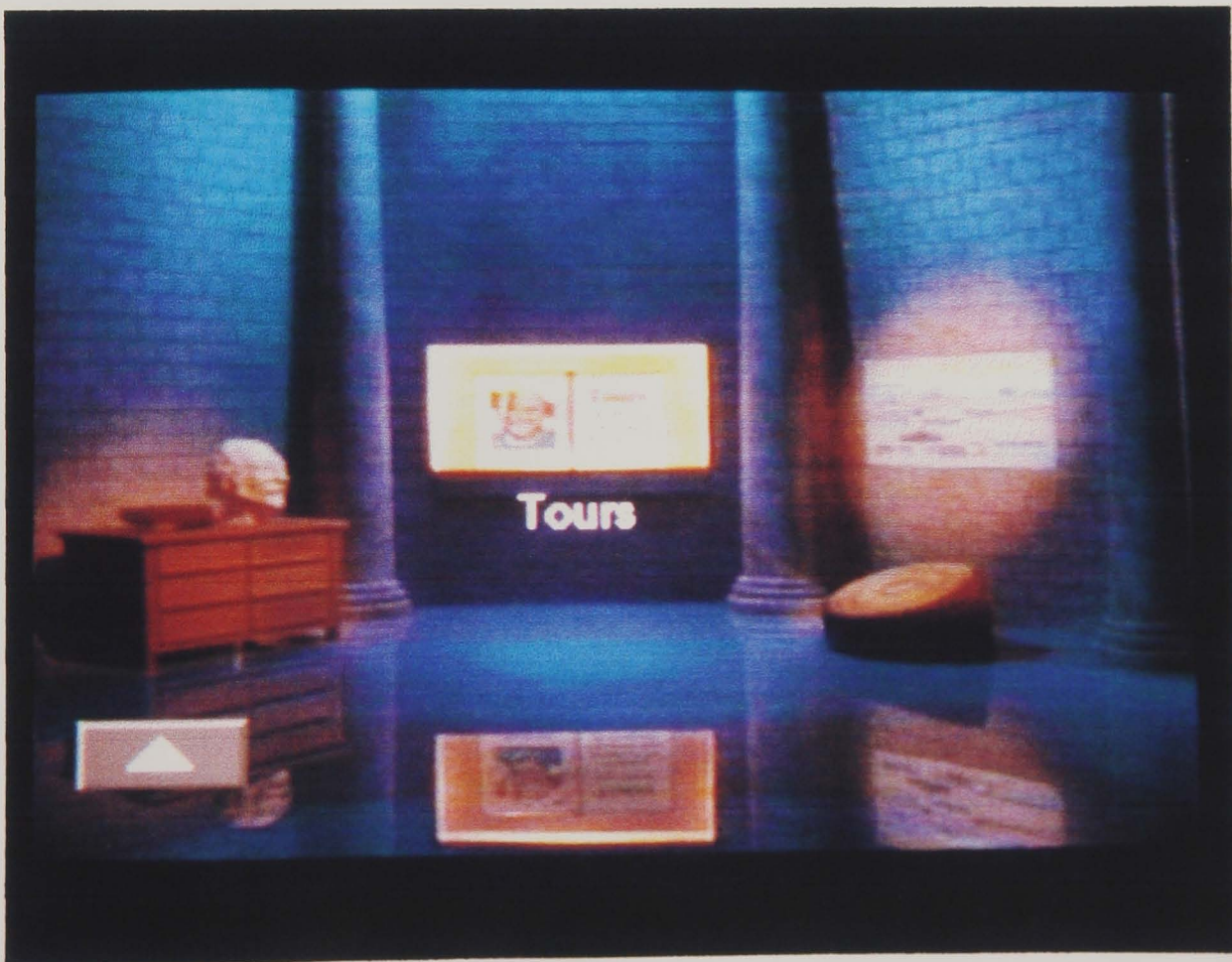


Figure 18. Guided tours.

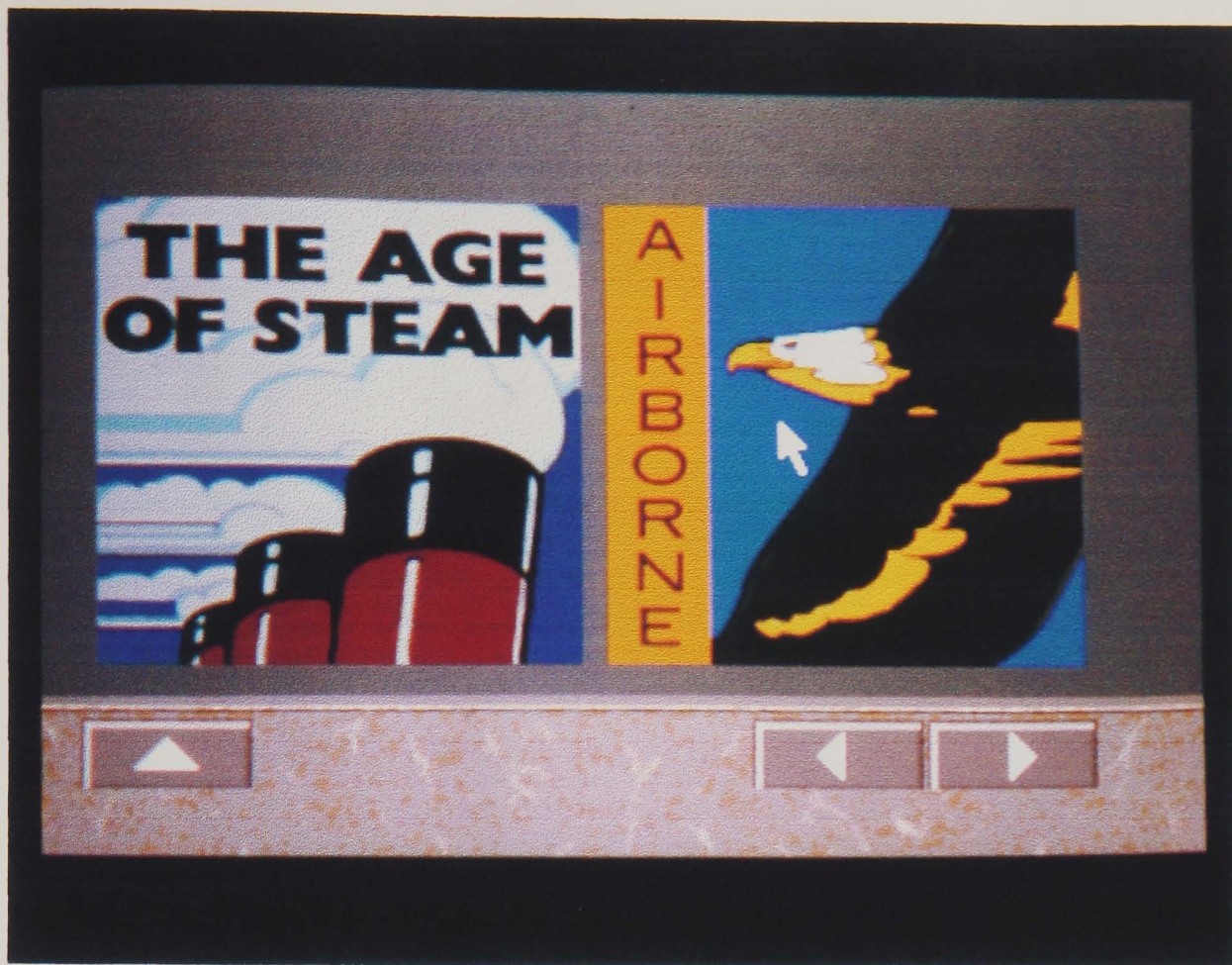


Figure 19. The "Airborne" tour.

Once "inside" a collection the user can browse other items via previous and next buttons as well as seeing more information (textual, audio or other pictures) about the item currently being viewed.



Figure 20. Information from the "Airborne" section of the museum.

Hypermedia links to related items are also available via a pop-up menu. Most items “explain themselves” through a short audio-video presentation.



Figure 21. Related Information - a stamp with an aeronautical subject.

The key feature of this disc are:

a multiplicity of ways of accessing the information. These are, in effect, ways of grouping (people or places for example) or indexing (timeline or subject index) the items in the database

a uniform browsing interface used for all items with access to previous and next items, further information and hyperlinks

guided tours with a theme connecting the individual items.

[Rankin 1991] suggests that an alternative to the database inspired informative CD-i applications such as “Treasures of the Smithsonian” would be to bias towards generating a rewarding experience as the user browses using a “trip to the shops” metaphor.

4.1.6 PhotoCD

PhotoCD (PCD) is a standard that defines the encoding method and disc structure for storing scanned images (specifically from 35mm photographs¹⁶) on CD.

Users of PhotoCD send their films off for processing and receive back a PCD disc (the disc holds ~100 images and can be updated so that the images do not have to be added “all in one go”) with his pictures on it. He can then show these pictures using a CD-i player connected to his tv. On each PCD disc is a CD-i application program that offers two picture browsers and a sequence building facility.



Figure 22. PhotoCD on CD-i.

In the first browser the viewer sees one picture at a time and navigates with previous and next buttons.

¹⁶For other formats Kodak have introduced PhotoCD Professional that can store images from larger format film.

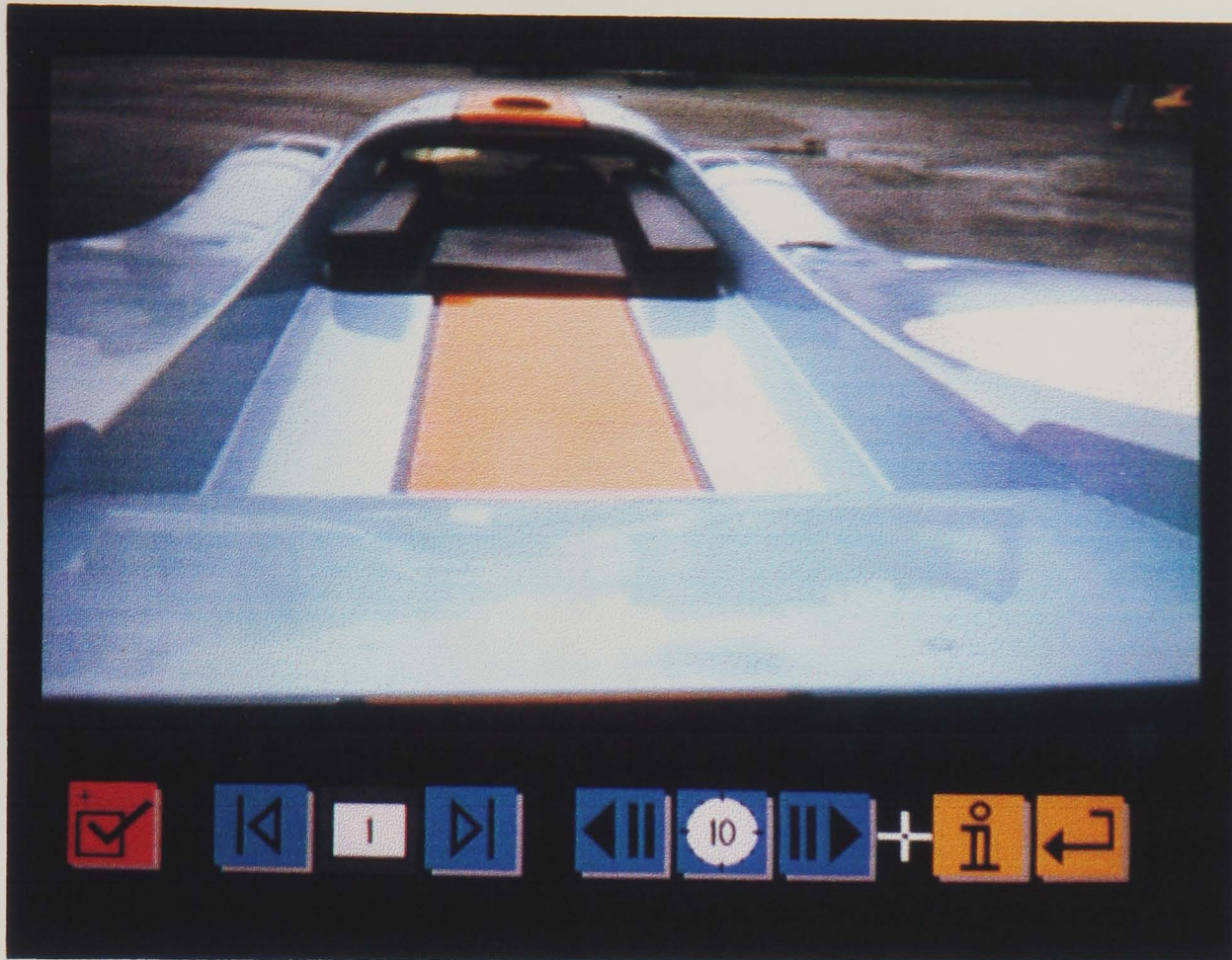


Figure 23. Picture-by-picture browser.

In the other there is a 2D picture browser with a 5x5 grid of mini-pics and next and previous page buttons.

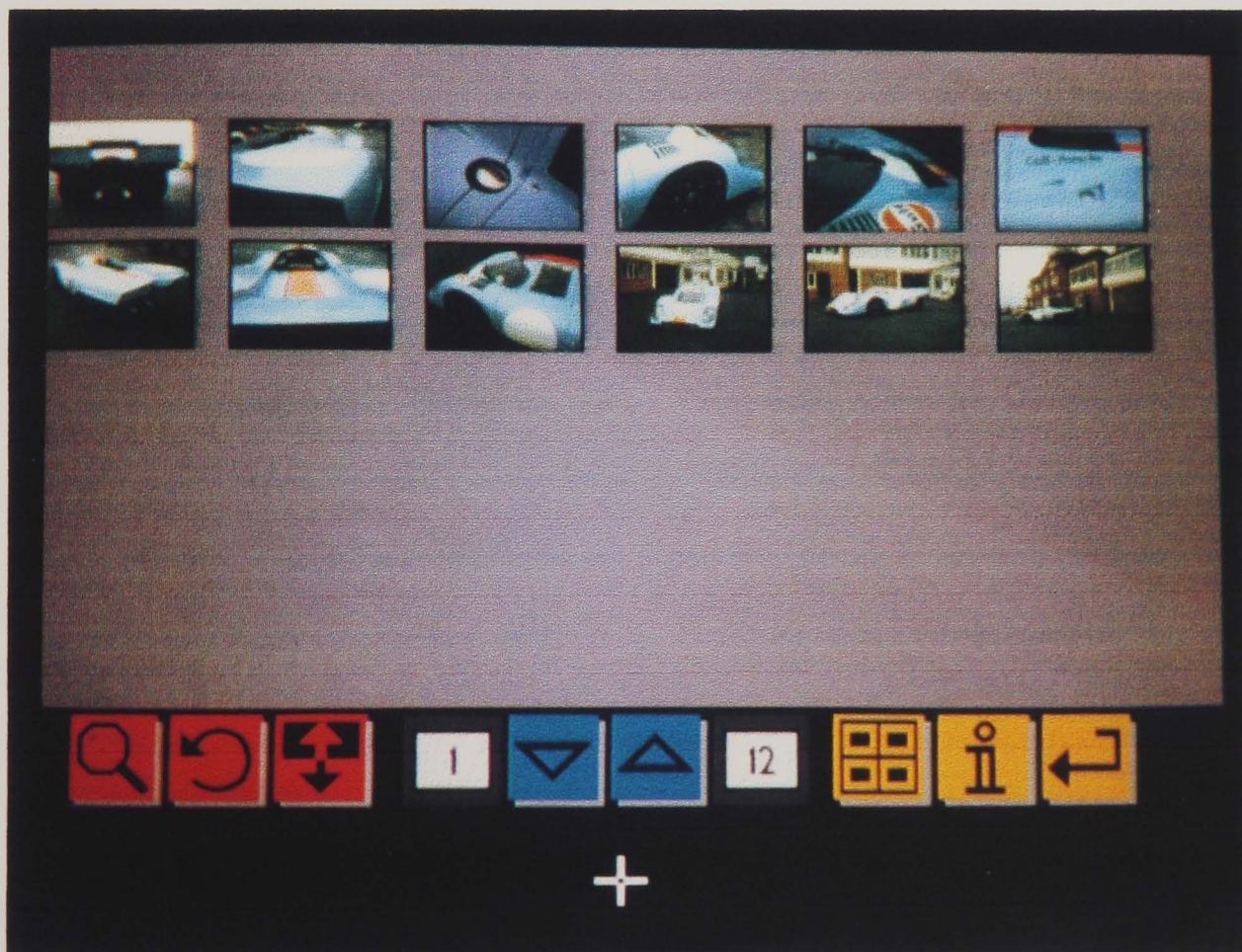


Figure 24. Minipic browser.

Using the sequence editing facility he can make slideshows which can either run automatically or be stepped through manually at the touch of a button.

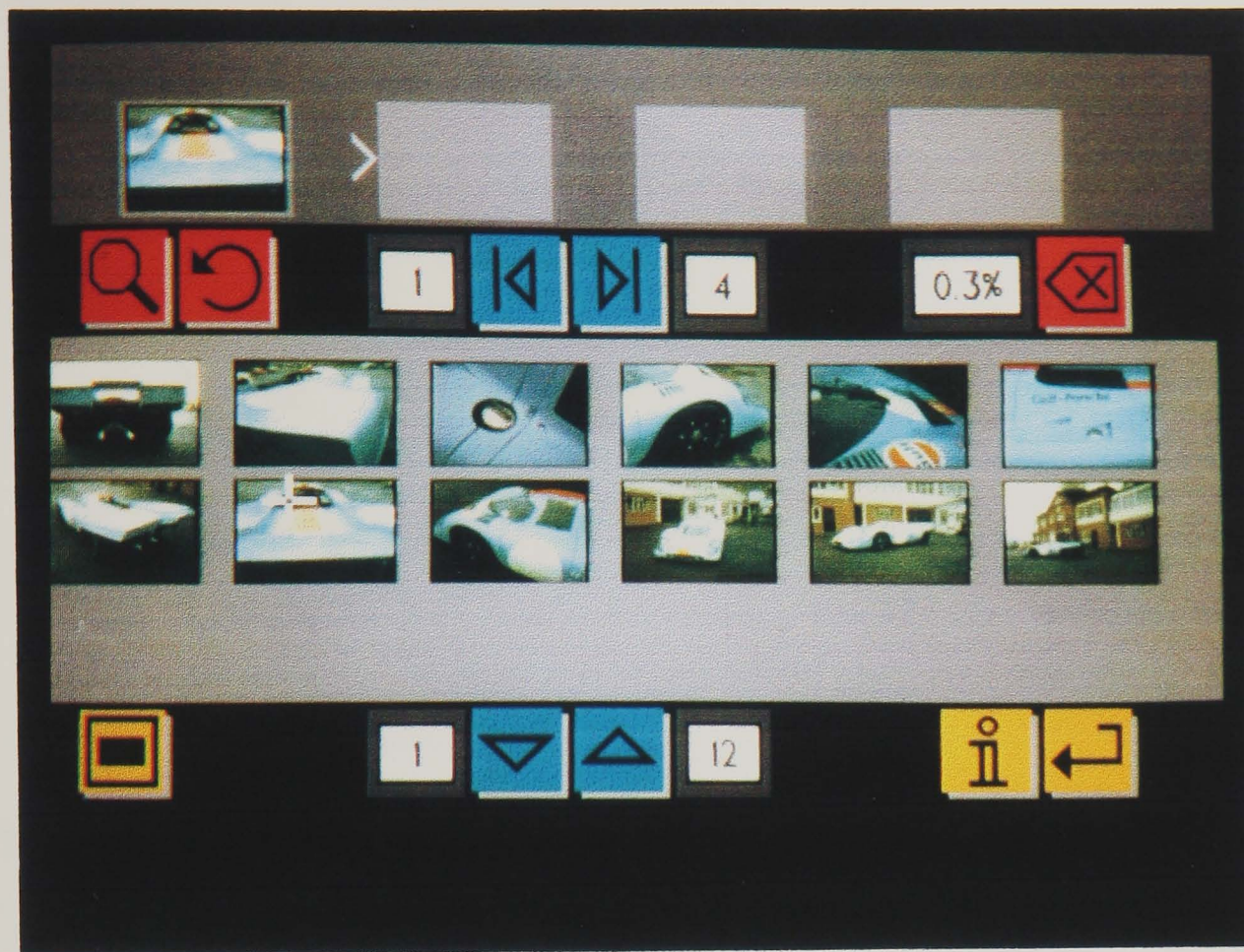


Figure 25. The sequence editor.

This disc shows simple, linear browsing of, in this case, picture objects. The sequence editor allows the user to create simple presentations - a sequence of pictures with variable time delays between them.

4.2 Observations

So what are the common features of the browsing/searching interactions arising from the conceptual analysis described above?

4.2.1 Printed Materials are Easy to Browse

The physical accessibility of books and magazines enables readers to get to the information in a variety of ways very easily. Stories can be read from beginning to end, magazines can be "dipped into", reference works can lead the reader from entry to entry hopping backwards and forwards within the book. Our familiarity with the conventions of these materials means that we already have basic mental models of the likely structure of a work even before we start to read it.

Perhaps our dependence on these conventions goes a some way towards explaining why, as [Conklin 1987] describes, people get lost so easily in hypertext systems which do not necessarily conform to book conventions.

4.2.2 Browsing Primitives

If these interactions are examined closely, stripping away the interface, a small number of browsing primitives can be found.

previous/next page/channel/item/screen - single choice (sequence/tour/narrative) or multiple choice

goto particular page/channel/item/screen

maps/indexes/contents lists

Browsing is essentially a “where next” sort of interaction. The interface presents the choices - often as simple as next/previous item.

4.2.3 Combined Browse and Search

Search behaviour is often part of the interaction, with browsing mixed with the proportions varying during the course of the interaction.

4.2.4 Browsing is Simple

It is not difficult to find examples of what appear, at first sight, to be complex browsers. In the CD-i world a good example is the Magic Tales disc described above. Close inspection of this browser reveals that the complexity is in the interface. Underlying this are a number of simple, linear browsers. The apparent complexity comes from the overlaying of several of these within one, complex, interaction device.

4.2.5 Subsets are Important

A good way to reduce the “agoraphobia” caused by the sheer volume of information is to provide a means of breaking that information down into manageable chunks or subsets. For example the chapter and section structure of printed works, guided tours on themes in the Smithsonian and the topic hierarchy in Tell Me Why (the deeper into the hierarchy one goes the smaller the subset).

4.2.6 Marking

The ability to indicate items or locations of interest for subsequent review or re-visiting is a common requirement in browsing situations for example bookmarks used in printed materials.

4.3 Summary

In chapter 2 four key aspects of the Film Browser were described namely:

- browsing (navigation)

- marking

- multimedia data

- subsets

It was postulated that these were more generally useful and this analysis has shown their use in a number of different existing browsing situations or applications. This suggests that these features form a useful starting point for the requirements of the Generic Browser developed in the next chapter in which browse and search will be considered from a more theoretical point of view.

The analysis detailed in this chapter combined with the theoretical examination in the next will be brought together in chapter six to form a model of browse and search suitable for implementation on consumer platforms.

5 Characteristics of Browse and Search

The previous chapter has looked at examples of browsing in the real world. To complement these this chapter examines research into the processes of browsing and search in the relevant literature. The aim here is to characterise these processes in sufficient detail that a model can be constructed to provide the basis of the function library: the tangible result of this thesis.

The previous chapter demonstrates browsing is a commonly used strategy for exploring information. This is, in a sense, its problem when one attempts to characterise it. [O'Connor 1985] observes that browsing "is still a term with a profusion of definition and consequent confusion". At first sight it is attractive to define browsing as the opposite of search as do [Liebscher & Marchionini 1988] with browsing being "an alternative to the complex Boolean search strategy" for "relatively simple and broad" search queries.

There is an overlap between these two activities; browsing can be used as a form of search (for something interesting) and that search can be used for exploration and discovery, traditionally the preserve of browsing. In between the extremes lie activities such as browsing used as a search mechanism for example [Palay & Fox 1981]. Here the user knows what he seeks but rather than specify it to the system as some form of search query he chooses to navigate round the information structure looking for it.

Rather than trying to provide a hard and fast separation between browsing and search this thesis proposes the alternative of viewing information access as a continuum with pure search at one end and pure browsing at the other.

Browse	Search
liesurely	focused
not focused	goal directed
exploratory	

Figure 26. Browse-search continuum.

The ends of the browse search continuum are "ideal" cases and are much less common than examples which contain combinations of both activities. It seems from personal observation that most information access/exploration activities contain both browsing and searching and that the proportion of each varies during the task.

For example, consider some activities possible with an Argos catalogue

find information on a particular Timex watch

- *pure search*

look at information on all Timex watches

- *search* for Timex watches then *browse*

look at ladies' accessories

- *search* for section then *browse* (many items)

look at anything (browsing for enjoyment or to fill time)

- *pure browsing*

Other examples are using an encyclopædia or the ACT College Search CD-i application which will be described in detail later.

5.1 The Task Perspective

Browsing's utility in information systems has blossomed with the advent of hypertext/hypermedia as an alternative to the traditional information retrieval (IR) mechanisms which largely rely on variations of and extensions to Boolean search. [Marchionini & Shneiderman 1988] observe that with such systems the searcher is usually a professional or relies on the expertise of one such as a librarian who is an expert in translating the users goal into the required combinations of search terms. They describe browsing as "an exploratory, information seeking strategy that depends on serendipity" which is "especially suitable for ill-defined problems and for exploring new task domains".

Some authors have avoided the problem altogether and included search within a model of browsing. For example [Salomon 1990] divides browsing into three types in his model. These are: goal directed search, goal directed browsing (where the user discovers a goal in the course of browsing) and casual browsing.

Browse		Search
casual browsing	goal directed browsing	goal directed search

Figure 27. Salomon's components of browsing.

In a similar vein [Cove & Walsh 1988] also have three components to their model: search browsing where the user has a specific goal; general purpose browsing where the goal is more general such as to find something interesting and serendipity browsing where the user

has no goal (other than the hope that something may turn up). These similarly fit with the idea of a continuum.

Browse		Search
serendipity	general	search
browsing	purpose	browsing
	browsing	

Figure 28. Cove & Walsh’s components of browsing.

From the user’s point of view the difference between browsing and search centres on the user’s goal. In the case of search the user wishes to find a particular piece of information. By contrast when browsing the user may have one of a variety of goals such as to pass the time, for the pleasure of the experience, in the hope of finding something interesting or to get an overview of the information space.

[Laurel et al 1990 p. 26] describe two types or “modes” of information access. These are *instrumental* and *experiential*. The instrumental mode is characterised by “a deliberate search for information to fill a particular need” and the experiential mode by “a more casual approach, often seen in activities such as browsing and exploratory learning”.

Browse	Search
experiential	instrumental

Figure 29. [Laurel et al]’s “modes”.

Because the goal of a search is well defined it is straightforward to see when it has been achieved - there is a tangible result. The search process has an element of closure associated with it [Ellis 1989]. In browse interactions there is less precision in the definition of the goal and so the measure of success is less tangible. Browsing is an open ended activity, often pursued at a leisurely pace. [Marchionini 1989] takes a similar view. He describes tasks having a specific objective, for example finding a fact such as “the first year speed skating was introduced into the Olympic games”, as a closed task which may best be found using search. Tasks better suited to browsing such as collecting information about a topic, for example “find information about women who have travelled in space”, are described as open tasks.

Browse	Search
open	closed
task	task

Figure 30. Marchionini’s components of browsing.

[Waterworth & Chignel 1991] attempt to make a distinction between responsibility (user or system) for selecting routes through an information space and the purpose of the exploratory behaviour. They define two dimensions as follows:

	<i>Targeted</i>	<i>Discovery</i>
User Handles Structure	NAVIGATIONAL QUERY	NAVIGATIONAL BROWSING
System Handles Structure	MEDIATED QUERY	MEDIATED BROWSING

Table 3. Dimensions of browsing [Waterworth & Chignel 1991].

Navigational browsing is typified by the link following of hypertext, the user only having a vague idea of what he is looking for. In navigational querying the user is still browsing but the topic is fairly precisely defined. In mediated browsing the user constructs queries such as “search Renaissance” to browse topic areas. Mediated search is typified by the Boolean search of many IR systems for example “search (Renaissance and Art) and (Religion or Christianity)”.

The authors do not provide evidence for these being two continua, rather they just provide examples of each of the four extreme cases. This suggests an alternative view - that they have identified four categories of browsing activity which lie on the browse-search continuum as shown below:

Browse	Search
Navigational browsing	Navigational querying
	Mediated browsing
	Mediated querying

Figure 31. Waterworth & Chignel’s browsing activities.

So from a task perspective the browse-search continuum spans a range of goal specificities. At the browse extremity there is only a vague goal - browsing to fill time for example. At the other extreme there are very well defined goals with the search for a specific piece of information. If the goal is simply to find something of interest then success or failure seems cut and dried, however such a goal is frequently tied up with gaining enjoyment in the actual process of exploring the information.

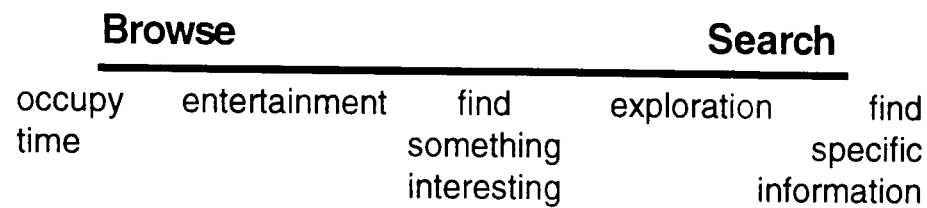


Figure 32. Task perspective on the browse-search continuum.

5.2 The Cognitive Perspective

The models described above attack the problem of characterising browse/search from a task perspective. It is also useful to consider what kind of behaviour is used to carry out browsing and search activities. Carmel et al choose to consider these cognitive aspects observing [Carmel et al 1992] that there has been “relatively little research on the basic human cognitive processes in browsing”. They develop a GOMS¹⁷ model of browsing which comprises three distinct browsing strategies: scan-browse, review-browse and search-oriented browse as shown in the diagram below.

¹⁷GOMS analysis [Card et al 1983] characterises users’ cognitive activities in terms of *Goals, Operators, Methods* which use the operators to satisfy the goals, and *Selection rules* for choosing between the methods.

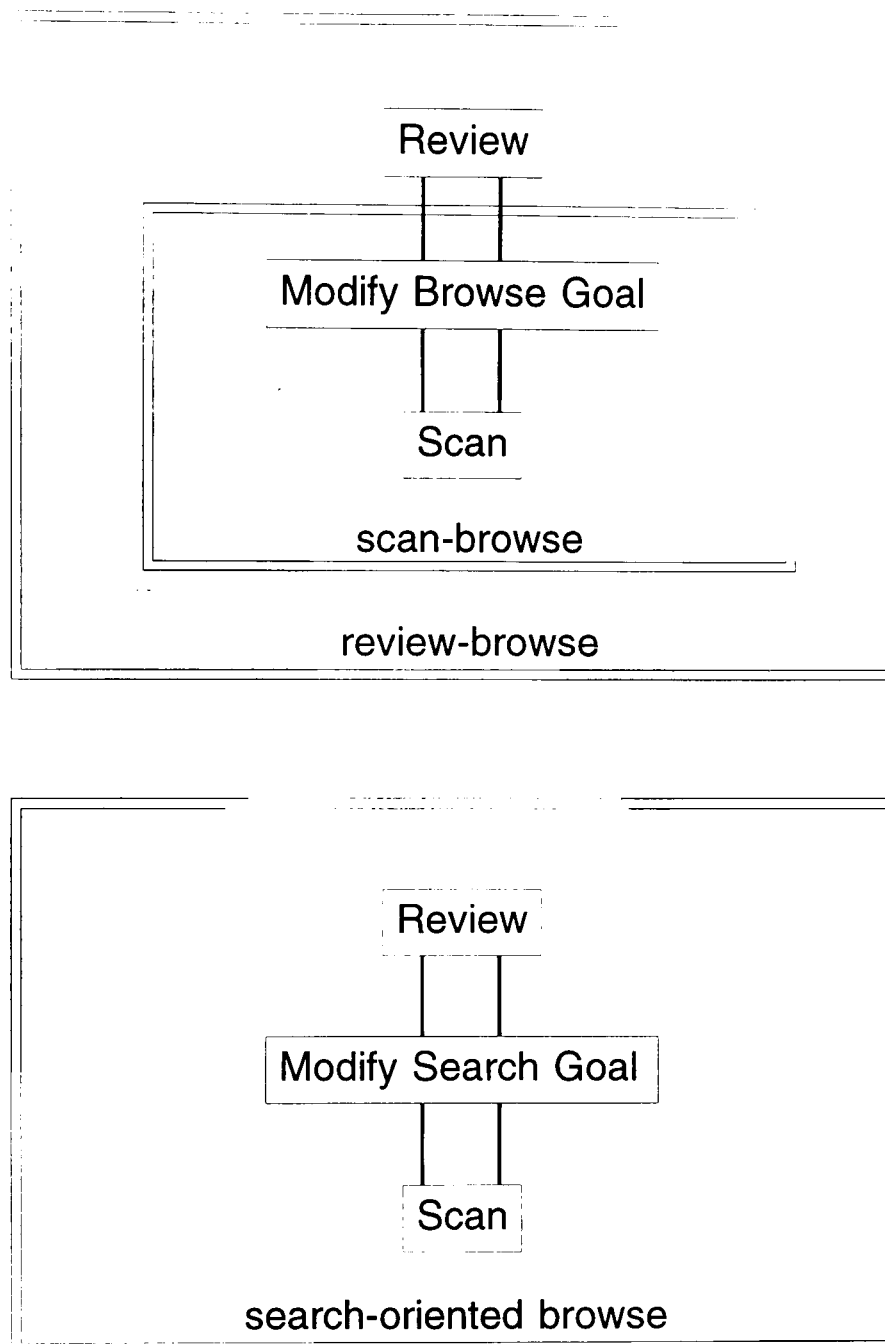


Figure 33. Three browsing strategies [Carmel et al 1992].

Search-oriented browse is the process of scanning to find and reviewing to integrate (update the user's mental model) information relevant to a fixed task. Review-browse does the same process for transient browse goals that are constrained by motivation (such as how interesting the information is). Scan-browsing is simply the process of scanning to find interesting information without any review.

[Garber & Grunes 1992] also describe a cognitive model, in their case for the task of selecting images from a photo library.

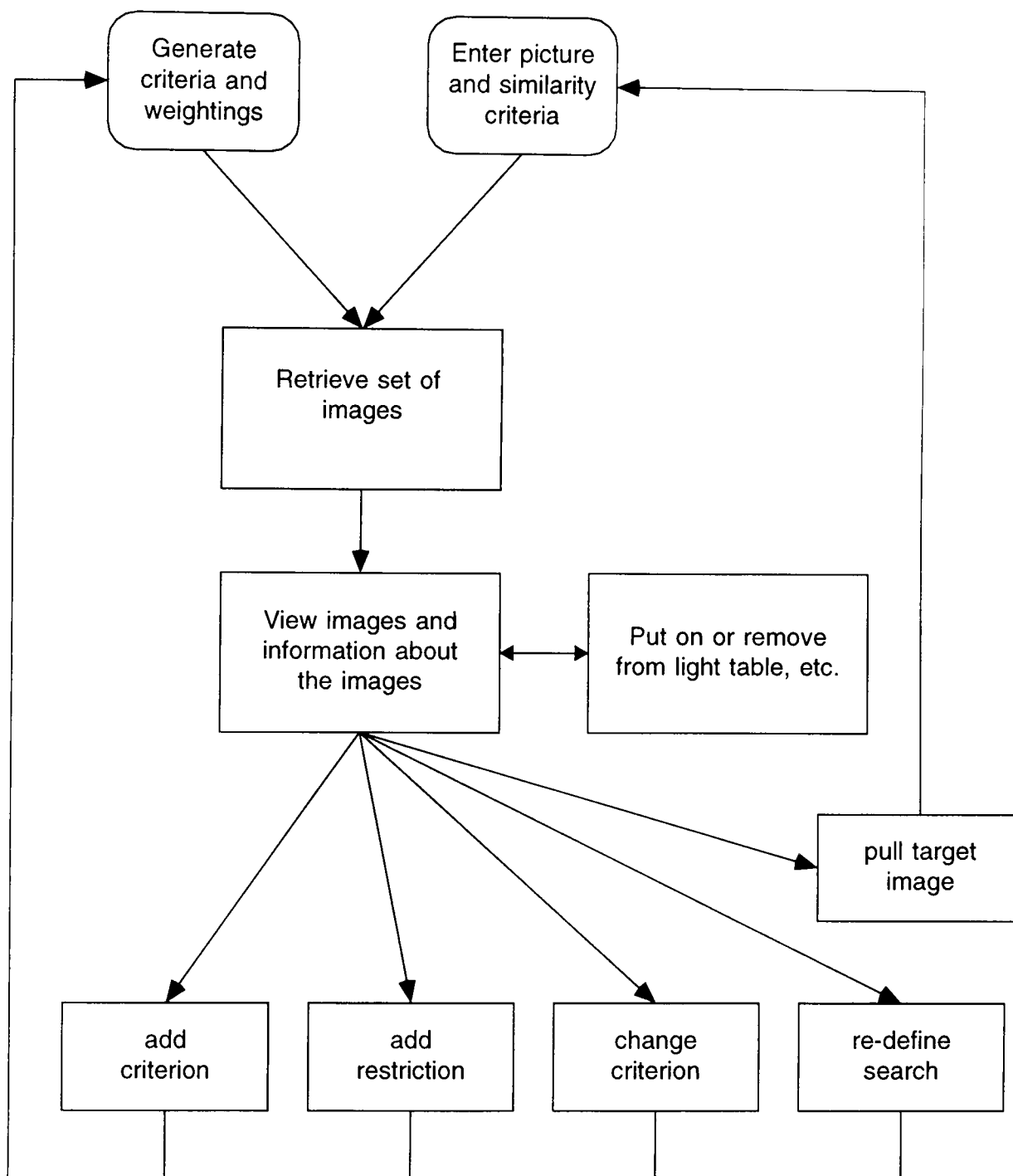


Figure 34. Search strategies [Garber & Grunes 1992].

Their model is basically the same as Carmel et al's search-browse. Again there is an iterative refinement process consisting of producing criteria for the search (in this case with a browser for the characteristics) followed by retrieval. The result is then viewed (browsing), the search parameters modified and the search repeated.

[Liebscher & Marchionini 1988] observe that for the Boolean, analytical search strategy "formulating search queries for answering complex questions requires a very high cognitive load". [Marchionini 1989] also notes the high cognitive load required to formulate queries and the low cognitive load of the recognition strategy used when browsing.

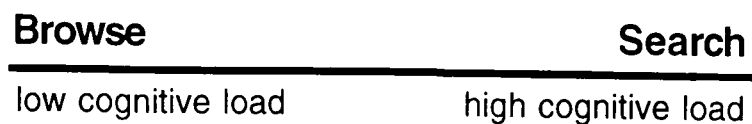


Figure 35. Variation of cognitive load.

In a paper on modelling a browsing interaction [Peck & John 1992] make a useful distinction between browsing and querying viz. “Browsers typically involve *recognition* of access paths, as opposed to *description* of access paths used in query languages, ...”¹⁸. Though the point they are making concerns the importance of perception in the browsing task, their distinction is generally applicable to the types of browse/search under discussion here.

At the search end of the continuum the user does not have to model the structure of the database, rather he must have a knowledge of the indexing methodology used. In browsing, information is obtained by the user interpreting and traversing the structure. The cognitive load is imposed by the navigation process and the requirement to model the structure of the information space. The use of placeholders, for example bookmarks, reduces the cognitive load of having to remember locations in the structure (this is the concept of marking developed later). In searching the user must model the search characteristics and also process the result to find the relevant items and modify the query to increase relevance

5.3 Integration of Browse and Search

[Liebscher & Marchionini 1988] observe that “an alternative to the complex Boolean search strategy is a simple browse strategy”. A relatively simple and broad search query is used to put the user “somewhere in the ball park” where scanning can locate the information sought. They note that library users prefer to browse through the library stacks rather than use the catalogue. [Jerke et al 1990] suggest combining the techniques of querying and navigating in a hypertext environment where “the user may specify his overall intentions by formal query which results in appropriate starting points for navigation through an information network according to the Hypertext paradigm.”

One aspect not emphasised in these models is the dynamic nature of many information access activities. Particularly outside the professional information retrieval field which is dominated by traditional databases, it is not possible to characterise an interaction as either browse, search or any point in between, rather the type of access changes with time. For

¹⁸my emphasis

example, a reader may open a magazine and start to browse, happening on an item of interest he reads it and then searches for related articles.

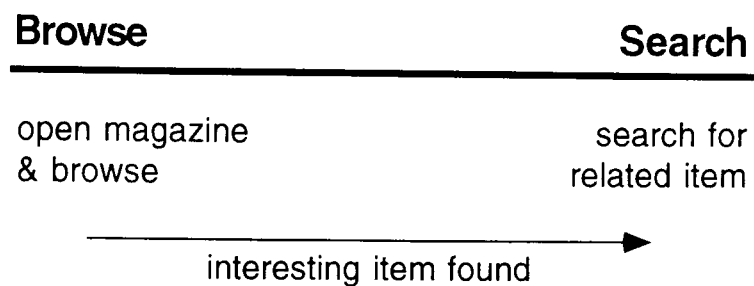


Figure 36. Shift of activity during interaction.

What started out as a typical browsing interaction, leisurely, unfocused has become goal directed and more search-like part way through.

This is an important aspect of information access which indicates the need to integrate browsing and searching interactions within the same application. As [Carmel et al 1992] put it “[...] in cognitive reality, any one of these browsing strategies might be interrupted or abandoned in favour of a different task, different strategy, or other activity. [...] The three browsing strategies identified are the components of complex browsing behaviour.” Similarly, [Duchastel 1990] notes that a “broad issue search will involve some constrained browsing, and conversely, browsing may well provoke the learner into investigation of particular issues that require search. In a sense, multimedia usage is a process that involves a constant interplay between browsing and searching.”

As another example suppose that a reader of an encyclopædia who starts by browsing in a leisurely way with no fixed search goal in mind, finds something interesting and follows references to find out more - a searching activity. The converse is also true when, whilst searching for an encyclopædia entry, the reader becomes side-tracked by an interesting item and starts browsing that item. [Laurel 1991 p. 25] describes this as a “mode swing” between browsing and focused searching “modes”. And in [Laurel et al 1990] “One mode may lead to another. An experiential activity such as browsing [...] may become a goal-directed instrumental [search] activity. Likewise the instrumental user may digress into experientially motivated behaviour.” Researchers at Apple recorded this behaviour in studies of the users of multimedia databases and in delegates’ use of the information kiosks which were set up by Apple at CHI ’89¹⁹ [Salomon 1990].

¹⁹CHI ’89 - The Computer-Human Interaction conference held in Austin Texas April 30th to May 4th 1989.

5.4 The Functional Perspective

So far browse and search have been considered from both task and cognitive perspectives. To enable a function library supporting such activities to be constructed a functional model must be extracted and to do this the processes must be viewed from the “machine perspective”. This perspective is necessarily more fine grained than the task or cognitive views which consider the user’s interaction with the system as a whole. That system must be opened up to allow the functional requirements to be exposed.

From the foregoing discussion in this chapter there is good evidence that a generally useful information access system needs both browse and search elements. [Halasz 1988] observes that

“[...] navigational access by itself is not sufficient. Effective access to information stored in a hypermedia network requires query-based access to complement navigation”.

from the experiences of users of the Notecards system (a general hypermedia environment).

Browsing is a process of finding information by navigating the information space whilst search involves giving the system some kind of retrieval specification. From a functional perspective the continuum becomes navigation to retrieval.

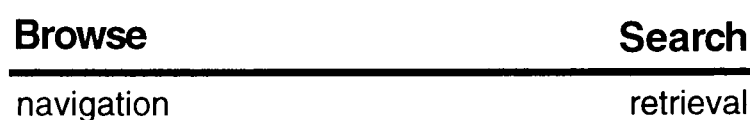


Figure 37. Browse-search continuum - functional perspective.

5.4.1 Navigation

Navigation is the process by which a user determines where to go next or how to get from place to place. There are three basic ways to specify the next node to go to, structurally, topically and presentationally.

structurally - These links form a structure where the links are just the means by which the system allows the user to get from node to node. A good example would be the “previous” and “next” buttons in a HyperCard stack. The user does not know what will be on the next card, the links just forming a browsable structure.

topically - These links, on the other hand, have meaning encapsulated in the link itself and relate the *content* of the two nodes at their ends. In this case the links form part of the information. [Carmel et al 1992] suggest that information should be provided as successively more detailed levels in separate nodes. An example might be a link labelled

“explanation” linking a node which describes an event with a node giving a more detailed account of why the event took place.

presentationally - Here the user specifies the next node by how the information is presented i.e. by which media type [O'Docherty & Daskalakis 1991]. For example the next node containing a picture. An example of this type of navigation occurs in the Apple Guides demonstrator featured in the CHI '91²⁰ video. Here the user can select from a menu where the media type of the information is indicated by an icon. [Andersson 1991] too has a media selection palette as one of his “minimal set of generic tools” for accessing hypermedia.

Navigation can be completely free as in a pure information system where there is a collection of information and a mechanism to get from item to item. In this case there is no “priority” given to any item, they are all treated the same so the user has complete freedom. When a structure is present in the information there are two possibilities: either the structure is just part of the information and freely traversable or the user is required (or just enticed) to follow a path through the information space defined by the author [e.g. Zellweger 1989, Marshall & Irish 1989]. Navigation is assisted by leaving place markers which form the user's own structure to the data space.

5.4.2 Retrieval

In information retrieval terms search is the process of matching a set of characteristics supplied by the user with the items in a database. In some systems a hit is only generated if all the characteristics match. Others have more flexibility allowing partial matches.

Weighting schemes allow quantitative distinctions between the degree of a characteristic present at different nodes. Weighting also makes it possible for characteristics in the query to be given different importance. For example, when choosing a skiing holiday resort closeness to the airport may be desirable but much less important than the provision of snow cannon. Giving a larger weight to the “snow cannon” attribute than to the “closeness to the airport” attribute would ensure that resorts with snow cannon would be higher up on the hit list than those close to the airport. Those with both facilities being at the top of the list.

The search process can be characterised by a user *specifying* what he wants (in some formal way), the system doing the search on his behalf and returning a set of matching items. This is an iterative refinement process the user revising the query in the light of the

²⁰CHI '91 - The Computer-Human Interaction conference held in New Orleans, Louisiana, April 27th to May 2nd 1991.

search results. The items in the database are classified by the assignment of a number of key characteristics of the information to each item. These are then available for searching on. In the IR field databases are usually interrogated using (often cryptic) query languages that do not encourage casual access to the information. An example of this is the SQL database query language [e.g. Cannan & Otten 1993]. For example:

```
Select Name
      from Cust-address
      where City = "BevHls" and
      amount > 5000
```

This would find all the customers living in Beverley Hills who have more than 5000 widgets.

The success of this type of search is defined in terms of two quantities, recall and precision. Recall is the ratio of the number of relevant hits to the number of relevant items in the database and precision is the ratio of the number of relevant hits to the total number of hits.

Such query-based search requires that the user can describe what he is looking for. It is often the case that a mismatch between how the user specifies the query and how the indexer marked-up the items in the database leads to poor precision. As [Laurel 1991 p. 24] puts it “people experience the requirement for precision as troublesome”. This problem is particularly acute when the user meets the database for the first time and/or uses it infrequently. This is exactly the most common situation in the consumer domain. Solving this problem motivates the quest for a suitably flexible search mechanism and the integration of search with the more familiar (for a non-technical audience) activity of browsing.

Although large bodies of information can be created for multimedia systems such as CD-i, these traditional access methods are unsuitable for the consumer market (and there is no a priori reason to think that a more comfortable access method would not be useful in the professional domain). A vital area of search problems exists where the user cannot (or does not want to) give an exact specification for the query and is quite happy with inexact matches, rather requiring the “best fit” to his specification [e.g. Gerber & Grunes 1992]. This may be used as a starting point for browsing [e.g. Liebscher & Marchionini 1988, Jerke et al 1990] as described above.

5.5 Generic Browser Facilities

Considering the results of the review in the previous chapter in conjunction with the discussion of browse and search above the basic facilities to be provided by the Generic Browser can be outlined. These will be fleshed-out in the next chapter.

This chapter has shown considerable evidence for viewing browsing and search as the extremes of a continuum of information access mechanisms. This in turn suggests that both browse and search facilities should be combined within one application framework. Within this overall framework a number of requirements can be seen.

In an information-rich environment the provision of hyperlinks is necessary for the leisurely browsing described as “casual browsing” by Salomon and “serendipity browsing” by Cove & Walsh. This type of “open task” as described by Marchionini can be seen as the page-flicking browsing of paper texts. This is the navigation extreme of the functional continuum.

At the retrieval extreme Marchionini places the “closed task” of search, Salomon “goal directed search” and Cove & Walsh “search browsing”. At this end of the browse-search continuum many IR systems use a version of the Boolean search mechanism characterised by a matching of characteristics specified in a query with items in the database. The problem of how the user is going to specify what he is searching for must be addressed. For example he might want to search by feature, by category, by media type or for a specific thing. It is desirable that the user should not be required to know how the system categorises or characterises its information in order to be able to specify his query.

The most obvious drawbacks inherent in the hyperlink approach are firstly that not all users would feel confident enough to explore without any guidance and secondly, as [Conklin 1987] describes, there is plenty of potential to get lost.

These problems are more important in the consumer domain. They make the middle ground of the browse/search continuum, where Cove and Walsh place “general purpose browsing” and Salomon places “goal directed browsing”, attractive. These avoid pure search with its problems of constructing the query and pure browse where the user may feel the agoraphobia of complete freedom to browse.

The dynamic nature of mixed browse and search shown by Laurel and Duchastel must be addressed in the integration. This is the type of browsing exhibited when using the CD-i application “Tell Me Why” for example. The user may also wish to get additional information on a particular item, compare one item with another and easily move to related information [Rudd & Cole 1991].

As well as the free browsing exemplified by hyperlinks, consumer applications require support for authors to choose the way in which the information is presented. The guided tours of “Tell Me Why” and “Treasures of the Smithsonian” demonstrate this requirement.

The review of browsing also showed that the ability to divide up the information space into subsets is a requirement and this is borne out by the “simple browse strategy” described by Liebscher and Marchionini where the information space is reduced by an initial search creating, in effect, a subset for browsing. This would also cater for the less adventurous user who would be unlikely to stray too far in an unconstrained browsing environment. This is particularly true if the user is able to refine the selection to home in on things that interest him as was demonstrated in the Apple GUIDES application demonstrated at CHI '91.

Marking as a tool for navigation is exemplified by the use of bookmarks in printed materials and its use for tagging items of interest for later review in the “ACT College Search” disc.

It may also be useful to be able to retrace a path (rather like following a piece of string unrolled as one traverses a maze). Examples of this are HyperCard’s “recent” command and the Trailmaker Tool [Hooper Woolsey 1991].

5.6 Summary

In the previous chapter the four key elements were proposed as a basis of a generic browsing application framework. These were

- browsing (navigation)
- marking
- multimedia data
- subsets

This chapter has shown how the addition of search allows a more general solution and addresses a wider range of potential applications. To summarise these requirements; for a range of applications that use browse and search, such systems should allow the user to:

- browse and search multimedia data
- browse freely in the information space
- look at related items
- have information presented as an overview or in detail
- mark items, possibly with a value

- collect sets of items for further browsing
- leave navigational placeholders to allow an item to be revisited
- retrace steps i.e. to backtrack
- follow guided tours
- search for items by a variety of query mechanisms
- freely move between browsing and searching access

Building on this understanding of the processes of browsing and search and how they might be used the next chapter takes these requirements and develops them to build an architecture for the Generic Browser.

6 An Architecture for Multimedia Browsing

This chapter discusses the design of the Generic Browser. It addresses the requirements laid down in the previous chapter with an architecture for browse/search systems. This is described below together with a set of functional requirements for its implementation. Its foundation is a data model which is amenable to the representation of information for both browsing and search. The facilities for browsing and search are then addressed together with their integration. The architecture thus constructed forms the theoretical basis for the implementation work described in the following chapter. It should be noted that the architecture described here is that of the final version of the Generic Browser incorporating the feedback from the prototype applications. Where such feedback has significantly altered the architecture, this has been indicated below.

6.1 Data Model

Most hypermedia systems use a data model that originated with hypertext systems [Conklin 1987]. This node-link model stores information in objects (nodes). These objects are joined in a browsable structure by links (arcs) forming a directed graph. Information retrieval systems also store information in objects (often the actual documents in the database). The similarity of the data models for these two domains (which have browsing and search as their respective primary information access methods) makes this model appropriate for the Generic Browser where browsing and search are to be integrated.

The dynamic nature of browse/search interactions suggests that using a single data representation for both browsing and search is necessary. If separate representations are used information such as marks would have to be maintained in both representations (so that searches and browsing of marked items was possible for example).

Any data model constructed for browse/search applications must be rich enough to represent a number of key aspects of the data. [Parsaye et al 1989 p. 249] suggest the following from a hypermedia perspective: representation of text & graphics; representing concepts; representing organisational structures; representing the relations between concepts. These are equally applicable to consumer multimedia, however here text and graphics covers everything from computer graphics through photographic images to video and audio.

The Generic Browser model has the following components:

- information nodes - multimedia data elements
- links - the basic structuring elements, multiple types
- paths - to represent sequences
- subsets - collections of nodes

The data model is a dynamic model. All of the above can be created, deleted and modified during application execution. One consequence of this is that additional information pertinent to the browse/search functions can be stored in the nodes. This preserves the object oriented nature of the browser with node-specific browser information being held within the node to which it applies.

The approach detailed here maintains the distinction between data and its presentation. This has been advocated by a number of authors [e.g. Campbell & Goodman 1988, Andersson 1991] and is part of the MHEG²¹ model [e.g. Price 1993, Kretz & Colaïtis 1992] which makes a clear separation between “content” and “projector” objects.

In the Generic Browser model a conscious decision was made to represent multi-stream multimedia presentations such as synchronous voice-overs with slideshows as single objects. This precludes the use of Generic Browser navigation functions from within such “objects” effectively regarding them as linear presentations. For a wide range of applications this approach is adequate and is the approach taken by both HyperCard and MacroMedia Director - multimedia application development tools popular on the Macintosh. It is usually the case that the presentation “player” has some controls such as those found on a VCR so limited interaction is still possible. An example of this is Apple’s Quicktime system extension [Apple 1993a, Apple 1993b] which provides limited controls for movies in any application.

The justification for this approach is twofold. Firstly, in the fast-moving field of consumer multimedia interest in the use of video and complex synchronised multi-stream temporal data has developed quite recently, in particular stimulated by advances in video compression technology. When the scope of the thesis was defined such information was not particularly common in multimedia applications (a look back to the examples of CD-i discs in chapter 4 shows no use of video for example).

Secondly, the advent of digital video has stimulated considerable interest and a whole area of new research is considering the problems of synchronising multi-stream multimedia

²¹MHEG is a standard for real-time interactive communicating applications being written by ISO’s Multimedia and Hypermedia information coding Expert Group.

presentations [e.g. Hardman et al 1993a]. The Amsterdam hypermedia model [Hardman et al 1993b] combines hypertext and multimedia models and integrates the representation of time and the temporal relations between components of a presentation. A similar evolution of the Generic Browser model will probably be necessary to enable it to represent interactive video. The scope of such modifications has prohibited their implementation for this thesis.

6.2 Representation Schemes

Underlying any information-rich application's navigation facilities is a structure holding the information. This representation scheme has a profound effect on the ease with which navigation and search functions can be built. It also affects the facility with which the author can assemble his data for the application. The expressive power of a data representation scheme governs the types of information that can be readily represented.

Early in the consideration of the Generic Browser the decision was made to adopt an object-oriented methodology.

The key points of object orientation are threefold:

encapsulation - all the data about one item stored in one place

inheritance - class members inherit default values from their parent class

object identity - every object has a unique identifier

Object-orientation is an accepted technique in hypermedia, database and artificial intelligence research. In database research, object-oriented databases are a particularly active area. In the hypermedia (and hypertext) field, an object centred view has been prevalent from the start. For example HyperCard uses an object oriented representation. Here the "card" is the basic information holder (encapsulation). Hypercard extends its object orientation to include methods as well as data so that objects contain functional information about how the data within them is processed. In current hypermedia research, the use of object oriented representations and programming continues [e.g. Hamakawa & Rekimoto 1993].

In the artificial intelligence world, a major representation system, frames, is also object centred. Frames, like objects, have data encapsulation, inheritance and object identity. Though some frame systems allow procedural attachment, most of these schemes are not as general as the *methods* of object-oriented programming.

6.2.1 Frames

What are frames ?

A frame system is a representation scheme first described by Minsky [Minsky 1981] and developed within the AI community. The basic concept is that of a flexible data structure for each item (known as a “frame”). This frame contains “slots” which contain “values”. The slots are used to store the attributes of the item being represented by a given frame. A collection of frames is sometimes known as a frame-base. There can be any number of frames in the frame-base, slots in the frame and values in each slot. A further important feature of frames is that their structure is dynamic. Run-time creation of frames is supported as is the addition and deletion of slots and values. This is in contrast to many database systems where the structure is essentially fixed at run-time.

Frames address the key features of an object oriented representation as follows:

encapsulation - the user does not access the information in the frame data structure directly, rather there is a suite of access functions to do this

inheritance - is provided by relations, in particular *is-a* and *instance* links allow class hierarchies with inheritance to be built

object identity - the frame-name forms a unique identifier through which all the information in the frame is accessed

As an example, suppose that there was a requirement to represent a collection of holiday resorts. A frame representing a resort would look as shown below:

san-tropez
instance resort
location south-of-france
hotels a b c

Figure 38. Pictorial representation of a frame.

or textually:

```
(san-tropez
  (instance resort)
  (location south-of-france)
  (hotels a b c))
```

san-tropez is the name of the frame; *instance*, *location* and *hotels* are slots; *resort*, *south-of-france*, *a*, *b* and *c* are values in their respective slots.

A relation is a link in a frame-based system. It is used to represent a relationship between two concepts. It is represented as a slot (with the name of the relation) with a value of the name of the destination frame. The frame-based system will automatically create an “inverse relation” - a slot in the destination frame pointing back to the source frame. For example if *location* in the previous example was made a relation, *south-of-france* would become a frame, linked to *san-tropez* as shown below.

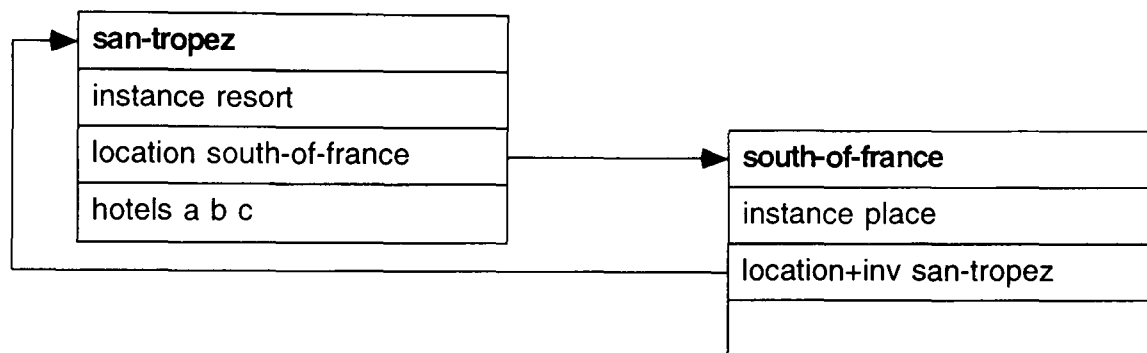


Figure 39. A relation between two frames.

If the application needs to know what resorts were in the South of France it need only get the contents of the inverse of the *location* relation slot (*location+inv*).

Why Frames ?

Having decided to pursue an object centred approach to the data representation for the Generic Browser a choice of methodology has to be made. [Halasz 1988] states

“it is clear that a liberal borrowing of ideas from frame-based and object-based technologies would be very beneficial to the advancement of hypermedia systems”.

This remark was made in the context of the desire to integrate hypermedia and AI technology and although he does not give concrete examples of the benefits he does observe that “at a high level of abstraction hypermedia systems, frame-based systems, and object-based systems present nearly identical data models”. This suggests that there is no fundamental reason why any of these representation schemes might not be used as the basis for an information browsing system.

A frame system was chosen because a compact implementation could be built to run on a CD-i player. This was based on previous experience of larger, workstation-based AI toolkits offering frames (such as KnowledgeCraft). These gave experience of the essential elements. Indeed the first implementation in this work was a prototype frame system, built to verify this assertion²².

²²XKC a LISP frame system [Cole 1991b].

Comparison with Object-Oriented Databases

In some senses frames provide a subset of the facilities provided by an object-oriented database. Whilst frames offer data inheritance, object-oriented databases also provide for method inheritance. An object's methods (which define what can be done with the data within the object) can be stored locally or inherited from the class hierarchy above.

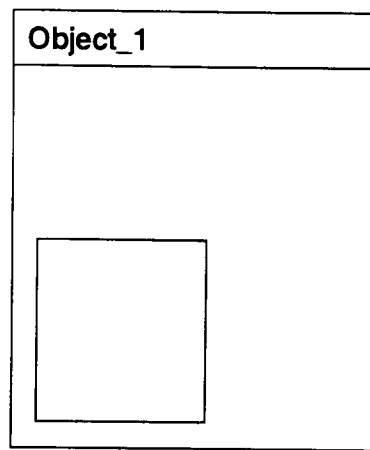
Object-oriented databases are an area of active research in the database field. As such there are no such systems available for porting to a "modest" platform such as CD-i. The complexity of this type of database also suggests that a small, efficient implementation for such a platform would not be straightforward and that some of the capabilities would be unnecessary.

6.3 Multimedia Data

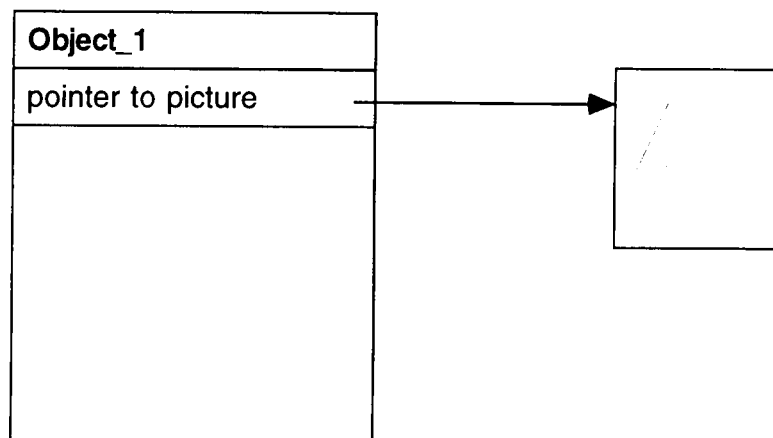
In the earlier review of the needs of consumer multimedia the importance of the representation scheme being able to represent all types of multimedia data - text, audio (music, sounds and speech), still pictures, graphics and video was presented. These are the raw components, the very "stuff", of a multimedia application.

Any given object will have one or more pieces of data of any of the above types associated with it. Since the object is being represented as a frame the multimedia data must be stored in the frame representation. This is fine from a conceptual standpoint, however, from implementation experience consideration has to be given to how such frames would actually be stored on the target machine.

Though the actual data itself (such as a bitmap for a simple image, for example) could be stored in the frame representation, it is often more convenient to store it elsewhere and reference that location in the data node. The most compelling reason for this is the sheer size of multimedia data files. On CD-i a single, full-screen, high quality image consumes 100k bytes, 1 second of stereo audio 17k bytes. It is impractical to store such items in memory. In the case of CD-i this data would be held on CD.



Picture data stored within the node



Picture data stored elsewhere (in a disk file, for example)

Figure 40. Avoiding storing picture data in a frame.

6.3.1 Content Modelling

“I keep six honest serving men
 (They taught me all I knew);
 Their names are What and Why and When
 And How and Where and Who”
 Rudyard Kipling²³

As was explained earlier when the Generic Browser data model was defined it was decided not to try to represent temporal aspects of multimedia data. However, applications will still need to represent the *content* of multimedia nodes so that they may be searched. This is more of a challenge for pictorial and audio data than for text, but considerable progress in the field of content modelling has been made. For example the BBC’s Telclass system [Evans 1987] is a concept classification scheme used to classify the content of video tapes for later retrieval. A hierarchy of concepts, each with a numeric code has been devised. Each piece of video is classified using these concepts and allocated a composite code. The

²³The Just-So Stories [1902] - The Elephant’s Child.

retrieval process simply requires concept terms of a suitable specificity (depth in the concept hierarchy) to be matched against the database. At MIT's Media Lab part of an ongoing research programme is investigating content classification schemes for movies [e.g. Davis 1993, Davenport et al 1991, Evans 1993]. Again these revolve around conceptual hierarchies, in Davis's case the concept hierarchy is iconic and covers action as well as objects and locations. Information about the shot (film type, camera position etc.) and post production effects are also included. Rather than divide the video into discrete "clips", Davis describes a multi-layered annotation which avoids this segmentation by treating video as a continuous stream of frames. So an annotation marking a change in weather will have different start and end points to, say, one indicating characters entering and leaving a scene for example.

When modelling the content of pictures (or audio) there is a trade-off between the richness of the representation and the amount of information that must be stored per asset. With this in mind, the approach taken with the Generic Browser prototypes is based on picture library classification practice. These systems use variations on "who, what, where, when" characteristics [Shatford 1986]. The Generic Browser classification scheme does not aim to be state-of-the art, rather to demonstrate how such a content model can be handled by the Generic Browser (both from a representation and retrieval point of view). Each of the characteristics (who, what, where, when) are held as slots, the values being keywords or other items in the information base.

The choice of content model is, of course, open to the application developer, the key point is that there is information associated with a picture, regardless of where it is used. There is also information that is solely dependent on the context i.e. where the picture is used - this could be represented in the interface if that were also described using frames.

The structure outlined above shows one method of encapsulating this picture specific information by associating a frame with each picture. Rather than store the *data* (or a pointer to the data) in a slot in the object frame, that slot will contain the name of a content frame. This will be a member of the relevant subclass of content; picture, audio, text and it will contain content information and a pointer to the actual data as shown below.

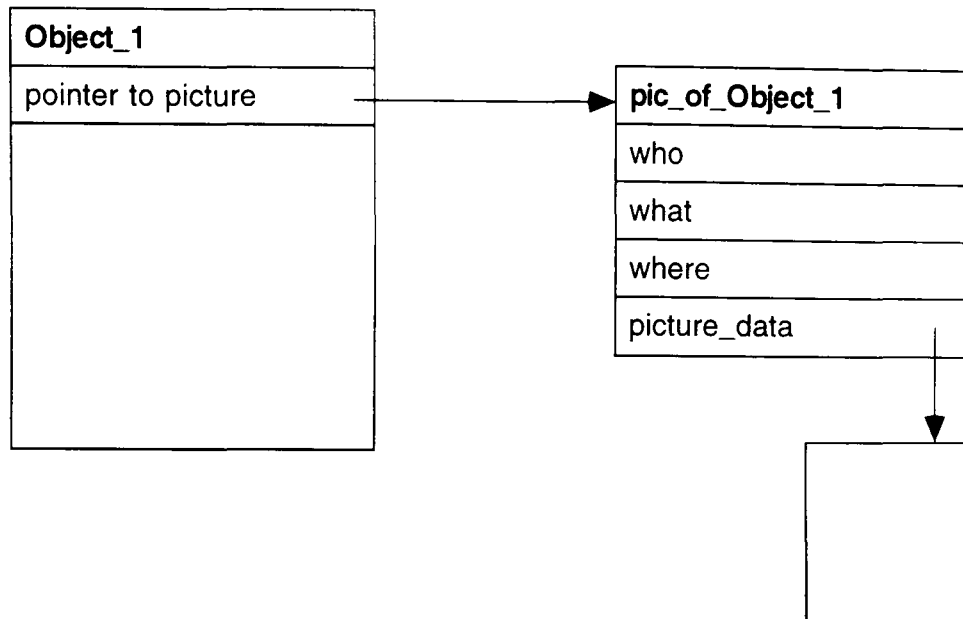


Figure 41. A picture object encapsulates the information about the picture of a physical object.

Examples of content objects in each of the multimedia data classes are shown below:

```

(a-picture
  (is-a picture)
  (who)
  (what)
  (where)
  (when)
  {view etc.}
  (picture-data))
  
```

```

(a-video
  (is-a video)
  (who)
  (what)
  (where)
  (when)
  (duration)
  {camera info, shot etc.}
  (video-data))
  
```

```

(a-sound
  (is-a audio)
  (duration)
  {content who, what, where, when}
  {speaker who, what, where, when}
  (sound-data))
  
```

```
(a-text
  (is-a text)
  (text-data))
```

6.4 Models of Browsing and Search

The Generic Browser model of browsing defines it as a navigational process moving from one node to another either by following links or predefined paths. The browsing activities can encompass the whole set of nodes or just a subset. Annotation of nodes is specified.

In contrast the Generic Browser model of search views this as a process operating on a collection of nodes rather than a node at a time. Search is a filtering process which creates subsets of “hits”. These hits need not be perfect matches. Both the characteristics of the data and the search terms in the query may be weighted and queries can be negative.

6.5 Browser Information Structures

Structure is vital to browsing. Without structure the information can only be processed serially, if at all. Two types of structuring are described below. The first is the hyperlink structure, now well established as a support for browsing of information. A second structure, the path, complements hyperlinks by supporting authored paths (such as guided tours) through the information.

Users build a mental model of the browsable structure. This may be exact, allowing direct navigation to sought after information, or just a “feel” for the shape of the information space. In both cases, though, the user doesn’t feel lost and is able to navigate. Indeed personal experience of an early prototype frame editor²⁴ highlights the problem nicely. In this editor the order that the frames were presented to the user in a linear browser was that of the hashed frame identifiers. This had two main disadvantages for the user: firstly the order of the frames in the browser changed when frames were added or deleted and secondly there was no easily observed order such as alphabetic and so it was impossible to predict where a given frame would occur within the sequence. This unsatisfactory ordering was soon changed but this simple example highlights how important it is for the users of a browser to be able to form a mental model of the structure.

Interesting multimedia applications will, no doubt, contain a mixture of the freedom of hyperlinks where the user decides whether he wishes to explore or not, and the more constrained path browsing in which the author has decided on the progression through the

²⁴ The FRED frame editor was developed by Martin Shiels at PRL.

information. Browsing subsets of the whole information space is another way to reduce the cognitive load imposed by the freedom of browsing. These three mechanisms, hyperlinks, paths and subsets are the structural primitives of the Generic Browser.

6.5.1 Hyperlinks

Vannevar Bush is usually credited with the origin of the concept of hyperlinks. In his paper entitled “As We May Think” [Bush 1945], he describes a machine he calls a memex. This opto-mechanical device allows items of information to be linked together by

“[...] associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of the memex. The process of tying two items together is the important thing.”

The associative links that Bush describes are equivalent to current “hyperlinks”. Structuring information in this way is attractive because they offer the opportunity to structure it in a similar way to that which the human mind is thought to. This should make such information easier to assimilate and remember.

From this node and link paradigm has grown first hypertext and then hypermedia. The basic principle remains the same. Each node contains a piece of information. It is linked to related information by the hyperlinks. The user browses such a structure by following the links that interest him. For example, in a hypertext, various keywords may be highlighted. Selecting a highlighted word causes a link to be traversed to another node containing related information.

Hyperlinks are now well established as a browsable structure and must, therefore, form part of any generalised browsing framework. However, in their basic form hyperlinks are rather crude and a possible refinement, typed hyperlinks, is described below.

Generic Link Types

Hypertext systems often have only one link type. Clicking on a “hot” concept, word etc. traverses the link to a new display of more information about the concept or word selected. One could thus view the link as a “more information” or “elaboration” relation. Of course the user has no way of knowing what sort of additional information will be available except by traversing the link and having a look. In some ways this lack of precision is in harmony with the idea of using the hypermedia system to explore the information space. In this case an element of serendipity occasioned by finding the unexpected at the end of a hyperlink is probably acceptable.

A less exploratory access mechanism would need to have the links annotated in such a way that the user was aware of what he would find by traversing a link without actually having

to do so. In text, a simple example is “for more information see ... on page ...”. Here the user knows that he will get more detail by turning to the specified page. If the text said “for background information see ...” then the reader is again able to decide whether to go and look or not. Contrast this with hypertext where all the user has is some indication of “hot words”. He knows that there is some related information at the end of the link but not how relevant or how it is related to what he is currently reading. [Andersson 1991] describes an annotation facility as being essential so that users of trails of links would have some idea of why they were following the links and what they should expect to see at a given node.

Journals such as New Scientist frequently use “extra information” boxes. These are chunks of detailed information (or different perspectives on the main story) which are printed within the article, differentiated from it by a coloured background. These boxes are referenced from the text and the reader can glance at the box to see what it contains before deciding whether to divert to read the box there and then, wait until the end of the article or not read it at all.

Considering these information boxes in more detail it becomes apparent that there are a number of different types of information presented in this way. Background to a story may be kept separate to allow a fast pace to the main thread. Extra detail on various aspects of the feature can be kept separate from the main text so that readers who want a less detailed view can get one by reading the main text alone. Contrasting arguments are also offered in this way.

Of course the extra information does not have to be textual. Figures can also be regarded as supporting information which the reader is free to view or not as he wishes.

One of the problems with hypermedia is that there is no “story”; no narrative structure (except within the nodes where the information ought to be coherent). To address this problem an alternative structure (paths) is introduced in the next section. Suffice to say for the moment that paths are sequences of nodes and so can capture the ordering required by an author. However there is a danger of simply regarding these two structuring methodologies as mutually exclusive. Interesting structures combining both techniques are possible. For example a path describing an event (which, because of its complexity, requires a number of information nodes) could have links at each node to related information. If these links were typed, the author would be able to indicate to the user what sort of a diversion he would get if he followed the link. Provision should be made to allow return to the original path at any time. This type of structure allows an author to “get his point across” at the same time giving the reader freedom to get extra information if he wishes in a more exploratory way.

If such a combined use is to be made of links and paths both structuring methods need to use a common representation i.e. they must both work with the same type of nodes. In the case of the Generic Browser this uniformity is provided by using frames.

Where links are annotated the user will have to interrupt reading the information to evaluate whether an embedded link is worth following. If the author can use arbitrary link types the user will have no foreknowledge of the possible links (the vocabulary of link types if you will) and so will have a high cognitive load imposed upon him by their presence. An alternative worthy of consideration is to restrict the types of links to a standardised set.

Whatever set of links is chosen, the destination of the links must be hypermedia objects. In other words the links must be suitable for referencing multimedia data. Such a set must be both flexible enough for the author to capture the relationships between the data and yet small enough so that the differences between the relations in the set are readily understood.

There are, of course, many possible sets of relations which might be suitable for this task. One such set is described by [Parsaye et al 1989 pp. 244-246]. An alternative approach is described below.

During work researching the automatic addition of illustrations to computer-generated textual explanations, a similar problem was encountered; that of describing the relationship between an illustration and its accompanying text.

A number of possible relation sets were considered including functional relations such as those proposed by [Brody 1984, Levie & Lentz 1982, Duchastel & Waller 1979, Alesandrini 1984] and, more promising, rhetorical relations [Grimes 1975 pp. 207-209, Hobbs 1978]. One well researched set of rhetorical relations is that used in Rhetorical Structure Theory (RST) [Mann & Thompson 1987]. RST is a theory for describing the relations between the components of multisentential text. This relational structure has been adapted to form the basis for planning the generation of coherent multisentential text. [e.g. Hovy 1988, Moore 1989]. More recently, interest has also been shown in widening the scope of such generation systems with the aim of allowing multimedia responses to be generated [e.g. André & Rist 1990]. From the RST set of twenty three relations nine have potential to link pictures to text. They are:

- circumstance
- solutionhood
- elaboration
- background
- evidence
- restatement
- summary
- sequence
- contrast

It seems likely that a subset of these might be appropriate offering application independence and a range of types of attached information.

To assess the suitability of RST for describing the links between illustrations and text a selection of illustrated documents were analysed. It was found that it was certainly possible to use RST relations to describe how the illustrations fitted into the rest of the rhetorical structure. The main problem encountered was that of the ambiguity of pictures.

However, variety is the spice of life and the author tends to agree with [DeRose 1989] who opines that “[...] no closed taxonomy is likely to be adequate for all future purposes, so coining new types should be possible”. DeRose’s solution to the problems inherent in such freedom - restricting new types to being sub-types of a standard taxonomy is perhaps trusting too much in the ease with which such a taxonomy could be created.

From the Generic Browser perspective it seems very desirable to allow the author freedom to choose the types of links he wishes to use in a given application. If future research comes up with a “universal” set of links such a system will be able to support them.

6.5.2 Paths

The concept of a path or ordered collection of nodes was introduced in Bush's original paper [Bush 1945]. As well as associative links, Bush envisaged that linked items would form a "trail".

"when numerous items have been thus joined together to form a trail, they can be reviewed in turn, rapidly or slowly, [...] It is exactly as though the physical items had been gathered together from widely separated sources and bound together to form a new book. It is more than this, for any item can be joined into numerous trails."

Such a trail encapsulates two ideas; a history of where one has been - the literal trail, the route that was taken and also the idea of a route that one might take. The latter when, as Bush suggests, the trail is given to someone else. For the Generic Browser a single object, a path, is used to represent any ordered sequence of nodes and can thus represent both these types of sequence.

The idea of a path derives from the commonly occurring need to browse information sequentially. A typical example of a path is a "guided tour". This type of path is frequently found in information systems for example [Hammond & Allinson 1988] and the tours in the "Tell Me Why" CD-i disc. As will be discussed further below, a path should be thought of as an object in its own right, providing a way of attaching "meta" information to a sequence of nodes. Whilst a path could be regarded as just a simple list of items for display, to be more generally useful paths must provide additional capabilities.

In its simplest form a path supports a "page flicking" metaphor although a wide variety of interpretations of this basic interaction are possible, built upon the basic functionality of the path.

Paths Contrasted with Hyperlinks

The path is a navigation structure that defines which nodes the viewer should visit and does not depend on the content of the nodes to do this. This is in contrast to hyperlinks where the link information is held in each node (see diagrams below)

path_1			
Object_1	Object_5	Object_3	Object_1

Figure 42. Objects on a path.

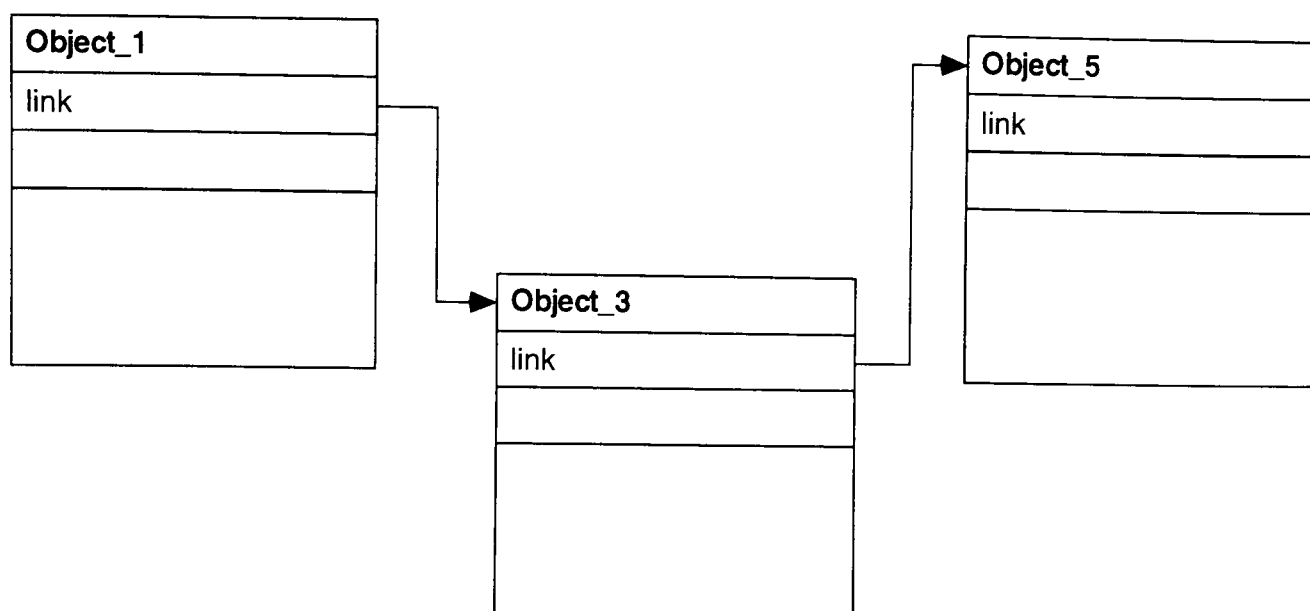


Figure 43. Objects connected by hyperlinks.

Paths are a location representation structure with just the nodes and their browse order stored. Link browsing on the other hand concentrates on the relationships between the nodes. With paths there can be meta information which pertains to all the nodes on the path. There is no such information available for a sequence of nodes linked by hyperlinks.

Bush's original trails were a record of the links traversed to get a particular view of some part of the information base. In the Generic Browser it is the nodes to visit rather than how to get from node to node that forms the path construct. One advantage of Bush's system is that the additional information conferred by the link types forms part of the "story".

Hypertext may not be coherent, not a narrative. There is a need to be able to construct paths which convey a coherent message. This is similar to the process required when converting normal text into hypertext [e.g. Gotel 1990] hence the interest in the "typed" hyperlinks described above.

Another distinction between paths and hyperlinks is that a path structure exists separate from the data items that it links together and thus an individual node can appear on many paths. Paths can be considered as "views" or alternative structures of the data space. Links between nodes are part of the node structure and stored in the nodes themselves. Multiple hyperlink structures with common nodes are more difficult to represent.

It is important to realise the difference between a hyperlink structure as presented to a user and the use of a linking or relation structure for implementation purposes. From the point of view of providing a comprehensive set of browsing functions both paths and hyperlinks must be provided. From an implementation perspective either or both of these types of browsing mechanism may be represented using relational links within the knowledgebase. Indeed Bush's trails are actually constituted from items joined by associative links but with a trail there is an order imposed on the network of nodes and the user can follow the trail.

The combination of hyperlinks and paths form a rich set of structuring primitives and both are required in the Generic Browser.

Paths as Objects

A path is an object containing not just the list of nodes actually on a path but additional information which is specific to the path as a whole. Recall Vannevar Bush's trails which he described as "items [...] bound together to form a new book".

The path structure is, in a sense, meta to the data - any individual node says nothing about the content that is inherent in the path as a whole. For example the path may give narrative form to a collection of nodes, express an opinion, provide a contrast and so on.

Because a path is a browsable structure as opposed to just a collection of data, such as a set, information about how that browsing process behaves can be stored in the path object. For simple paths the behaviour at the ends of the path needs to be stored. The operation of the next and previous commands are defined by this information. The following alternatives should be provided:

- stop when the end (or beginning) of the path is reached

- restart at the end (or beginning) of the path, continuing browsing from the start (or end)

- goto a specified node on reaching the end (or start) of the path.

This last option allows paths to be chained together. Another option, that of calling a function at the end of a path, was considered. However, as will be seen in the next section the constraints of the implementation prohibited its inclusion in the Generic Browser.

One possibility not listed above is that of reversing the direction of browsing at the ends of the sequence. Whilst this is a possibility it means that "next" and "previous" alternate their meanings. This is an unnecessary confusion so this option is not included (it is straightforward to achieve this functionality in the application using the basic options described above).

In the Generic Browser paths are stored in frames. This makes paths objects in their own right and preserves their uniformity with other structures in the Generic Browser.

In some instances, such as a slideshow, the path will be traversed automatically with no intervention from the user. For some narrative paths only forward browsing may be allowed (albeit at a pace determined by the user). For other paths the user may be able to

browse both forward and backward at will. These differing types of functionality can all be provided using the basic functions described below

Functions for Browsing Paths

Since the path is used for navigation (browsing) functions must exist to traverse the path. Since a path is an ordered sequence “previous” and “next” functions fulfil this need. As noted above, the behaviour of these functions at the ends of the sequence is a property of the path. Also it must be possible to add and delete nodes from the path. Since the path is conceptually an object in its own right, it must be possible to access the contents of the path (i.e. all the nodes on the path in order). For each path, the current position on that path must be stored. This is known as the *current_frame_on_path* and is another example of data which applies to the path as a whole i.e. as a single object. The *previous* and *next* functions update this information and a function to access the *current_frame_on_path* provides the means by which the path nodes are accessed.

Expanding the Path Concept

Further consideration of the requirements of path-type structures reveals some areas which are not covered by the linear paths discussed so far. The issues raised by branching paths and related issues such as the representation of temporal data within path objects are considered below.

Branching Paths

The rationale of path structures is that they provide an author with a high level construct for representing sequences of nodes. These can be used for guided tours, explanations, slideshows and so on. Although such structures can be encoded functionally as part of the application code, such an approach is not general and hides the node structure (part of the application *data*) within the program code. Having a path construct enables the author to work simply with the data, secure in the knowledge that functions exist to traverse the structures that he is authoring.

Given these advantages it is natural to wish to continue to describe more complex structures in the same way. Branching paths are the next step in complexity. Here, the user may be presented with a number of alternatives at various points along the path. These branches may rejoin the “main path” or form alternative “endings”. For example

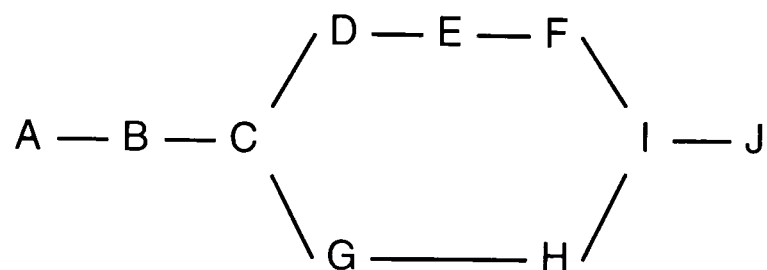


Figure 44. A branching path

In this example of a branching path there are two branch points C and I. The two branches DEF and GH rejoin but this is not a requirement and tree structures are possible.

These objects could be used for a number of types of application:

interactive stories - these have existed in book form for some time [e.g. Harrison 1985, Packard 1993] and have been put on computer using hypertext which can handle all the jumps from one text (or multimedia) fragment to another. Such a multi-threaded plot structure also maps directly onto a branching path structure.

documentaries - An evolution of linear documentaries might well allow limited freedom to choose which aspects of a topic will be viewed (whilst still retaining the author’s sequencing information).

quizzes - computer-based quizzes are often based around some form of multiple choice mechanism. These choices could be represented as the branches of a branching path, each branch containing a message as to whether that particular answer is correct or not, even, perhaps, why it is (or is not). The actual questions would be the branch points.

A key issue with branching paths, and one which ties representation issues with interface issues, is that of how the user should be presented with the choices at the branch point. From the representation perspective provision needs to be made in the branching path structure so that presentation information can be stored with the appropriate branches of the path. From an interface viewpoint there must be a mechanism to display the choice information, capture the user’s choice and trigger the appropriate branch in the browser. As an example suppose that the branch choices are to be offered to the user as a menu. When a choice point is reached, the menu strings for each branch (which must be stored with the branch) must be extracted to build the menu. When the user makes a selection, his

choice must be decoded against the menu items to find which branch is to be traversed and the user informed accordingly.

Two types of branch selection mechanisms need to be catered for:

user selection (via buttons, menu or other interface widgets),

automatic selection under the control of the application - for example random selection, selection of branch from those not yet visited, selection of branch that most closely matches the current theme etc.

A structure for branching paths

Having considered the requirements of branching paths, their representation needs to be drawn up. One variation on this theme is explored below²⁵. In this representation the branch point is signalled by the path item being a list; thus

(a ((b c) (d e)) f g)

represents

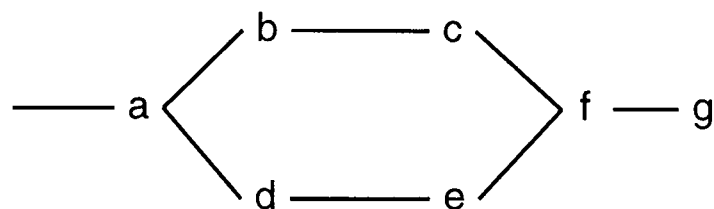


Figure 45. A branching path structure

When the path-pointer (the user's position on the path) encounters a list the user would have to follow one of the sublists. At the end of that sublist the next item in the main list would be visited.

In the case of a "diversion" from the main path the structure looks as follows:

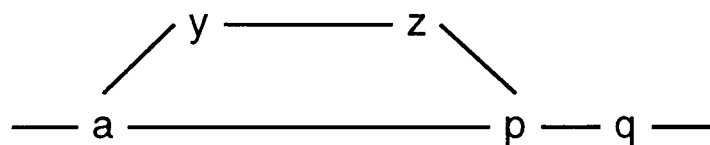


Figure 46. A "diversion" from a path

²⁵Although limited prototyping of branching paths has been done, this representation and the other aspects of branching paths are still at an early stage of development.

and is represented:

```
(a ((y z) ()) p q)
```

At this point there is no means of representing the message to be displayed at the branch point. To solve this problem there could be a sub-type of path called, perhaps, "branch" which inherits all the path data plus has information for presenting itself as an option at a path choice point. So

```
(a ((b c) (d e)) f g)
```

would be represented

```
(a (B1 B2) f g)
```

where

```
(B1 (is-a branch) (presentation-info xx) (path b c))
```

```
(B2 (is-a branch) (presentation-info yy) (path d e))
```

Considering this representation there is still a need for a choice point frame to contain information on how the choices are to be presented and selected. Maximum flexibility needs to be provided at this choice point. Ideally there needs to be a hook so that a user-defined function can be called.

```
(a B f g)
```

```
(B (is-a choice_point) (has_choices B1 B2)  
  (selection_method menu))
```

```
(B1 (is-a branch) (path b c))
```

```
(B2 (is-a branch) (path d e))
```

The next problem to consider is how to attach extra data such as the message to display for each of the choices. These messages are each associated with a branch and so could be stored in the branch path frame. For example

```
(B1 (is-a branch) (choice-entry "choice 1")  
  (path b c))
```

Given that there needs to be a choice point frame, this could be the first item in the branch list. One possible structure for a choice point would include a list of the branches. Alternatively the actual branches could follow the choice point frame in the branch list.

What happens at the end of the branch is also important. Since the branch is a path it is possible to set the “at end slot” so that the branch can never be left (by browsing). If the end is passed, the user should return to the main path after the branch. Thus the “at end” of a branch should allow for returning to the previous path. Similarly, if backwards traversal of the path using the *previous_on_path* function is enabled, the “at start” of a branch should return to the choice point.

This raises the problem of what to do when a recombination node, where several branches meet, is reached using the *previous_on_path* function - should this be treated as a choice point too for example?

There are a number of options :

branching paths are completely bi-directional

backwards traversal of branching paths is inhibited

previous_on_path skips branches and goes back to the previous choice point

One could have the behaviour of the path following mechanism dependent on the type of frame in the path e.g. data item, choice point etc.

This gives

```
(choice_point
  (sub_paths <branch_1> <branch_2>)
  (choice_presentation <menu> | <random>
    | <auto> | ...))

(branch
  (is-a path)
  (path)
  (at_end)
  (at_start)
  (choice_entry "choice string"))
```

In order to cope with nested paths there needs to be some kind of “path return stack” to keep track of the paths left so that they can be rejoined at the end of the diversion.

Another aspect of a path being an object in its own right is that it is possible to search paths. One might wish to find all paths that go through specified nodes for example. This was one of the reasons why storing paths as a single list in the path frame was found to be

preferable to using the “forward & back” slots mechanism which was used in the first prototype.

To summarise, a methodology and representation for branching paths has been outlined here. The key aspects are having a branch sub-type of path which can appear in the path list, a modified path traversal mechanism that is sensitive to the type of object (node or branch) next on the path and a path stack on which the current path is pushed whenever a branch is followed and popped when the end of the branch is reached. A full implementation of branching paths has not been done, however preliminary prototyping suggests that it should be a straightforward enhancement to the Generic Browser.

6.6 Navigation

The process of browsing data held in frames is straightforward. An important concept for browsing in the Generic Browser is that of the “current frame”. This is the “you are here” of the Generic Browser. All the frame browsing functions update this current frame. Since the “current frame” is simply a frame identifier the interface component can use this to extract information from the current frame and display it for the user. The process of browsing has now become changing the current frame, either by following links or traversing path structures and then displaying information from the new current frame. The concept of a current frame ties in with the idea of a user having a focus whilst browsing, an item that is receiving his attention - this focus is the current frame.

6.7 Marking

“Read, mark, learn and inwardly digest”
The Book of Common Prayer

Marking is a powerful concept. It allows items in the database to be identified as belonging to subsets, created by either the system or the user of the application. Subset browsing has already been identified as an important means of avoiding the user losing his way in the information space. When marking is applied to the navigation structure of the application, perhaps on a screen by screen basis, the user can use marks as placeholders in the same way as bookmarks are used whilst reading, enabling a particular “place” in the application to be returned to at a later date without the need for tedious backtracking [e.g. Canter et al 1985].

Having identified the need for marking facilities it is necessary to consider what kind of support for marking should be provided. An example of the most basic form of marking is that provided in HyperCard. Here there is just one mark type which can be placed on any card. Marks can also be placed on cards that match a criterion using HyperTalk commands such as

mark cards where "Apple" is in background field
"Company" .

The unmark command can also be used with search patterns. Hypercard allows a number of operations to be performed on marked cards such as printing marked cards, showing marked cards and go to next and previous marked cards.

A generalisation of the mark concept would allow :-

an arbitrary number of named marks on any frame

values to be associated with marks

treatment of a set of similarly marked objects as a single object

browsing of marked sets

This would allow subsets to be created by marking. Such collections of frames could be browsed or form a restricted part of the frame-base for other operations.

As will be seen later in this section, marking forms the basis for the integration of search. By using marking itself, the Generic Browser is able to identify search hits in such a way that they can be immediately offered for browsing.

The provision of marking as the means of creating subsets of the information in the frame-base makes it desirable to provide basic set operations so that such subsets can be combined.

6.8 History Mechanisms

A history is a record of where the user has been or what he has done. In this section, a variety of "history" mechanisms are discussed. Uses of history information such as allowing the user to retrace his steps if he gets lost and offering unvisited information to him will be discussed. This will be followed by a description of the facilities offered by the Generic Browser.

6.8.1 Navigational History

In this type of history the system records *where* the user has been, usually in the order in which he visited each location. The primary use of this information is for providing some form of backtracking within an application often via a "go back" button. This allows the user to retrace his steps, returning to previously visited locations in reverse chronological order. This function requires that a list of all locations visited is maintained by all functions which change the current location within the application. In many interactions it will be

undesirable to keep all the locations visited and so a maximum backtracking depth may be enforced restricting the user to being able to only go back a fixed number of steps. In applications where branches or decision points are common, another strategy for restricting the backtracking is to allow backtracking to the last decision point or branch, either via the intermediate steps or directly. Of course, *go back* functions can also work on a data centred view rather than a location-based view. In this case, backtracking simply retraces the user's step through the data items visited.

An alternative backtracking mechanism gives the author more control over where the user goes when he chooses to backtrack. Rather than the system recording where the user has been, go back information in each node points back to the location the author wishes the user to return to. The problem with this mechanism is that it does not encapsulate any form of context i.e. how the user got to the current node. In the case of there being a number of routes to a given node it is reasonable to expect that there may be a variety of suitable backtrack destinations. This lack of generality makes this type of backtracking unattractive for the Generic Browser although it could be implemented for specific applications using information stored in the data or location frames.

This information is extra-nodal.

6.8.2 Statistical Information

The system records *how often* an item or location has been visited. For example, it may be useful for an application to know whether, and if so how many times, a given location or data item has been visited. Similarly, statistical information can be recorded for each function though this is probably of more interest to the program developer than of use within the application itself. For the Generic Browser, such statistical information should only be recorded for the items visited.

This information exists within the nodes.

6.8.3 Temporal Information

The system records *when* an item or location was visited. Though a navigational history records where a user has been, in order, it does not record when a given location was visited. In some applications this may be useful information.

This information also resides within each node.

6.8.4 Action History

The system records *what* happened. Related to the *go back* function an *undo* function is most commonly found in editors. It actually undoes the effects of previous actions. Thus

rather than a location or items visited history, it is necessary to maintain a record of all the functions called via the user's actions so that their effects may be undone when the *undo* command is invoked.

This information exists outside the nodes.

Though undo functionality is useful in editing-type applications it is of less use in general information browsing and searching applications. Provision of undo capability imposes considerable overheads on all the functions used by the application in that they must be reversible and, when they are called, record all the information necessary for reversal at some later time. For these reasons, it has been decided not to offer undo functionality in the Generic Browser.

A particular item to note from the above is that some history information is stored in a separate history data structure (navigational and action history) and some is placed within the node itself (statistical and temporal history).

6.9 Search Methods

In this section a number of different search facilities will be examined and their utility for consumer multimedia systems investigated. The one chosen for the Generic Browser, the vector space model, will be examined in detail. It is not the purpose of this work to develop new search algorithms or techniques, rather it is to show how search can be integrated with browsing for consumer applications. Accordingly a representative search mechanism, vector search, has been chosen to illustrate how an established technique can be used in the framework of the Generic Browser. With consumer applications largely occupying the "middle ground" of the browse-search continuum, applications which are pure search will be rare. Far more likely is that search will be used to establish a subset of the database for subsequent browsing as was shown in the previous chapter.

The type of search described here matches the characteristics of objects against a query specification. This allows all media types to be treated in the same manner. Of course there are specialist search techniques for the different media. For example, text search is addressed by full-text retrieval techniques [Salton 1989 pp. 255-274]. Pictures and video can be accessed by the pattern and region matching techniques which are still a very active area of research in areas such as image compression and object recognition. Both of these examples show the search process operating on the data (content) itself rather than on a representation of the content as is the case with searching content models.

6.9.1 Basic Search Functions

The rôle of search within the Generic Browser is to extract a subset of the data that can be used as the basis for further searches or as a set of objects to be browsed.

A frame-based system is not a database and since no search functions are provided basic ones such as those listed below are required:

get all frames with a particular slot

get all frames with a particular value in a specified slot

As marking is used to generate browsable structures within the frame-base, equivalent functions that *mark* the resulting frames rather than returning them are also required. In addition it is desirable to be able to perform these functions on subsets of the database. Accordingly functions that search either a class or a marked set are also required.

6.9.2 Vector Space Model

The basic search functions described above are used for locating items that exactly match the search criteria. Other search tasks require the location of the “best match” even if this is not an exact match. Many consumer search tasks, such as choice of a holiday for example, fall into this category. Here the user homes in on a short-list of potential holidays by choosing those characteristics that he thinks are desirable or those which he definitely does not want (see **negative queries** below). The search mechanism produces a ranked list of hits so that the user is free to choose from those items which match (or nearly match) his requirements.

The method chosen to provide this type of search is based on the “vector-space model” [Salton 1989 pp. 313-326] which is often used in text retrieval systems. This methodology was also used with success in the Apple Guides prototype [Don et al 1991].

How it Works

The vector-space search method works as follows. Each item is described by a number of characteristics. The presence or absence of these characteristics is recorded for each item, usually as a set of vectors where I_{ij} records the presence or absence of characteristic j in item i . For simple systems, this is a vector of 0's and 1's, but more complex systems can be built where weights for each characteristic are recorded. The query is expressed in the same way - as a query vector Q_j . The search process consists of evaluating a similarity measure between each item vector and the query vector. This measure is usually some variation of the vector distance (such as the inner product) between the query and items in

an n dimensional space (where n is the number of characteristics). The hits can then be ranked according to their similarity with the query.

An attractive feature of the vector search method is that the query and items are stored in the same way. Thus in the Generic Browser, in common with the other objects, the items are frames as are the queries.

Similarity Measure

[Noreault et al 1981] evaluated the effectiveness of 24 (out of a possible 67) similarity measures. They concluded that at best there was a 20% improvement compared with a random ordering of the output and that a large number of the measures performed almost as well as the best. Given that, for efficiency reasons, the simplest similarity measure is very attractive and that, for consumer applications, accuracy is less important than execution speed, the inner product similarity measure has been chosen for use in the Generic Browser

$$S_{xy} = \sum_{i=1}^n x_i \cdot y_i$$

where

$$x_i \cdot y_i = x_i y_i \cos(\alpha)$$

where α is the angle between the term vectors. When the terms are linearly independent their vectors are orthogonal (at 90°) to each other. $\cos(90^\circ) = 0$ so the formula simplifies to

$$S_{xy} = \sum_{i=1}^n x_i y_i$$

For comparison, the cosine similarity measure is calculated as follows:

$$\frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2 \cdot \sum_{i=1}^n y_i^2}}$$

This is substantially more complex than the inner product and thus takes more computation time. If greater accuracy were required a different similarity measure could easily be substituted for the inner product.

In the Generic Browser the result of the vector search is that each frame “hit” is *marked* with its similarity to the search query, making a marked set of hits.

Advantages and Disadvantages

Salton describes three main advantages of the vector-space search method [Salton 1989 pp. 317-319]

- 1) Items can be ordered in decreasing order of similarity with the query allowing them to be displayed in decreasing order of presumed importance.
- 2) The size of the retrieved set can be adapted to suit the user. For casual users only the top few items need be displayed. A more exhaustive list may be supplied to specialists.
- 3) Items retrieved early in the search, which are most similar to the query, can be used to improve the query using relevance feedback. (This is of more use in text retrieval.)

Salton also compares the vector-space model with two other information retrieval methods, Boolean model and probabilistic model and finds the vector-space model “the simplest to use and in some ways the most productive”.

Simply searching for the presence or absence of particular characteristics is a fairly coarse mechanism. More subtlety can be provided by weighting. Weights are values attached to the characteristics in the data frames and/or the query (the same mechanism is used for both data and query, a useful uniformity of representation). This allows the author to represent degrees of the characteristics when describing objects. For example, he may wish to describe a resort as fairly quiet. It is not sensible to try to give an exact amount of quietness, all the same, it must be possible to indicate that this particular resort is quieter than another. This is where weights are used, the relative quietness of the resort being denoted by, say, one having a quietness of 0.3, the other 0.5.

Similarly, it is often the case that a user querying a database with more than one characteristic may want to indicate the relative importance of particular characteristics. Weights can be used here in the query as well.

The vector space model's chief disadvantage is that it requires the characteristics to be linearly independent and if they are not it produces inaccurate similarity measures. This may not be a problem when using this method for inexact searching because the user will not be expecting complete accuracy, indeed, this restriction is widely ignored in the information retrieval field without adversely affecting the methods utility [Raghavan & Wong 1986].

6.9.3 Negative Queries

The basic idea behind negative queries is the desire to be able to specify characteristics which the user does *not* want in the items found by a search. There are a number of approaches that are possible.

A simple solution is to have a negative search function which returns all the items in the search set that do not contain specified characteristics. This can be combined with the normal search functions to allow the user to specify both characteristics that should and should not be present in a hit.

This method works fine for Boolean search methods but the situation becomes more complex when inexact matches are permitted.

One possibility is that negative query terms should be treated the same as positive ones. That is to say that a negative term *decreases* the similarity measure rather than increasing it when that term is found in a data item. This captures the idea that the user “probably doesn’t want” a particular characteristic. However, unless weighting is allowed, if the number of positive terms was large compared with the negative ones the effect of the negative terms would, in many searches, be overwhelmed. This is unlikely to be transparent to a typical consumer who does not expect to see items which have characteristics that he has flagged that he doesn’t want. For these situations a pure rejection filter is most appropriate.

For the Generic Browser, in keeping with its philosophy of being general, both of these facilities should be offered.

6.9.4 The Integration of Search with Browsing

As has already been shown number of authors have considered the potential benefits of combining some form of search facility with the browsing mechanism usually found in hypermedia systems particularly hypertext [e.g. Faloutsos 1990].

An integrated system is described in [Jerke et al 1990]. It allows formal search queries to be used to find suitable locations in the information network for future browsing. In the particular implementation they describe, the results of the search (holiday destinations) are presented on a map from where the user selects one as a starting point for multimedia browsing. The authors found the basic concept of providing both access mechanisms useful in this type of application.

For the Generic Browser the aim is to integrate the search mechanism closely with that provided for browsing. To this end the search component generates the results of the

search as a marked set. This means that these items can be browsed with no extension to the browsing functions already specified.

At this point it is important to note where compromises in either the browse or search components have been made in order to provide the level of integration required by the Generic Browser. The main compromise has been in the search engine. For efficiency reasons most database systems operate on highly optimised data representations which are geared to the requirements of the search mechanisms [Salton 1989]. For example some systems store the data in a tree structure where the nodes are positioned in the tree according to the search key. This reduces the search time. Others use so-called hash tables to reduce the amount of space required for the index structure. In the Generic Browser the data representation has to be amenable to browsing as well as search and the resultant scheme is less suited to efficient search. Some of this inefficiency could be alleviated by performing the search operations on the underlying “internal” datastructures (below the frame level). It would also be possible to build indexes of the type mentioned above for static parts of the database.

6.9.5 Function Search

Work on the Cruise Browser²⁶ revealed a missing class of search operations, that of comparing a value associated with each item with one in the search query. For example, there might be a requirement to extract all cruises costing less than £200 per day. Here the search function must compare the search specification of £200 with the daily cost in each cruise frame, only generating a hit when the target frame has a lower cost. This is an example of the class of search functions that use a comparison function. This function is Boolean and returns true when the query value compared with the target value is true. This generates a “hit”. Because standard Boolean comparison operators (less than, greater than etc.) are likely to be required in many applications these should form a set of provided functions.

6.9.6 Combining Search Operators

[Fox et al 1988] describe a modified version of the vector search model where sub-matrices within the document matrix are processed by different search operators. For example, the normal similarity measure could be used for some terms, a weighted version on others and a function applied to a third part of each document vector.

²⁶The Cruise Browser an electronic version of a cruise holiday brochure, described in detail in chapter 9. was built as part of the Homestead Esprit project.

This provides an attractive way to combine search operators. If such a mechanism were provided the type of search to be used would have to be indicated in the query frame.

6.9.7 Variations of Multimedia Search Mechanisms

In [Faloutsos et al 1990] a number of variations on the basic search theme have been suggested for further research. These are considered below in the context of how the Generic Browser addresses these issues.

Operations on search result lists - because the Generic Browser produces a marked set containing the search hits it is straightforward to process this list. As will be seen in the next chapter this is particularly straightforward using the LISP-like list processing functions that form part of the frame-based system (Cframes).

Relevance weighting and ranked output - ranked output is already part of the vector search functions provided by the Generic Browser and that type of search is also amenable to relevance feedback [Salton 1989 pp. 319-326] though this is more important where the number of items is very large as it is in text retrieval applications. However weighting does reduce the efficiency of the search because of the extra processing required.

Restricted search domain and neighbourhood searches - marked sets and classes provide two varieties of restricted search domain. Neighbourhood searches are not catered for explicitly in the Generic Browser though producing a marked set from neighbouring nodes would allow this type of search. Such a marked set would have to be created by finding all the links leading from the current frame and marking their destinations. This process could be repeated for the nodes so found to spread the "net" wider.

String search within an article, string specification by pointing to the current text and incorporation of string search in the authoring tool - these refer to string search which is only really applicable to a text-based system rather than the more general frame-based system of the Generic Browser. It may be necessary to include string search on slot contents, however, if the database was mainly composed of text held in frames, experience to-date suggests that text fragments are more conveniently stored as files rather than as the contents of a slot. A separate string search routine could be used to scan these files.

Although it is possible to do string searches on the text in a multimedia information base, in all probability there will not be very much text. This is because text is less easy to read on a tv screen than a computer monitor and users are more likely to want audio and pictures than screenfuls of text. The lack of a keyboard in consumer multimedia systems such as CD-i makes the interface to string search very difficult to provide cleanly.

Speeding the search - obviously this is a goal for any software developer and the Generic Browser is no exception. In the prototyping phase of the development of the Generic Browser, speed has not had the highest priority though a first pass at improving it will be described.

6.10 Authoring Perspective

There are essentially two “customers” of the Generic Browser. So far we have concentrated on the needs of the user of a browsing application. This person uses the Generic Browser indirectly as he uses the application. If he is able to do what he wants then the combination of the skill of the author with the functionality of the Generic Browser has been successful. The other customer is the author of the application. He makes use of the basic building blocks provided by the Generic Browser to construct the application for the end-user. The Generic Browser is one level below authoring tool kits in the authoring process since it provides the basic functionality that tools would have to make available to authors.

These functions may be used directly if the application is written as a ‘C’ program where the author would make calls to the Generic Browser library or indirectly because the author is using a tool which is itself based on the Generic Browser.

The advantages for the author of using the Generic Browser are firstly it is general. If an author develops an application as a one-off in, say, ‘C’ then the subsequent applications will either involve re-coding from scratch or, more likely, modifying the old code. The latter process is error prone particularly if the code was not written with re-use in mind. Using the Generic Browser means that both the data representation and access functions are standardised so that techniques developed using them can be exploited in subsequent work. The Generic Browser also promotes the production of clearer code because of its object oriented approach. This should in turn reduce maintenance problems. By addressing the area of information-rich multimedia applications, where there is little existing support, with a generic solution a gap is filled and wasteful bespoke applications can be avoided.

Although the authoring level allows Generic Browser functions to be used to realise functionality that is not provided by the Generic Browser directly the aim of the Generic Browser is to provide the basic building blocks so that if more complex functionality is required it is straightforward to produce it using the Generic Browser without resorting to clever tricks such as accessing the Generic Browser data structures directly. To put it another way the Generic Browser must provide the author with the functions needed to produce an application that offers the user the functionality that she wants. The extra requirements of the author revolve around clarity and uniformity of representation and

sufficient breadth of functionality to make addressing the user's needs straightforward. If the author is to use a tool based on the Generic Browser the same requirements apply from the tool builder's perspective.

6.11 Summary

Chapter 5 generated a set of requirements that are addressed by this architecture as follows:

- browse and search multimedia data

This requirement has meant that the representation must address the need to reference multimedia data stored outside the frame representation (stored on CD for example). This can be achieved by storing the filename (or offset within one large datafile) in a slot in the object frame.

It has also meant that a content representation must be able to be stored with each item (to enable searching of all media types). The technique of storing attributes to represent the content of a media object has been shown. The flexibility of the frame system allows authors to choose the granularity of their content representation by defining the relevant slots and ranges of allowed values.

- browse freely in the information space

The use of relations in the frame system allows lists, hierarchies or networks of items to be created. A mechanism to browse this structure (i.e. traverse these links) is then required.

- look at related items

The relational links of the frame system provide exactly what is required to link related items together.

- have information presented as an overview or in detail

This requirement can be addressed in two ways, firstly the data can be stored within the data frame in more than one version, the correct one being selected by the application. Alternatively the detailed view of the data can be treated as a related item linked to the overview by, say, a *more information* relation. Both of these are catered for with a frame system.

- mark items, possibly with a value

The data representation scheme must allow marks to be stored with the data items so that subsequent browsing is possible. The dynamic nature of frames allows

slots and values to be created at run-time. This is the necessary underpinning for the concept of marking.

- collect sets of items for further browsing

This is addressed by allowing sets of marked items to be browsed.

- leave navigational placeholders to allow an item to be revisited

Marks can be used for this purpose.

- retrace steps i.e. to backtrack

By ensuring that all navigation is performed using Generic Browser functions the necessary history information can be stored by these functions when they execute.

- follow guided tours

These can be constructed using the path construct.

- freely move between browsing and searching access

The use of marking of search hits enables the results of a search to be browsed.

- search for items by a variety of query mechanisms

The vector space search mechanism described above provides an alternative to the simple matching that is also required. The hit marking mechanism is applicable to any search mechanism that is required.

6.12 Functional Specification of the Generic Browser

The architecture described above has been used to generate a functional specification for the Generic Browser. In this case, the term generic is taken to mean independent of application, not biased towards a particular form of browsing/searching and independent of the data and its presentation.

The Generic Browser comprises two main functional units

the browse engine

the search engine

Both of these components operate on a common data representation.

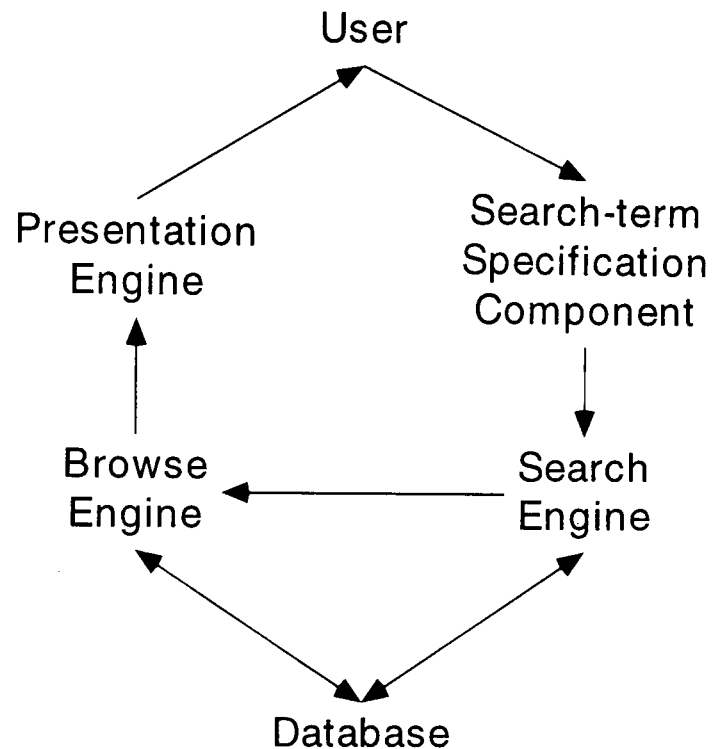


Figure 47. Components of the Generic Browser.

There are two components needed by the Generic Browser which are intimately tied to the application and which therefore cannot be provided in generic form. These are a presentation component and a search term specification component.

Generic Browser Functional Specification

The Generic Browser will provide:

- a data structuring environment using a frame representation scheme,
- hyperlinks between objects (possibly named) and a means to traverse them,
- a mechanism to enable items to be marked with the following features:
 - ◊ an arbitrary number of named marks on any item
 - ◊ values to be associated with marks
 - ◊ treatment of a set of similarly marked items as a single object
 - ◊ browsing of marked sets
- a path object which “contains” a sequence of data items and information specific to the path as a whole,
- a means to traverse paths,
- a means of recording a history of data items visited and a mechanism to backtrack through that history,
- a search mechanism that marks data items with specified characteristics,

- a search mechanism based on the vector space model with the following features:
 - ◊ operates on the data held in frames
 - ◊ represents the query as a frame
 - ◊ marks the “hit” frames with a similarity value
 - ◊ supports two mechanisms for negative queries,
- a search mechanism that applies a function to compare query and target characteristics and marks the resultant hits (function search),

This functional specification above has formed the basis for the implementation described in the next chapter.

7 Library Implementations

This chapter opens with descriptions of the two main phases of the development of the Generic Browser library. The first prototypes were implemented in a version of LISP but following unattractive performance evaluations on the target platform, CD-i, a move to a 'C'-based implementation was made. There then follows a more detailed examination of the implementation of the main features of the Generic Browser function library.²⁷

7.1 CD-i - The Target Platform

As has been stated in chapter 3 this work aims to be general and at the same time implementable. In particular it is targeted at consumer multimedia platforms. To demonstrate the practicality of the Generic Browser concept the implementation work covered by this thesis has been performed on the CD-i consumer multimedia platform. The specification for such a machine appears in appendix A.

Code for CD-i players is generally developed on a more powerful platform, in this case a Sun workstation. A 'C' cross-compiler on this machine produces OS-9 code which is linked with the relevant libraries to produce a module that will execute on a CD-i player.

7.2 Lisp Version

Having chosen frames as the basis for the Generic Browser work it was necessary to provide a frame-based system on the CD-i player. No frames-based system was already available so the decision was made to implement one especially for the Generic Browser.

The initial choice of implementation language was LISP. Previous Project prototyping work had been implemented in LISP on Sun workstations for convenience. At the time of choosing CD-i as the target platform another project was considering LISP as the basis for an extensible authoring language (similar to HyperTalk) and compatibility with that exercise was thought to be useful.

LISP is powerful and flexible and, because it is interpreted, is particularly suited to the rapid prototyping methodology used for this work. Preliminary work on the Generic Browser library was carried out on a Sun workstation which allowed faster prototyping. However the target platform was kept in mind and a suitable LISP interpreter for porting to CD-i sought.

²⁷All implementations of the Generic Browser Library and of XKC were done by the author.

7.2.1 XLISP

The public domain LISP interpreter called XLISP²⁸ implements a subset of COMMON LISP. It is written entirely in 'C', with source code provided. The key feature of this system, its small size, making it suitable for use on the CD-i player.

XLISP having been ported²⁹ to CD-i the interpreter was modified to enable it to be called as a 'C' function [Cole 1991a]. This was accomplished by modifying XLISP so that rather than, as is usually the case with LISP, the XLISP process being the main program, a 'C' program could call the XLISP evaluator as a subroutine. This was necessary because on a CD-i player an application is a set of 'C' functions that are called by a supervisor program that handles events such as mouse clicks etc.

7.2.2 XKC

With the XLISP interpreter ported to the CD-i player the next step was to write a frame-based system in XLISP. One problem with the choice of a frame-based representation is its potential size. This is because the frame approach was developed in the AI world where efficient use of computing resources came second to high level representations and powerful toolkits³⁰. Previous work in the Project³¹ had been done using KnowledgeCraft³² running on Sun workstations. This experience allowed a minimal set of functions to be specified containing the following:

- create/delete frames
- add/delete slots
- add/delete values
- create and manipulate relations

²⁸XLISP was written by D.M. Betz and placed in the public domain.

²⁹XLISP was ported to the CD-i player by David Eves at PRL.

³⁰It is interesting to note that over the last few years there has been a shift in the AI world towards applications of AI techniques that run on readily available platforms such as PCs. This has in turn given rise to more efficient implementations of AI tools running on such machines.

³¹The Intelligent CD-i Project was investigating how techniques of AI might be used in consumer multimedia applications such as a Cookery Advisor.

³²KnowledgeCraft is a comprehensive AI toolkit which has the disadvantage of requiring a powerful, virtual memory workstation to run on.

Inheritance over is-a/instance links was specified as was provision of “typeless” data³³. The principal omission from this list as far as “traditional” frame-based systems are concerned was demons. These allow functions to be called when the values in a slot with a demon are changed. Provision of demons has a major impact on the run-time efficiency of the system making it run slower. It was felt that their absence from XKC would not adversely affect the Generic Browser. This has been borne out in practice. All that demons provide is an *automatic* mechanism for triggering procedures when data is changed. If the Generic Browser needed to call a function when data was changed this could be done more efficiently by including a call to that function before or after the call to change the data.

A simple frame system offering the core functions above was written in XLISP [Cole 1991b]. This was based on the frame manipulation functions described in [Winston & Horn 1984 ch. 22] and was known as XKC. For reasons of compatibility with KnowledgeCraft, the functions were named to correspond with the KnowledgeCraft conventions, in particular frames were referred to as *schemas*.

7.2.3 Generic Browser Prototype - Basic Functionality

The LISP version of the Generic Browser was written using XKC and XLISP running on the Sun. This implementation was regarded primarily as a frame *browser* (search had not been included at this stage). It was envisaged that the user would browse items connected by “previous” and “next” links. As will be seen later, when the prototype applications are described, the use of just these links was not found to be sufficiently flexible and so the concept of paths was explored as the primary browsable structure.

Though lacking a number of features later considered essential (not least the integration of search) this first version of the Generic Browser proved to be an invaluable test-bed for the ideas which form the concept of the Generic Browser. The function set implements the key functions of the Film Browser described in chapter 2. The following functions were implemented in the LISP version:

³³All data items be they text, Boolean, integer, floating point or the names of frames are treated the same, without the need for declaring their type.

Browsing

- next/previous
- next/previous marked
- next/previous tagged
- traverse a link
- backtrack

Marking & Subsets

- add/delete a mark
- add/delete a tag

Multimedia Data

The frame representation used was found to be ideally suited to the representation of multimedia data

7.2.4 Marks and Tags

The LISP implementation distinguished between marks with and without values (known respectively as tags and marks). It became clear in project discussions that this was not a justifiable distinction. The separation of marks and tags was found to be an implementational convenience rather than a conceptual difference. Subsequent versions of the Library had marking with optional values which was found to be much more intuitive.

In the LISP version marks were represented by connecting all marked frames by a relation³⁴ with the name of the mark. To allow access to marked frames the frame *marked-items* was linked to the start of each chain of items connected by relations with the name of the mark as shown below.

³⁴At a conceptual level information nodes are linked together to form a browsable structure. At an implementation level these links are implemented using relations.

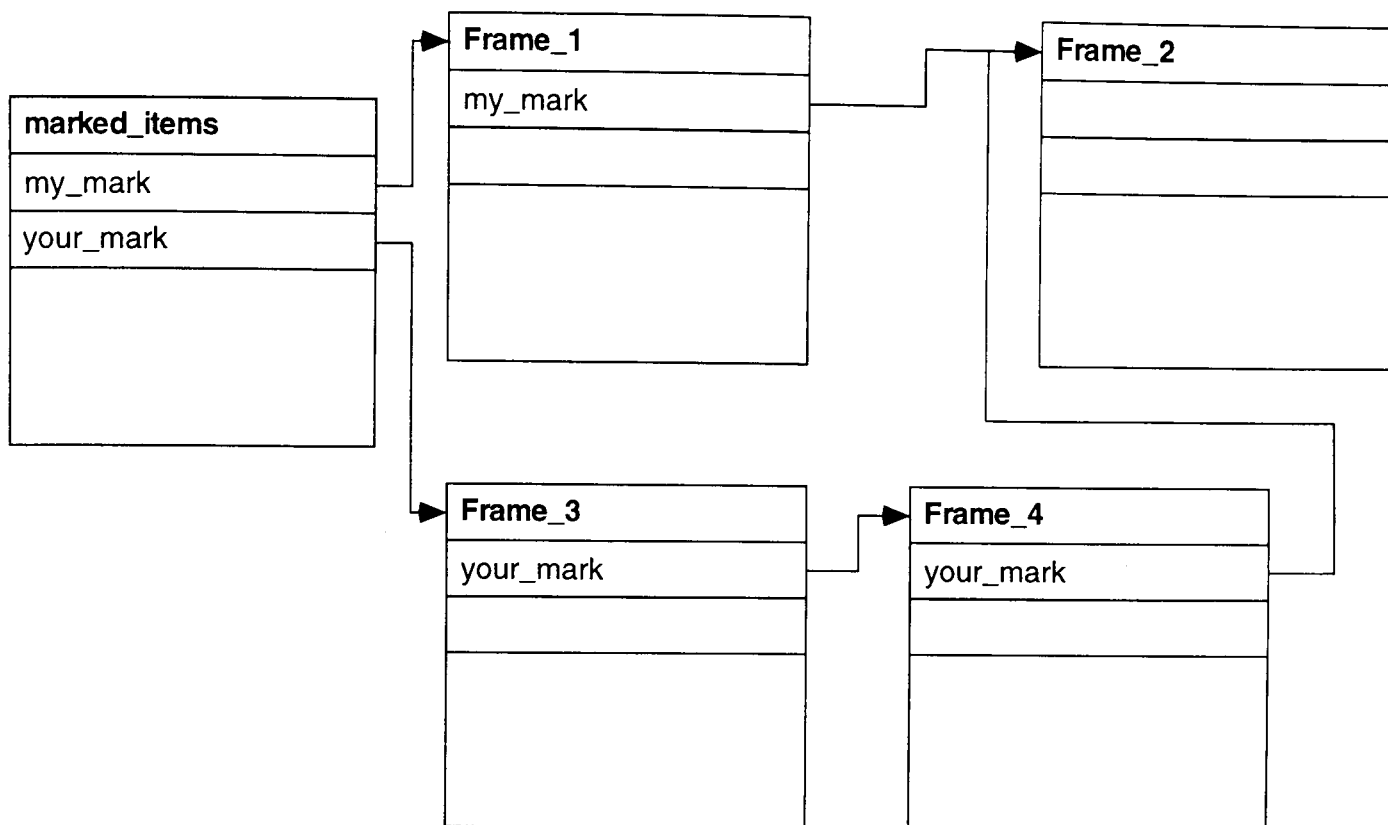


Figure 48. Marked items data structures (LISP version).

In the example above *Frame_1* and *Frame_2* are marked with *my_mark*. *Frame_2*, *Frame_3* and *Frame_4* are marked with *your_mark*. *my_mark* and *your_mark* are relations.

The disadvantage of this structure is that *marked_items* forms part of the linked list so that the *next/previous_marked* functions must avoid it as the mark list is traversed by following the relations. Also the members of a given marked set can only be found using *get_relatives* on *marked_items*. The *get_relatives* function is provided XKC (and other frame-based systems). It returns a list of all frames linked to a specified one by a given relation. So to find all the items marked with *your_mark* the following would be used:

```
get_relatives("marked_items",
             get_inverse("your_mark"));
```

It would be more efficient to have a single object that contained a list of all members. A path is just such an object and subsequent versions of the Generic Browser utilise paths to hold marked sets. Here the path is constructed by the system at run-time. This contrasts with the use of a path by an author to create a browsable sequence.

7.2.5 Node structure and Use of Relations

The user defined relations which are part of XKC (and Cframes³⁵) provide a mechanism to support a form of hyperlinks. User defined relations automatically maintain the inverse relation; this provides bi-directional links at no cost to the author. Though the standard XKC functions are used to create and delete links, link traversal is handled by the Generic Browser. This allows the changes to the current frame to be recorded in the history. There is a strong similarity between the links in hypermedia and these relations.

The primary distinction that can be drawn between hyperlinks as found in, say, HyperCard, and the relational links in a frame system revolves around the anchors³⁶ of the link. In a frame system the relations link frames together whereas in HyperCard the link goes from a button on a card to another card. In the first case the anchor can only be placed on the data node whereas hypermedia systems generally allow the anchor to be within a node such as on a button or “hot” word.

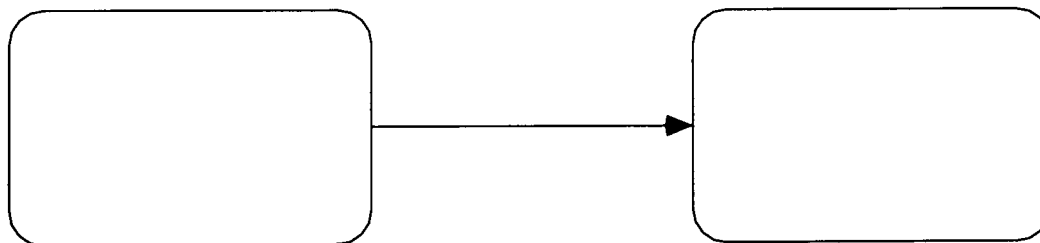


Figure 49. Screen linked to screen.

³⁵Cframes was written and implemented by M.A. Shiels [Shiels 1991a].

³⁶In hypermedia terminology an anchor is the source of a link in hypertext/hypermedia systems.

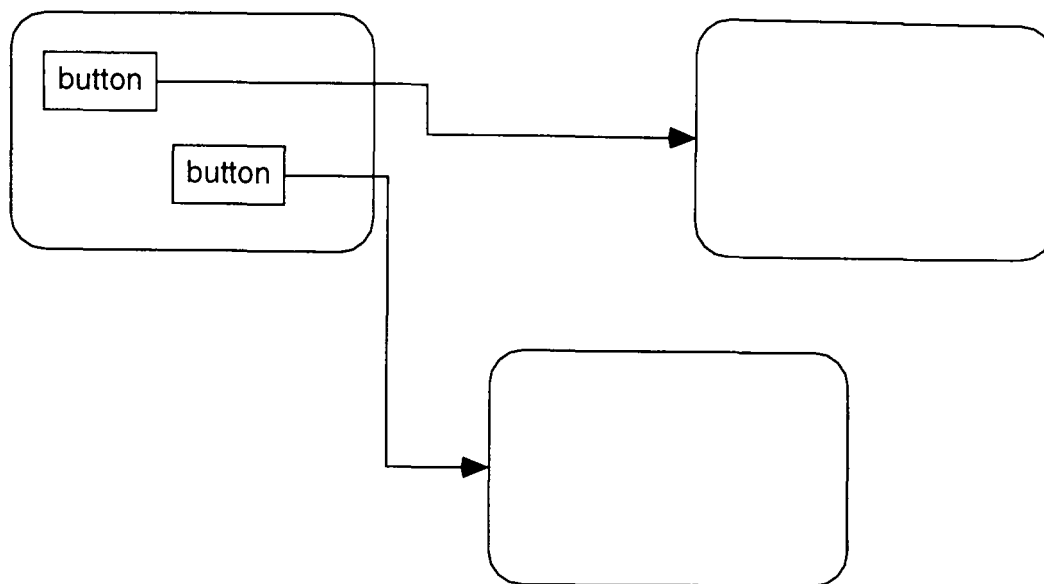


Figure 50. Hotspots linked to screens.

Frame to frame linking means that “true” hypertext cannot be implemented *just* using the relations provided in the Generic Browser. This is because there is no way to associate the individual words in the text (which would either be held as a value or values in a slot or, more likely, in a separate asset) and the links to other frames.

This shortcoming of the use of relations as hyperlinks can be addressed as follows: If the links are not sensitive to the context and every occurrence of a specific word always causes a specific link to be followed (a dictionary definition or glossary entry for example), links for each word can be made and placed in a look-up table. These can then be used to translate between the word selected, as a piece of text, and the link to be followed. For straightforward applications the word and the name of the link can be the same. This lookup table approach could be extended to cope with words used in context by treating each “hot” area in the document as a separate entry in the table. This table would then supply the required link name. This may seem unnecessarily complex but a real application will already have some form of mapping between the pointer location and the hot word in the text so this could be extended to return the link to be followed.

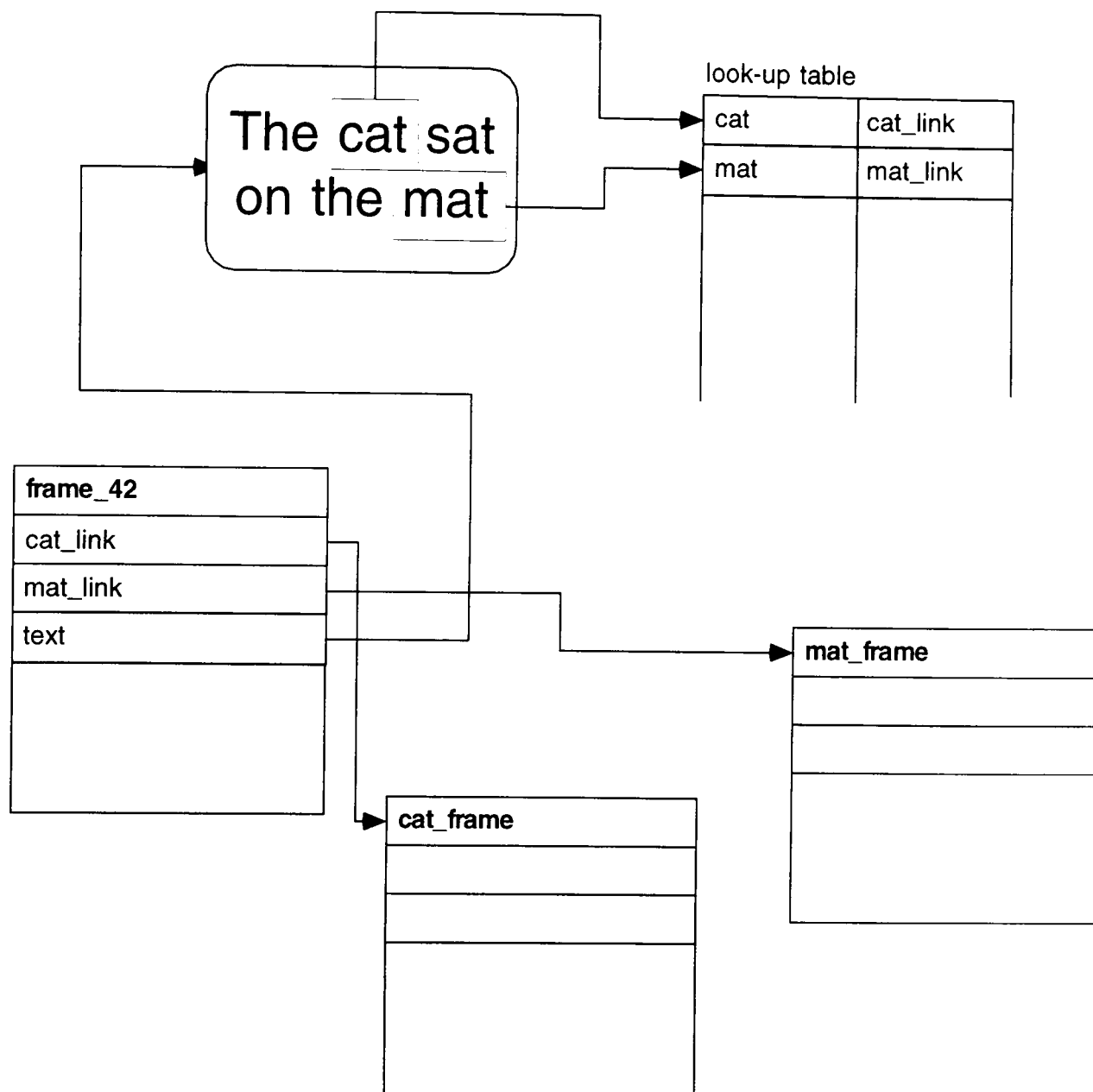


Figure 51. Look-up table links hotspots to frames.

An alternative of simply recording the name of the target frame in the lookup table could be used. However if links are used then inspection of the frames will show which frames may be accessed from it and, more importantly, using the *traverse_link* function allows the history mechanism already provided for links to be kept up-to-date. If links are not used, the application will have to take care of the history information itself.

7.2.6 Refinement of the LISP Version

The first LISP prototype was refined by modifying the implementation of the library and testing the result in a simple browsing application. This prototyping activity continued throughout the development of the library and though the actual prototype applications were small scale their utility has been that they forced an application developer's viewpoint on the evaluation of the library as it evolved. It was during this prototyping phase that the interactive nature of the LISP programming environment was most useful.

As an example of the LISP version of the Generic Browser the following code fragment creates three frames, *frame-1*, *frame-2* and *frame-3* and links them together with *next* relations (the inverse relations, *previous*, are created automatically). The current frame is then set to *frame-1* and the subsequent frames browsed by traversing the *next* relations.

```
;;; first create the frames

(create-relation 'next :inverse 'previous)
(defschema frame-1 (next frame-2))
(defschema frame-2 (next frame-3))
(defschema frame-3 (next frame-1))

;;; now browse

(goto-schema 'frame-1)
(next)
;;; frame 2
(next)
;;; frame 3
(next)
;;; frame 1 again
```

7.3 'C' Implementation

The biggest disadvantage of using an interpreted language such as LISP and XLISP in particular was that, running on a CD-i player, its runtime performance was poor [Shiels 1991a]. So, whilst it was a useful “proof of concept” exercise it was clear that a more efficient, higher performance implementation was required if it were to be possible to build realistic applications using the Generic Browser.

This section will describe the evolution of the Generic Browser implementation as the changes and refinements suggested by the prototype were incorporated and themselves tested in subsequent prototypes.

7.3.1 Object Orientation

Given the obviously object oriented style of a frame-based system the question naturally arises as to why the Generic Browser was not written in an object oriented programming language such as C++. The principal problem is again performance. A C++ compiler produces larger, slower code than a pure C implementation. In view of this and given the inherent modest performance of the CD-i player, C was chosen instead. However an

object oriented style has been maintained as far as possible in the Generic Browser function set. For example parameters are modified using function calls rather than modifying global data directly and all information pertaining to an object such as a path is stored in the object (in this case the path frame). To achieve the required response time, it was decided to use the Clisp library [Drasch 1989] to provide the underlying LISP-like functionality in C. On top of this, a frame-based system (Cframes) similar to XKC was built by another project member [Shiels 1991b]. To complement Cframes the Generic Browser work was recoded from LISP into Cframes and C.

The Cframes version of the Generic Browser included some new functions:

- create, add and delete from a path
- find (and mark) frames with slots containing specified values
- perform a vector search, marking the result

These are discussed in more detail below.

7.3.2 Implementation Schemes for Simple Paths

Two possible implementation schemes for paths are considered below. In the first the path is represented as a list in a dedicated path frame. In the second, the path structure is formed by linking instantiations of the relevant nodes together.

Paths as Lists

Since a simple path is just an ordered list of data items, one possible implementation is to use a list to hold the contents of the path. This has the advantage that the path object can now exist in its own right (as a frame, with the path list itself as values in a slot).

The first problem with this implementation is that there is no link in the frame-base between the data items and the paths on which they appear. Thus it is not possible to inspect an item frame to see which paths that item is on. Obviously this can be solved by searching all the paths but a better solution is to maintain this information in the data frames as well (with an “on-path” relation for example).

This highlights a situation encountered several times during the design of the Generic Browser, that it is necessary to decide whether to recompute a piece of information every time it is needed or whether to cache it in the data frames. This extra information would be updated each time the frame access and manipulation functions are called. Another example is that of marking (which will be described more fully shortly). The mark is a piece of information attached to a data item. Consequently it is stored in the data frame. But it is frequently desirable to access all marked items (to browse the marked set for example). As

a result, a list of all the items marked with a particular mark is stored in a special marks frame. Thus to access the marked set, rather than having to search the entire frame-base, one slot access is all that is required. The overhead of providing this ease of access is that all the functions that modify marks must maintain the integrity of the information in the marks frame as well as that in the data item.

An important aspect of storing a path as a list is that there is a need to be able to traverse the path. So, the position within the list must be stored (the *current frame on the path*).

Although the list is stored in order, there is no concept of an index to a list in Cframes, nor in XKC for that matter. In 'C', this would be a trivial problem as an array could be used. Two potential solutions are considered.

Firstly the path list can be split at the current frame, storing the two parts in two separate lists. Traversing the list simply requires list surgery to move items from one list to the other; the head of one of the lists being the current frame.

For example, if *kettle*, *toaster*, *iron*, *coffee_maker* and *food_processor* are items on a path *domestic_products* and *iron* is the *current_frame_on_path* then the path frame will look like this:

```
(domestic_product
  (instance path)
  (forward iron coffee_maker food_processor)
  (back toaster kettle))
```

Going forward along the path moves *iron* from the *forward* slot to the *back* slot. The *current_frame_on_path* is now *coffee_maker*, the head of the *forward* slot value-list as shown below.

```
(domestic_product
  (instance path)
  (forward coffee_maker food_processor)
  (back iron toaster kettle))
```

The second solution is to store an index to the current frame. This requires an efficient function to extract the *n*th item from a list.

Taking the same example:

```
(domestic_product
  (instance path)
  (path_list kettle toaster iron coffee_maker
              food_processor)
  (path_ptr 3))
```

Traversing the list is now simply a matter of incrementing (or decrementing) the index to the current frame as shown below.

```
(domestic_product
  (instance path)
  (path_list kettle toaster iron coffee_maker
              food_processor)
  (path_ptr 4))
```

The advantage of the first scheme is that handling end conditions whilst traversing the list is particularly straightforward since at either end of the path one of the two sublists is empty. Its disadvantage is that the list surgery is not very efficient. The second method is almost the exact opposite as traversing the path is efficient whilst the end conditions have a larger overhead. Also, handling additions and deletions from the path is more costly than in the first method.

One notable advantage of the second approach of using an index to the current frame is that the actual path frame can be created (and modified) using a frame editor as the path is simply stored as a list. The split list method is not quite as intuitive for the frame editor user.

Normally these implementation details are hidden from the user of any Generic Browser functions, indeed, for test purposes, in the Cframes implementation, a compiler switch can produce Generic Browser libraries with either method for storing paths.

Paths as Link Structures

An alternative to the list structures described above is to link path items together. Simply linking data nodes together with a path relation, whilst conceptually simple is implementationally flawed. Specifically, consider the case where a particular data item needs to appear twice in the path (a slide show for example). If an attempt were made to construct such a path using, say, “next” and “previous” relations a problem is found when the data item must be linked to its neighbours in the path for the second time as relations can only have one destination.

A possible way around this problem is to use instances of the data nodes. These would inherit all the data from the actual data item, but each item could have as many unique instances as were required to build the path. The instances being linked by the path relations. This approach is shown in the diagram below.

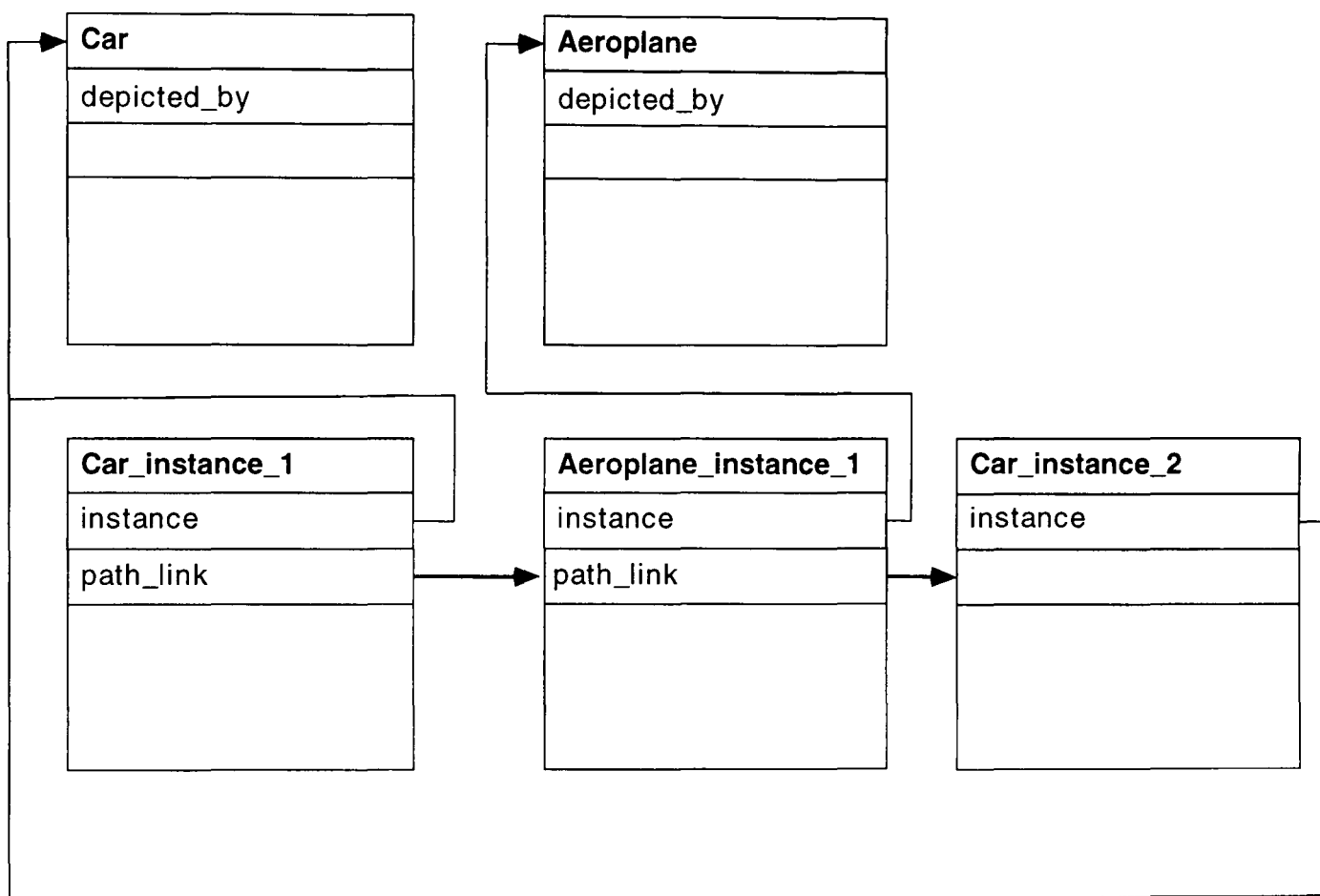


Figure 52. Use of instances and links to form a path structure.

The disadvantages of this method are the extra storage for the instances (although the use of inheritance minimises the data actually stored in the instance nodes) and the problem of naming the instances. It was felt that this solution did not confer advantages in sufficient measure to compensate for these deficiencies and was not taken further. In the final version of the Generic Browser paths are stored as lists with a path pointer providing the current frame on the path. This was chosen because it was seen as very desirable that the path be stored as a single list.

7.3.3 Marking

The concept of marking has always been important in the development of the library. Named marks allow more than one marked set to exist in the database at any one time. This is essential if marks are to form the basis of subdividing the database and of denoting the results of searches. This is in contrast to HyperCard which though it has the facility to mark a card, only one mark is allowed per card and only one set of marked items per stack.

Also part of the design of marking in the Generic Browser is the ability to add a value to the mark.

There are two aspects to the implementation of marks in the Generic Browser. Firstly the mark and its value must be stored in the item frame. Secondly there is the requirement to browse marked sets. The former is easily accomplished by adding a *marks* slot to the item frame. This has mark-value lists as its values. For example suppose the *Ferrari_F40* frame was to be marked with a *desirability_factor* (on a scale of one to ten) then the result would look as follows:

```
(Ferrari_F40
  (is-a supercar)
  (marks (desirability_factor 10))
```

and similarly

```
(Ford_Escort
  (is-a basic_car)
  (marks (desirability_factor 3))
```

and, to demonstrate multiple marks

```
(Bentley_Turbo_R
  (is-a luxury_car)
  (marks (desirability_factor 8)
         (affordability 2))
```

The need for marked sets to be browsable suggests that there should be an object to represent the marked set. In the Generic Browser the natural choice is to use a path. When the mark manipulation functions of the Generic Browser are invoked they perform two basic actions: one updates the *marks* slot in the item frame, the other updates the path associated with that particular mark.

For example, when the *Ferrari_F40* frame is modified as shown above the *mark* function is called. This puts the (*desirability_factor 10*) value in the *marks* slot and adds *Ferrari_F40* to the *desirability_factor* path (if it is not already there). In the Generic Browser subsets created by marking and subsets created by an author and expressed as paths use the same underlying representation.

7.3.4 History Mechanisms

In the previous chapter four types of historical information were examined; navigational history (backtracking), statistical information, temporal information and action history

(undo). In the Generic Browser prototypes only backtracking has been implemented. Of the other three, an undo facility that allowed the effects of any function to be reversed is not particularly useful in multimedia applications. Unless only a single step undo were provided there would be a large overhead of storing all the necessary information for each function call to allow it to be undone. This type of history mechanism is of most use in editor-type applications not information browsers. In a browser, backtracking is really a navigational tool rather than a mechanism for undoing since all that has been *done* is to view information. It is not possible to undo the fact that an item has been browsed even if one can remove a tag from it.

Statistical and temporal information would be easy to add to the data nodes each time one was visited using Generic Browser functions however none of the prototype applications needed to make use of such information and the final version of the Generic Browser does not provide it³⁷. Since all navigation in the Generic Browser ultimately calls a single function, *goto_frame*, this function could also update the statistical information as each node is visited. Temporal information can be recorded in the same manner. Statistical and temporal information would be stored in nodes as marks with values. When an item is visited it gets marked, subsequent visits update the mark value - thus it is easy to process visited nodes by searching with a threshold filter to get most popular for example. If an application required these types of history information then they could easily be added to the Generic Browser.

History as a Path

For history information that is not stored in the nodes such as navigational information (which nodes have been visited and in what sequence), some kind of ordered list is required, accessed as a stack. A stack is a last in, first out (LIFO) data structure where the most recent item placed on the stack is the first to be removed. As history is “created” the information, such as a new location, is placed on the stack. The process of backtracking simply becomes popping items off the stack. In the case just mentioned, popping an item from a navigation history returns the last node visited. A “go back” function simply goes to that node (without pushing anything onto the history stack).

In the Generic Browser the history is stored as a path rather than in a special data structure (such as a history stack) for reasons of uniformity of representation. Navigation along that path is a special case because only backwards travel through the history is allowed. (Though in principle there is no reason why a user shouldn't go back and forwards through

³⁷This decision was also influenced by the efficiency constraints of CD-i where enlarging the data nodes with history information was not seen as a good idea unless actually required by an individual application.

the history it is difficult to see how this would be presented to the user in such a way as not to confuse him.) As the user goes backwards the historical nodes are removed from the history path as though it were a stack. Of course, unlike the normal path functions, *go_back* does not update the history. For example if the user visits frames *f2 f5 f7 f6 f4* then the history would be as shown below:

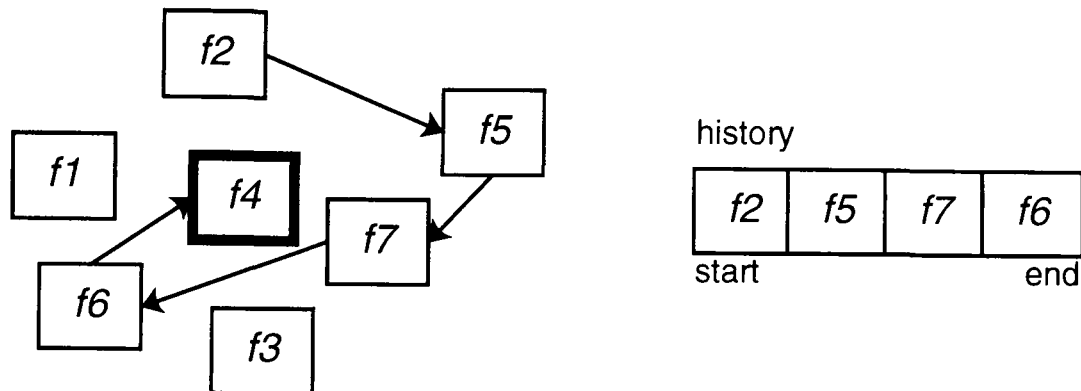


Figure 53. The History path.

If the user then backtracks one step, the current frame becomes *f6* and the history is as shown below:

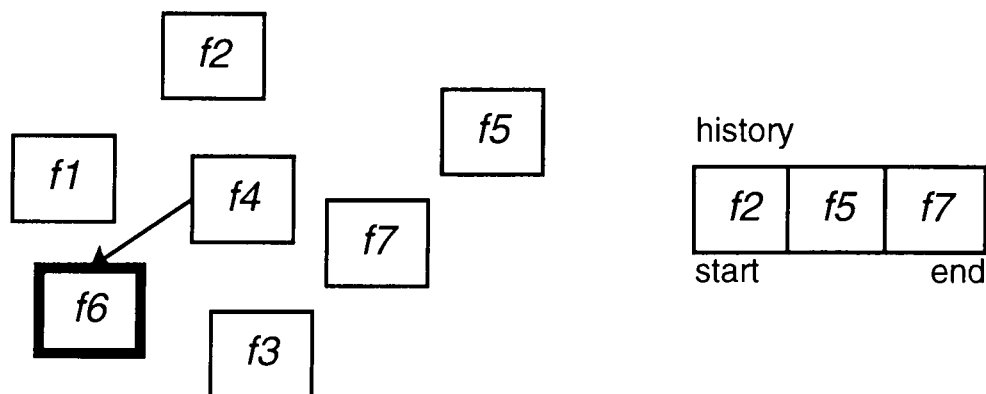


Figure 54. Backtracking along the History path.

This use of a special path for recording a history of the nodes visited was the first stab at providing a history mechanism in the Generic Browser. As the Generic Browser was used in prototype applications it soon became apparent that this type of history was too inflexible in that all it recorded was the nodes.

A “go back” function implemented using it would force the user to retrace his steps one at a time, particularly unnecessary in a browser forming part of a larger application. In this case, the author would like to be able to specify the destination of a “go back” function on a screen by screen basis.

A solution to this problem which does not require additional functionality in the Generic Browser is as follows: when the application program receives a trigger to change screen the destination of the “go back” function for this new screen is added to a history path (providing that the “top” item in the path is currently different - this avoids repeated nodes on the top of the stack); when the user wishes to backtrack the history path is simply browsed using the *go_back* function.

Alternatively the go back functionality might have been implemented using a backtrack relation which, for each frame, would point to the frame to backtrack to. This was rejected because it did not take into account how the user got to the current frame. This contextual information may well affect the go back destination.

7.3.5 Search

A number of different search mechanisms provide the search facilities in the Generic Browser. The “basic” search functions find frames with particular value(s) in specified slot(s). Care has been taken within the design of the Generic Browser so that new or revised functionality can be added at a later date. In particular, using marks to indicate the results of a search means that alternative search mechanisms can simply be added, marking their “hits” in the same way as the vector search.

The final implementation of the vector search mechanism works directly on the information stored in frames (rather than vectors held as, say, arrays). This is a trade-off between efficiency and the uniformity of representation which makes life easier for authors of applications using the Generic Browser. In particular the searchable information is in the data - the frames describing the objects to be searched and not embedded in the program code.

To use the vector search facility in the Generic Browser a query frame is constructed. This will contain those slots which the user wants searched in the frame base. The values to be matched are placed in these slots in the query frame. The class of frames to be processed and a list of slots to be searched are passed to the vector search function. The vector search routine then matches the values stored in the query frame slots with those in the corresponding data frames. The procedure stores the similarity value as a mark in the data frame and keeps an ordered list of all the frames with a non-zero similarity measure in a *search_result* slot in the query. For simplicity and to optimise execution speed the similarity measure used is the inner product measure described in the previous chapter. This similarity value is simply the number of characteristics that are present both in the query and in the item under test.

Weights and Normalisation of the Search Results

Although both weights and normalisation of similarity measure have been prototyped the efficiency of the search functions deteriorates as the complexity of the calculations for each term in every item increases and so these versions of the search functions are not currently included in the library used for CD-i applications.

It is possible to simulate weighting by repeating characteristics either in an item (to indicate that the characteristic in question is more strongly present in that item) or in the query (indicating the characteristic is more important than the others in the query).

The similarity value is also not normalised because it is usual for the order to be more significant than the absolute value the values returned allowing relative ranking. If normalisation is important to an application it is a simple matter to divide the similarity measures by the number of search characteristics.

Negative Queries

One of the requirements of the Generic Browser's search facilities is to be able to handle queries of the form "find all frames *not* containing ...". As [Raghavan & Wong 1986] observe there is no reason why the elements of the search query should not be negative. Using a negative term in the query vector will reduce the similarity measure when it is matched in an item thus demoting that item in the ranked list of matching items.

The drawback of such negative query terms is that items that match the negative query term are not excluded from the search result, merely demoted in the order. Two solutions to this problem have been considered:-

Modification of the vector search mechanism. This can easily be modified to exclude terms matching negative queries. However this may cause problems if weights are to be used when it may be desirable to allow negative query terms rather than simply excluding them.

Provision of a separate function for excluding items from the search result. The database could be searched with the exclusion function to eliminate items matching a pattern. Subsequent searches could then be done on the resulting marked set. Negative query terms can now be used to reduce the rank of items which they match.

The search process

To do a vector search, three stages are required:

- set up the query frame
- do the search
- process the result

For example given a frame base as follows:

```
(f1
  (is-a x)
  (s1 a c d g j)
  (s2 p q r))

(f2
  (is-a x)
  (s1 a b c d g j))

(f3
  (is-a x)
  (s1 d e f))
```

To set up a search query a new frame is made with the slot to be searched, *s1*. In the query frame, *query1*, the slot *s1* contains the characteristics to be matched, namely, *a, b, c, g, j*. Note that if other slots were to be included in the search, those slots and the search values would also be included in the query frame.

```
define_frame ("(query1 (s1 a b c g j))");
```

now calling the search function

```
vector_search_class("x", "s1", "query1",
                   "search_tag1");
```

will result in

```
(f1
  (is-a x)
  (s1 a c d g j)
  (s2 p q r)
  (marks (search_tag1 4)))

(f2
  (is-a x)
  (s1 a b c d g j)
  (marks (search_tag1 5)))

(f3
  (is-a x)
  (s1 d e f))
```

```
(query1
  (s1 a b c g j)
  (search_result f2 f1))
```

The result can now be processed. Note that the actual value for each search hit is the number of characteristics present in both that frame and the query. The slot search result is ordered so that the best match is at the start of the list.

If slot *s2* were to be included in the search as well, the query would be defined as

```
define_frame("(query2 (s1 a b c g j) (s2 p r))");
```

and the call to vector search would be

```
vector_search_class("x", "(s1 s2)", "query2",
  "search_tag2");
```

with the result

```
(f1
  (is-a x)
  (s1 a c d g j)
  (s2 p q r)
  (marks (search_tag2 6)))

(f2
  (is-a x)
  (s1 a b c d g j)
  (marks (search_tag2 5)))

(f3
  (is-a x)
  (s1 d e f))

(query1
  (s1 a b c g j)
  (search_result f1 f2))
```

A further, more concrete example is given at the end of this chapter.

Performance Issues

Analysis of the performance of the Generic Browser vector search routines showed that their performance was somewhat slow. This was particularly true for larger numbers of nodes as the performance decreased non-linearly.

The vector search routines perform three basic steps:

- 1) The relevant subset of the frame-base is extracted as a list of frame names (except when the whole frame-base is to be searched when the underlying Cframes data structures are accessed directly).
- 2) For each frame in this list each slot in the query is compared with its counterpart in the data frame. A cumulative count of matches is kept for each frame. Once the query slots have been compared with those in a given frame the accumulated match is added to that frame as a mark and the frame name appended to the result list.
- 3) Once all the frames have been searched the list of hits is placed in the query frame's *search_result* slot.

It is the second step which is most time-critical. Two actions can be taken to improve matters. Firstly maximally efficient code can be used. In particular, the LISP-like conventions used with Cframes where functions are repeatedly called, should be replaced with single calls and local variables which is more efficient in 'C'.

Secondly the marking process can be split into two phases, the marking of the frame and the adding of the frame name to the mark's path. The name of the path frame is added to this path at the *current_frame_on_path* position, an aspect of paths not important for marking. This process is inefficient and this becomes significant in the vector search functions because the mark function is called for each hit. The solution used in the Generic Browser has the marks added to the frames as they are "hit" in the search loop. At the end of this loop the search result list is added to the end of the mark's path. (It must go at the end to avoid invalidating the *current_frame_on_path*)

As part of the work in the PRL project the Cframes library itself was modified to include vector search - but without marking, simply returning two parallel lists - the hit frames and their similarity measures.³⁸ This Cframes vector search was much more efficient (scaling linearly with number of frames) than the original Generic Browser version it was based on as it operated on the Cframes internal data structures avoiding Cframes' character-based list

³⁸These modifications to Cframes were done by its author M. Shiels.

processing used at the Generic Browser level. The Generic Browser was modified to use this Cframes vector search function within its vector search functions. Also a *mark_all_frames_in_list* function was implemented to allow the results of the Cframes vector search to be marked.

7.3.6 Mechanisms for Branching Paths

The ideas for branching paths described in the previous chapter have received only limited exploratory implementation and have not been used in the prototype applications.

7.3.7 Debugging and Error Handling

Two versions of the Generic Browser library have been implemented: a debug version with error checking and a non-debug version optimised for speed. In the non-debug version of the library there is no error checking. This maximises efficiency at the expense of robustness. Although the application can be tested linked to the debug version of the library, unless that testing is very rigorous it is possible that an error may escape detection. In this case, the program will crash when the error condition is encountered. This may seem rather drastic but it is better that the application programmer precedes calls to the Generic Browser library with protective tests on the arguments. This need only be done in those parts of the code where problems may occur (and also allows remedial action to be taken by the program - a facility not possible in the more general error handling provided by the library). This provision of an unprotected non-debug version and a debug version with error handling is only necessary because of the performance constraints of CD-i. On more powerful platforms the error handling could be left in the non-debug version.

For compatibility, the Generic Browser uses the error handler in the Cframes library³⁹. This error handler prints out a suitable error message together with the function call with the offending argument highlighted. This type of error detection requires that each argument to a given function is tested for validity. To do this a set of test functions covering all possible types of invalid argument for Generic Browser functions was produced.

For example *new_path* takes two arguments, the first is the path and the second is the new list of frames to become the path. The first argument must be tested to ensure it is not null, is a valid frame name (i.e. is not a list), that a path of that name exists or, if not that a frame of that name does not exist so that a new path can be created. The second argument must be a list (or null).

³⁹developed by J. Goodlet at Sussex University

The debug version of the library is compiled with the relevant compiler switches set to produce code suitable for use with the system's symbolic debugger. The penalty for this is reduced execution efficiency and increased code size. Consequently the non-debug version is compiled with the switches set for execution efficiency and so there is no information for the symbolic debugger.

7.3.8 Test Suite

A vital part of any software development procedure is that of testing. A test suite has been written which tested the individual functions from three perspectives:

Firstly each function is tested to ensure that it performs to specification. For example the *next_on_path* function is provided with a path and the current frame is then checked before and after the function is called to prove that it has operated correctly.

The second test checks that the functions behave correctly when erroneous values are passed to them. For example if the *next_on_path* function is not passed a path as its parameter a suitable error condition should be invoked depending on what has been passed (null, a non-path frame etc.).

The final aspect of the testing checks boundary values, that is those pathological cases which may catch out the code. For example what happens when *next_on_path* is called when the end of the path has been reached?

In addition to these tests some simple performance measures have been included which compare the time taken for particular functions to execute compared with a simple Cframes function such as *get_values*. As the prototype library has been refined repeated running of the test suite has preserved the basic functional integrity.

7.4 A Worked Example

As an example of how the Generic Browser is used, this section will take the reader through the creation of a simple application. This example is based on an electronic version of a skiing brochure, whose function is to help the user choose a skiing holiday.

Listed below is an unstructured collection of characteristics of skiing holidays. This can form the basis for choosing what frames should form the basic class structure, how the instances of these classes should be related together and what their content should be.

- which country e.g. France, Austria, Switzerland
- what level of skiing expertise e.g. beginner, expert
- type of accommodation e.g. hotel, chalet
- ski runs available e.g. cross country, black runs, moguls
- cost
- which resort e.g. Obergurgl, St. Anton

The user needs to be able to access the information in a variety of ways. He will probably want to browse the resorts in a particular country or browse the accommodation at a resort. He may want to specify certain characteristics of, say, the resort and only browse those resorts that come close to his specification. On the way he will want to mark items of interest for later review.

Having accumulated a selection of things associated with skiing holidays they must be sorted out into frames, relations and slots. The basic objects, which represent the conceptual objects in the domain, will be represented as frames. These are:

country
accommodation
resort

These objects can be placed in a class hierarchy. The root node “entity” is simply a placeholder to enable a single hierarchy to be constructed, a convenience when using graphical editors.

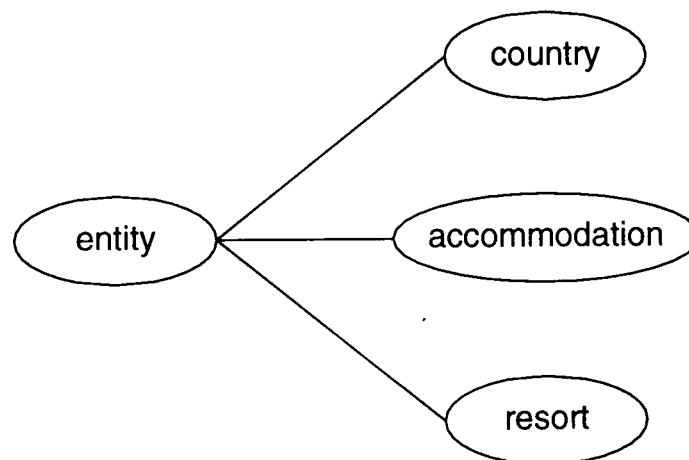


Figure 55. The basic class hierarchy

Some of these higher level objects will themselves split into more detailed objects. For instance accommodation can be divided into chalets, apartments and hotels, as shown below:

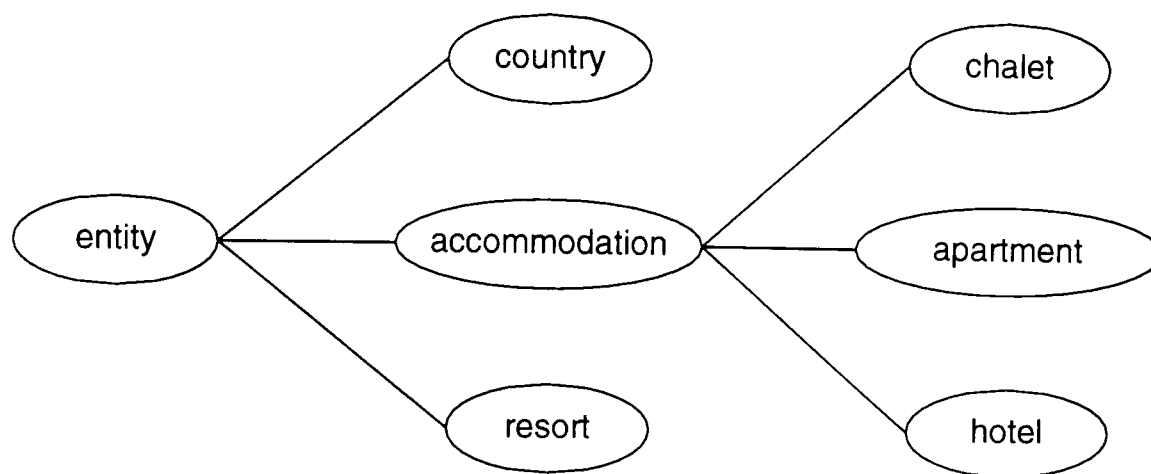


Figure 56. Types of accommodation.

As an example of the frame definitions, the accommodation class frame would have the following structure:

```

(accommodation
  (is-a entity)
  (is-a+inv hotel apartment chalet))
  
```

Now that the basic class structure has been defined, the next step is to choose how to represent the characteristics of these objects.

Users often select resorts by which country they are in. This link between resorts and countries must be represented in the frame-base. There are two possibilities. Firstly there could be a country slot in each resort frame or, secondly, the resorts can be related to their country frame using a user-defined relation. In the former case a search would have to be instigated to find the resorts located in a particular country. The latter case maintains this information automatically (this imposes a small memory overhead but is faster than using a search). For most applications the relational approach is preferred.

In the case of using just a country slot the resort frame would look like this:

```

(Obergurgl
  (is-a resort)
  (country Austria))
  
```

Now to find all Austrian resorts the following Generic Browser call would be needed:

```

find_all_frames_in_class("resort", "country",
  "Austria");
  
```

If the relation approach is used the following frames are needed:

```
(country
  (is-a+inv Austria))

(Austria
  (is-a country)
  (inverse_of_place Obergurgl))

(Obergurgl
  (is-a resort)
  (place Austria))
```

Now all that is needed to find Austrian resorts is:

```
get_relatives("Austria", "place");
```

The following could be used as a (less clear) alternative:

```
get_values("Austria", "inverse_of_place");
```

Apart from simply browsing the resorts in a particular country, the user may want to specify some features which make particular resorts attractive to him. The application can then promote these resorts to the front of the series to be browsed. To do this each resort frame must contain some representation of the characteristics of that resort. For the purposes of this example five resorts will be used. The table below summarises the characteristics pertaining to each of the resorts

	beginners	pretty_traditional_settings	good_cross_country_trails	snowboarding_&_monoskiing	traffic_free	snow_cannons	ski_guides	short_transfer_times	non_skiers	good_nightlife	glacier_skiing
Lech	●	●	●			●					
Obergurgl		●	●			●					
St. Anton				●		●	●		●	●	
Verbier		●		●		●		●		●	●
Zermatt		●			●	●			●	●	●

Table 4. Resort Characteristics.

Each of these characteristics will belong to the class characteristic as shown below:

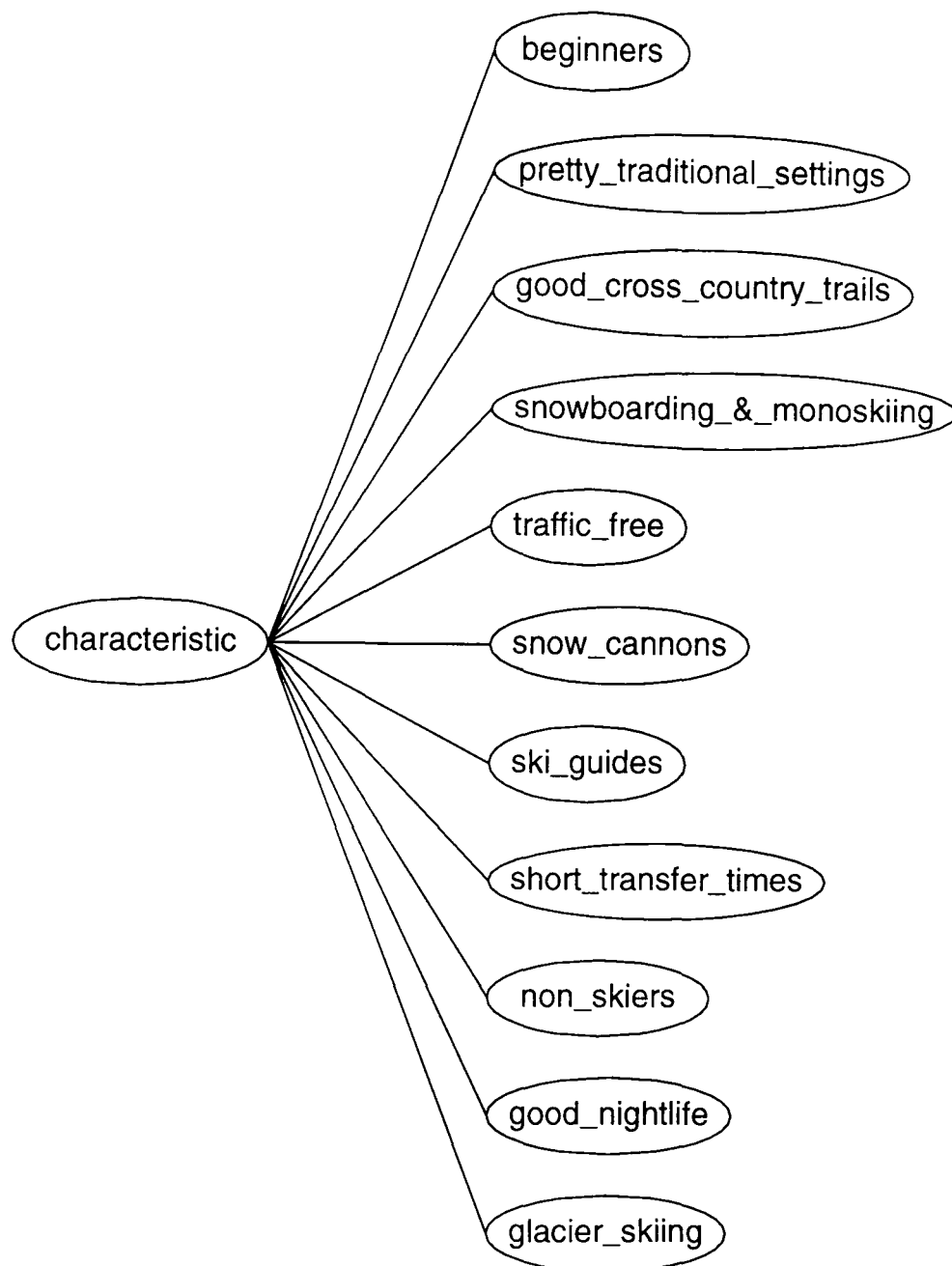


Figure 57. The characteristic class.

The characteristic frame looks like this:

```

(characteristic
  (is-a+inv beginners
    pretty_traditional_settings
    good_cross_country_trails
    snowboarding_&_monoskiing
    traffic_free
    snow_cannons
    ski_guides
    short_transfer_times
    non_skiers
    good_nightlife
    glacier_skiing))
  
```

Each resort is now linked to the relevant characteristics by characteristics relations:

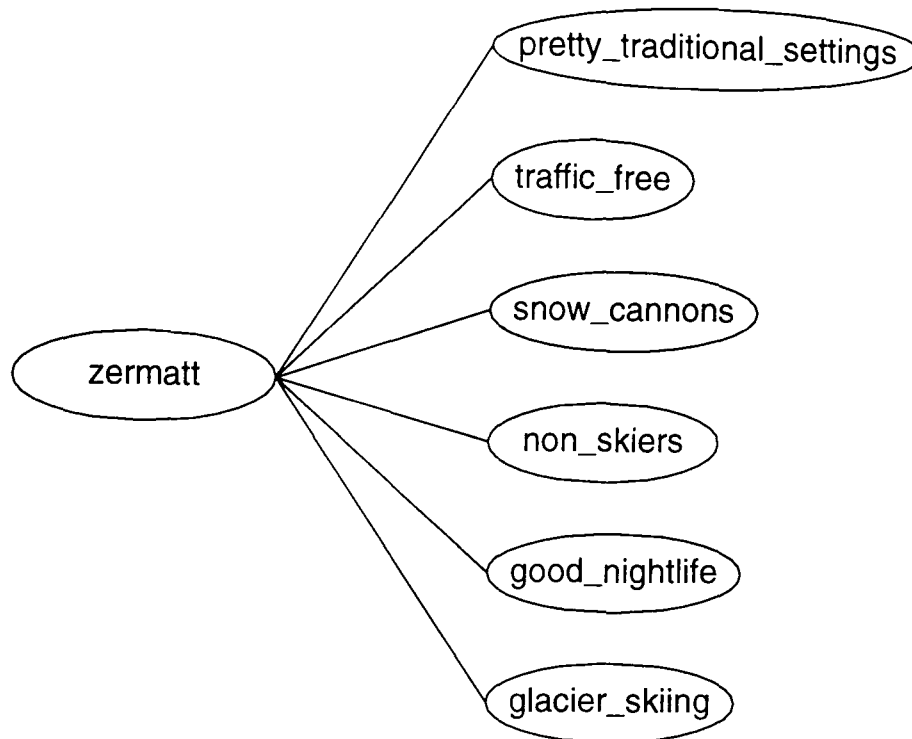


Figure 58. Zermatt's characteristics.

So the relation definition looks like this:

```
(characteristics
  (is-a relation)
  (inverse characteristic_of))
```

And an example of a resort frame looks like this:

```
(zermatt
  (characteristics pretty_traditional_settings
    traffic_free
    snow_cannons
    non_skiers
    good_nightlife
    glacier_skiing)
  (is-a resort)
  (place Austria))
```

Each resort has a variety of accommodation on offer. The accommodation class is subdivided into hotels, apartments and chalets. These types of accommodation will have characteristics in the same way that the resorts have and the same relational structure can be used to represent them.

In the Generic Browser the basic browsable structure is the path. In this example application there is a requirement to be able to browse the resorts. This is achieved as follows:

First a path is created with all the resorts in it.

```
gb_create_path("resorts",
               get_relatives("resort", "is-a"));
```

The application can now browse the resorts by simply using the relevant functions:

```
gb_next_on_path("resorts");
gb_previous_on_path("resorts");
```

To get the name of the frame resulting from the above use:

```
gb_get_current_frame_on_path();
```

A common requirement in browsing applications is to allow the user to flag those items of interest in the set he is browsing and to allow him to browse just that subset. The Generic Browser supports these processes as follows:

```
gb_mark("my_mark", NULL);
```

Here the current frame is added to the marked set *my_mark*. There is no value for this mark so NULL is used. Values are associated with marks to denote, for example, degree of interest in a particular item.

The marked frames can be browsed using:

```
gb_next_marked("my_mark");
gb_previous_marked("my_mark");
```

There can be any number of marked items in a marked set and any number of marked sets (each with a different name).

The next requirement that must be addressed is to provide the user with a means of indicating which characteristics of, say the resorts, he is interested in and selecting appropriate resorts from the frame-base. This selection process compares the specification given by the user with the characteristics of each resort and computes a similarity value. This value reflects how closely a particular resort matches the user's requirements. The Generic Browser vector search function provides this functionality, conveniently marking

each searched frame with a mark value of the similarity between that frame and the specification encoded in a special query frame.

To illustrate how to use the vector search function the requirement for selecting resorts will be used. The first step is to create a query frame. This contains those slots which it is desired to match. In this example there is a single slot, characteristic, which is being searched. The values in this slot are the specification given by the user. So, if the user wants to find a resort with *glacier_skiing* and *snowboarding_&_monoskiing* then the query frame will be:

```
(query
  (characteristics glacier_skiing
                    snowboarding_&_monoskiing))
```

This frame can be created using, for example, *cf_define_frame*.

The vector search function is now invoked as shown here:

```
gb_vector_search_class("resort", "characteristic"
                      "query", "search_tag");
```

There are two results of executing this search. Firstly a new slot named *search_result* will be created in the query frame. This will contain an ordered list of the frames which, to some degree, matched the query. These frames are listed in order with the closest match first.

Secondly, the matching frames are marked. The marked set is called *search_tag* and each mark has as its value how good the match was. The query frame and the matching resort frames resulting from performing this search are shown below.

```
(query
  (characteristics glacier_skiing
                    snowboarding_&_monoskiing)
  (search_result verbier st_anton zermatt))
```

```

(verbier
  (characteristics pretty_traditional_settings
    snowboarding_&_monoskiing
    snow_cannons
    short_transfer_times
    good_nightlife
    glacier_skiing)
  (is-a resort)
  (place Austria)
  (search_tag 2))

(st_anton
  (characteristics snowboarding_&_monoskiing
    snow_cannons
    ski_guides
    non_skiers
    good_nightlife)
  (is-a resort)
  (place Austria)
  (search_tag 1))

(zermatt
  (characteristics pretty_traditional_settings
    traffic_free
    snow_cannons
    non_skiers
    good_nightlife
    glacier_skiing)
  (is-a resort)
  (place Austria)
  (search_tag 1))

```

verbier is top of the list as it matches both search criteria. Both *st_anton* and *zermatt* match one apiece and follow in the list. The other resorts contained neither search term and so are excluded from the result.

To browse the result of the search the application simply browses the marked set named *search_tag* using

```
gb_next_marked("search_tag");  
  
gb_previous_marked("search_tag");
```

7.5 Summary

The functional requirements of the Generic Browser were laid out in chapter 6. Key features of the implementation of these requirements are summarised below.

The use of a frame representation scheme has been shown to be a feasible structure for implementing the Generic Browser. The relational links provided by the system can be used to link data frames into a variety of topologies ranging from simple linear lists through hierarchies to networks. A single function is used to traverse links from the current frame. Because relations are named objects in a frame representation, named links are automatically provided. This means that application designers can experiment with the possibilities of indicating the type of information available by following a link at the source rather than the user having to traverse the link to find out.

The use of a marks slot in the data frame to store mark-value pairs allows an arbitrary number of named marks (with values) on any item. To fulfil the requirement that a set of marked items can be treated as a single browsable object all items with a given mark are stored as a path with the name of the mark. This enables marked subsets of the database to be browsed without having to provide a special set of browsing functions (the path browsing functions being used).

Aside from its use to support browsing of marked subsets, the path construct enables sequences of data items to be treated as an object, a guided tour for example. Browsing functions for paths have been provided.

A further use of paths is their use to store a navigation history. As each browser function changes the current frame the history path is updated. A special backtrack function allows the history path to be browsed without updating the history.

Basic search functions that allow data items with specified slots, optionally with particular values, to be found. The result of these searches is a marked subset available for browsing (as a path). An alternative search mechanism based on the vector space model has also been implemented to demonstrate a means of non-exact matching. Again the search results in a marked subset. Because the searchable characteristics are stored as values within each data item's frame any items found by browsing can be used as a basis for further search to

find other items “like this one” by simply creating a new search query frame with the same characteristic slots and values.

Two requirements have not been implemented in the final version of the Generic Browser due to time constraints - negative query handling and function search. Mechanisms for how these could be achieved within the Generic Browser framework were described in the last chapter and there should be no problem in adding these functions to the Library.

A characteristic of this implementation is the uniform use of frames for all of the data structures used by the Generic Browser both external and internal to the browser itself. Paths are stored in frames and marks are stored as paths. Searching takes place on frame representations of the data and the result is a path of marked frames. The query is also held in a frame. This use of frames means that the data is both readable and modifiable by the author and at the same time amenable to efficient manipulation by the browser functions.

Two examples show this: the choice of representing a path as a single list in the path frame as this is most intuitive for an author and the decision not to use instances for data nodes to form a path because of the difficulties of naming instances in such a way that the conceptual framework was not blurred by the needs of the path representation.

Clarity has also been maintained at the expense of some performance degradation by the decision to store the search characteristics in the frames. This is also important because it preserves the uniformity of representation between the query and the data and ensures that the characteristic information is available in a browsing as well as for the search. The trade-off between caching commonly required information and calculating it when required was also highlighted. On systems such as CD-i it is better to do a little work often (cache information such as the items in a marked set) rather than a lot of work from time to time (searching all the frames for those in the marked set) because, in general, the user is not tolerant of long delays. The result, an evolving prototype library, has been tested by using it to construct a series of prototype applications. These are described in chapter 9.

8 User Interfaces

The field of research into user interfaces is a large one and it is not the intention of this thesis to address this area in detail. The purpose of this chapter is to see what user interface issues are raised by the Generic Browser approach and how these can be addressed.

The user interface of an application can be divided into two components, the so-called *look* and *feel*. The *look* is the presentation component and determines what the controls such as buttons and sliders look like, together with the overall graphical design of the application. The *feel* is the interaction mechanism; how the buttons, sliders and menus work. Together these two components implement the interaction design of the application.

For a browser the user interface often appears simple consisting of just previous and next buttons. Even if browsing does not present a major challenge to the interface designer the addition of search does pose some problems especially in the consumer domain. In particular how should the user specify search criteria - the form filling interface used in professional applications may not be suitable particularly if a text keyboard is not available. The presentation of the search result is less of a problem as in the Generic Browser this is available as a marked subset for browsing interaction.

In addition consumer applications are seldom just an interface to data often having some kind of location structure (screens) which the user navigates. The link between this and navigation within the data is investigated to see if it is desirable to integrate them and if the same functions can be used for both.

These issues and others have come to light as a result of, in particular, the prototyping work and are discussed below.

8.1 Generic Browser Interface and Widget Set

One question raised early in the development of the Generic Browser was the possibility of there being user interface controls which are specific to browse/search applications. As the prototypes were developed this possibility was borne in mind but none were found. The considerable research that has gone into the refinement of the basic user interface widgets provided by Graphical User Interface (GUI) toolkits made such special controls unlikely but it was useful to have this demonstrated in practice.

It was also suggested that there might be a generic interface for browsing applications. Again the study of browsers did not indicate this, partly because the look of an application is very important in the consumer domain. What can be said however is that whilst there does seem to be a convention of conducting browsing using some form of previous and

next buttons, no such convention exists for search so that the interface to search facilities is very much up to the interaction designer.

Over the past few years there has been a shift from bespoke user interface development from scratch to the use of GUI toolkits. (This is exactly the type of shift that the Generic Browser promotes for data structuring and access.) A GUI toolkit provides a standard set of widgets (buttons, sliders, cursors, windows etc.) which can be instantiated to form the interface to a specific application.

This GUI layer, the functional component of the user interface, should not be confused with the presentation layer which specifies the physical manifestations of the interface objects. For example the GUI layer specifies a button of a certain size with a specified callback, the presentation layer specifies the button graphics i.e. how it should appear on the screen. This “look” is defined by the designer and is usually unique to one application or a range of related applications.

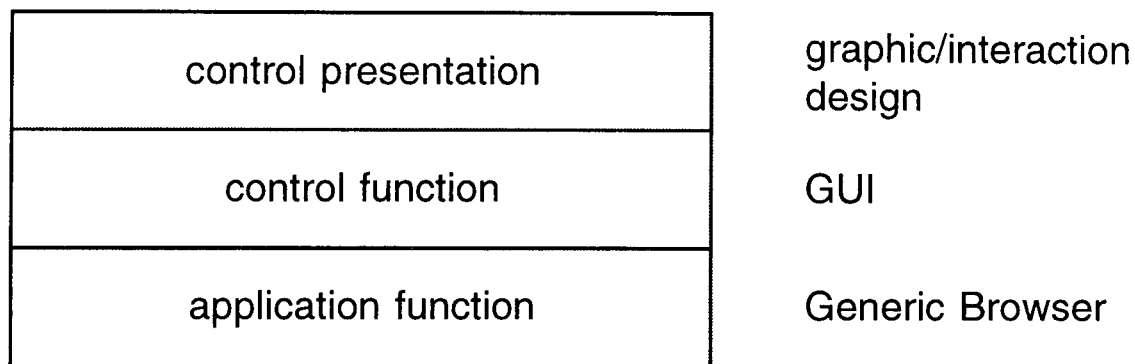


Figure 59. Application interface structure.

In most GUIs functions known as “callbacks” are associated with the interface widgets such as buttons. When the button is “pressed” the callback function is executed. From the Generic Browser perspective, browser functions are most likely to be found in these callback functions for buttons. For example if there are “previous” and “next” buttons in a simple browsing application *previous_on_path* and *next_on_path* functions will be found in the button callbacks.

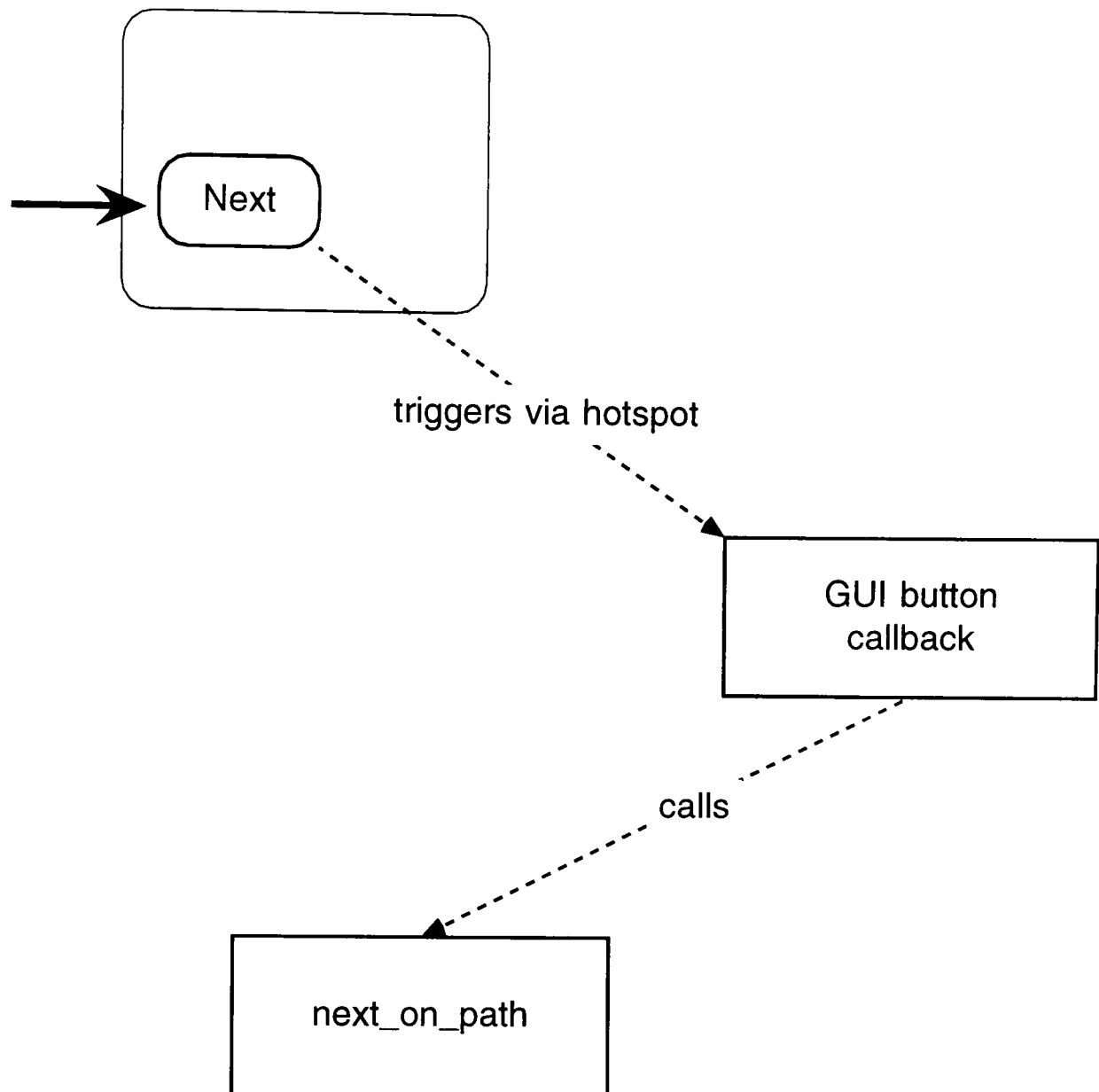


Figure 60. How the user's input triggers the Generic Browser.

8.2 Navigable Structures

Throughout the development of the Generic Browser there has been a clear separation between the browsing functionality and the user interface. It was anticipated that information-rich applications would have a simple interface to the Generic Browser functions which would control the navigation through the information by telling the application where the information is. The presentation of that information would be the responsibility of the application. This division is similar to that proposed in the Dexter Hypertext Reference Model [Halasz & Schwartz 1990] where a storage layer contains descriptions of the data and a runtime layer presents the data to the user according to presentation specifications.

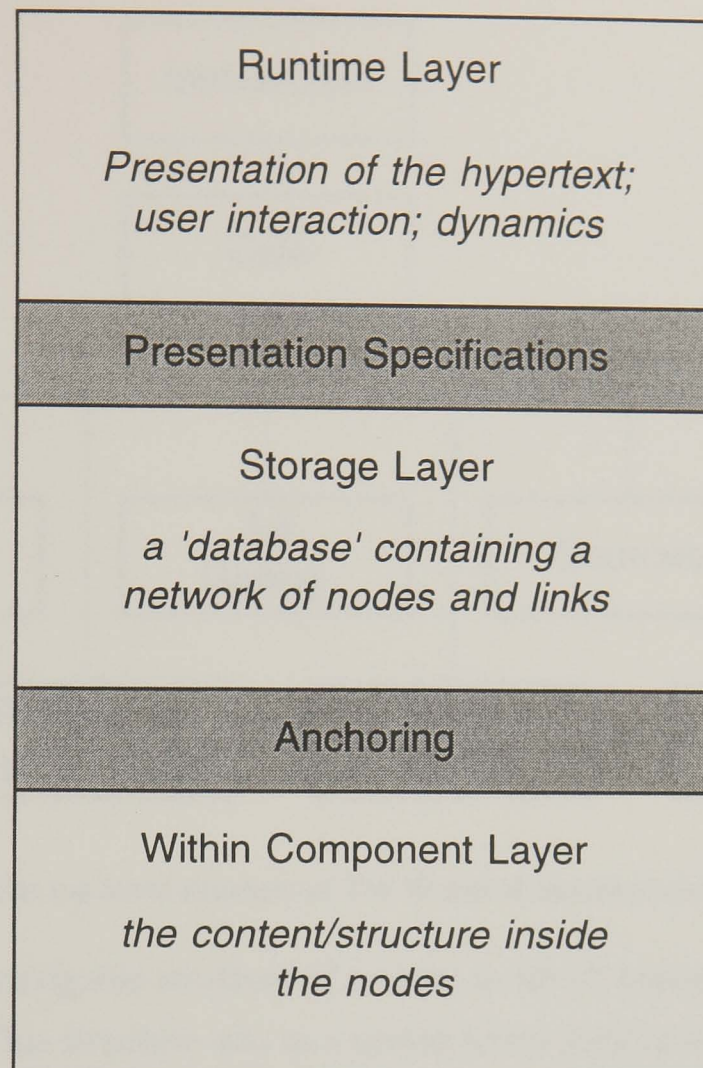


Figure 61. Layers of the Dexter Model [Halasz & Schwartz 1990].

Essentially the Generic Browser model has one location with many assets. As existing applications were studied it became clear that an alternative model with the application viewed as a network of locations each with one or more associated assets was also prevalent in consumer applications. In such a structure there are significant links between the browser and interface components of the application. There is a browsing navigation interaction as the user traverses the structure of the actual application itself, screen by screen, and it is necessary to see how such a model can be accommodated by the Generic Browser. As an example The World of Impressionism CD-i disc (described in detail in chapter 9) has such a location structure as shown below:

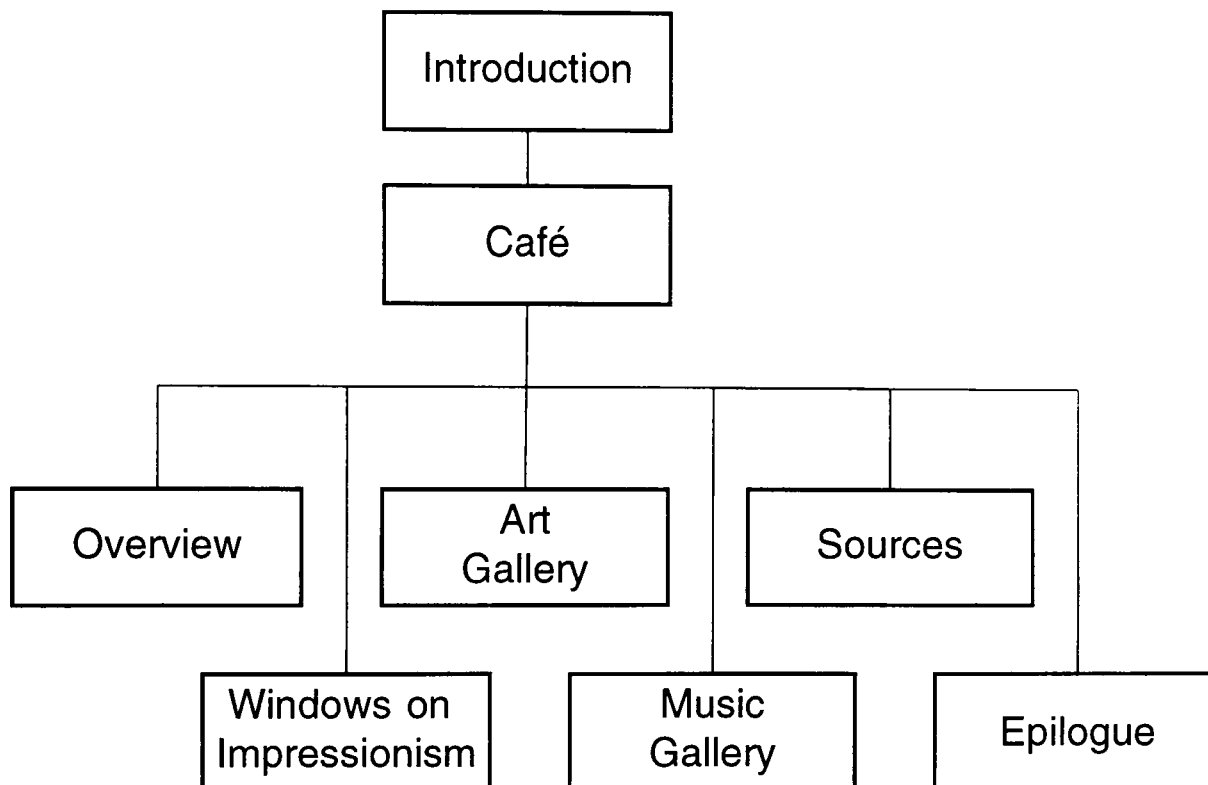


Figure 62. The top-level structure of The World of Impressionism CD-i Disc.

There is a hierarchical navigable structure of screens in which buttons allow movement between the screens. This structure acts as a nested menu leading to leaf nodes which are browsers as shown below.

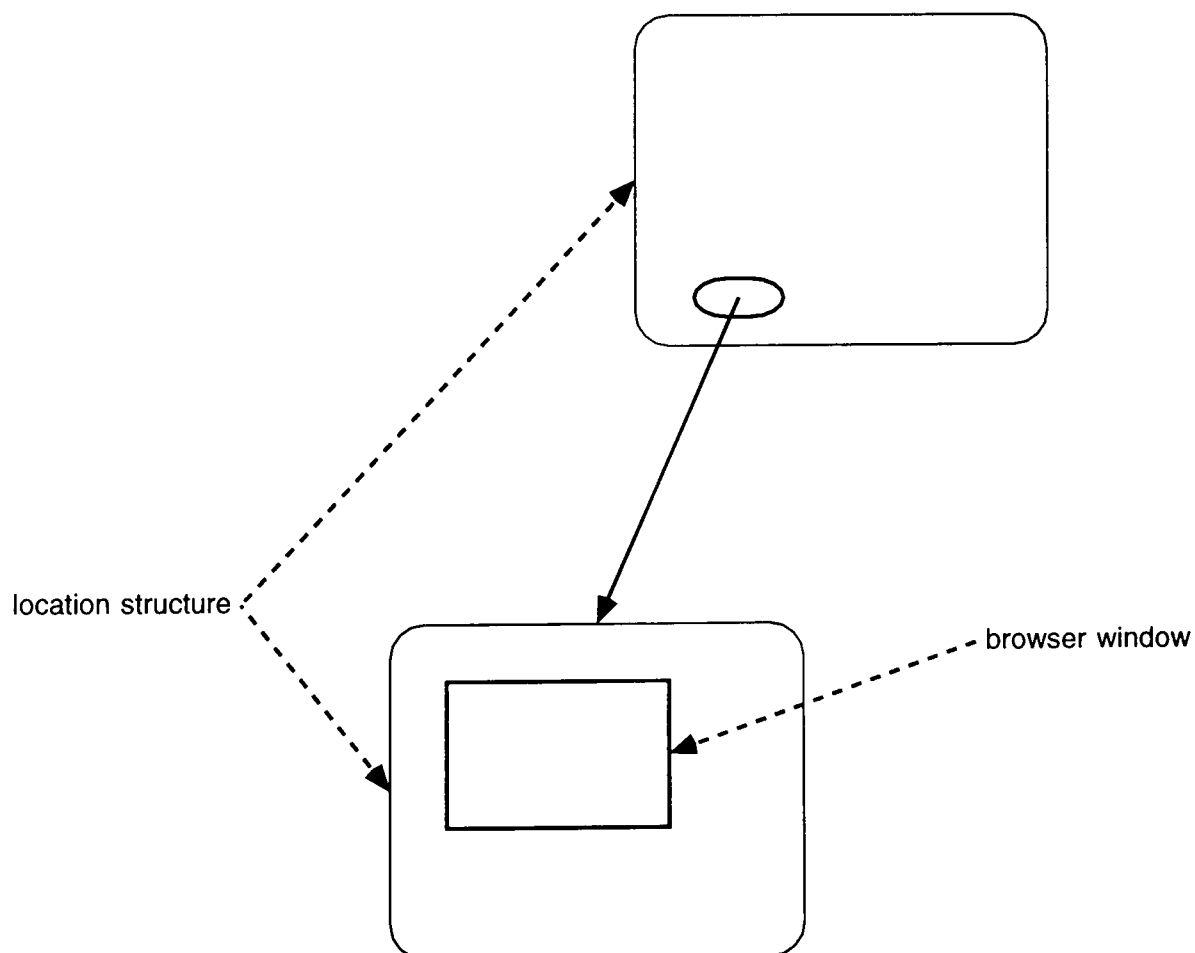


Figure 63. Hierarchy with browsers at leaf nodes.

The important decision that has to be made when designing such an application is where the dividing line between structures held as data and structures embedded in the program code

should be made. In this example the browsable information would be held as data but the location structure would be embedded in the program. Embedding the basic structure within the program code has been a feature of many applications developed for CD-i so far.

However the trend within authoring is towards having more of the program structure as data because this makes application creation simpler and therefore less costly. The route taken to achieve this is typified by Apple Media Tool (AMT) on the Macintosh. Here the author puts together the structure of the program using a graphical editor. The nodes of the structure are the locations that the user can visit, the links define how they get there and include hotspot information, picture transitions etc. A similar level of functionality is provided by MacroMedia Director. As has already been mentioned none of these tools addresses the needs of data structuring and this suggests that a marriage between this type of tool and the Generic Browser would solve the problem of data browsing and search in applications developed with these tools.

To do this integration would require that the Generic Browser function set be made available so that these functions could be executed when a link were traversed or button pressed. Essentially AMT can be regarded as providing the GUI functionality with Generic Browser functions used in its callbacks. Of course since AMT is an application creation environment the whole application and not just the interface can be developed using it.

An alternative to this approach of combining the Generic Browser with an existing authoring system is to have "location structures" represented as frames. The data items could be linked as leaf nodes to the locations, inheriting (across instance or user-defined relations) all the interface information such as where to go next as shown in the diagram below.

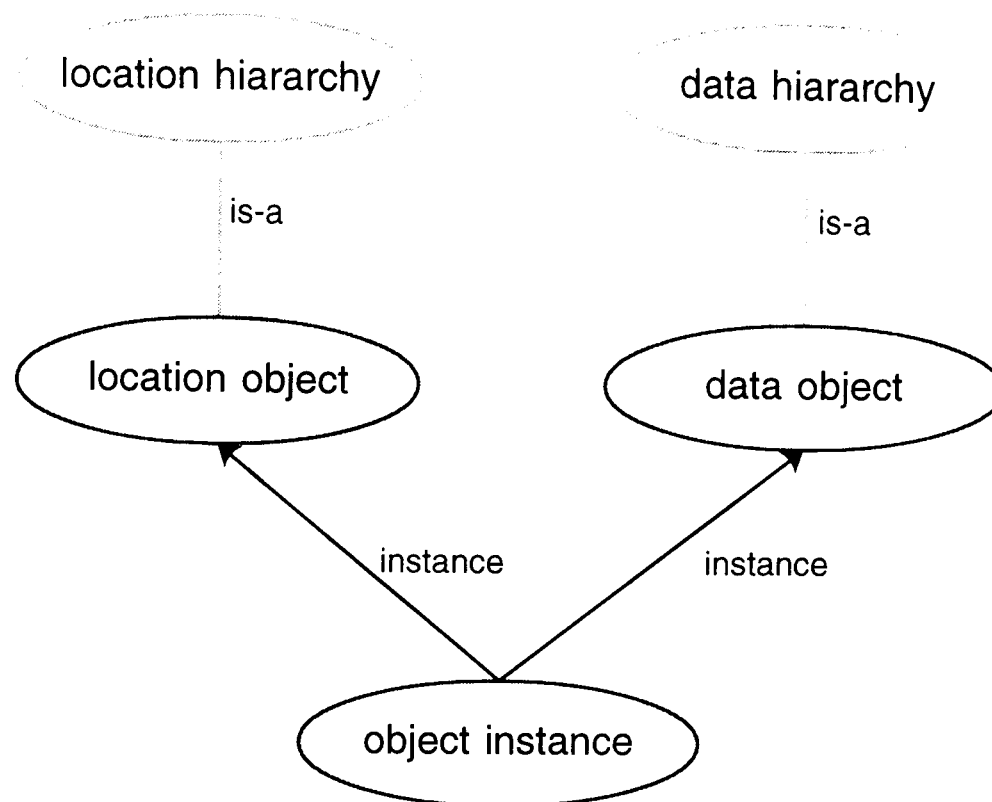


Figure 64. Object linked to both location and data hierarchies.

In this example the *object instance* is an instance of both a location object and a data object. It would have links to form a navigable structure such as that in *The World of Impressionism*.

Some frame systems would not allow such a stretching of the semantics of is-a/instance, only permitting an object to be a member of one class. An alternative would be to use another relation, provided that inheritance is permitted over such user defined relations. In both these cases the system must allow multiple inheritance. If this were not permitted the display mechanism would have to traverse the link to the location object to extract the display information.

The problem with this approach is that once the basic location structure is represented within the frame description it becomes necessary to continue with the integration and include the user interface objects such as buttons and their hotspots as well. This leads to an authoring environment like those described above.

The representation of hotspots is a particular problem in the 'C' implementation of the Generic Browser described in this thesis. Hotspots require callbacks which are functions. This would mean that some reference to the callback function would need to be stored in the frame representation. It is not possible to just store the name of the callback function since the function must be callable. It would be necessary to store the address of the function. This is only available at compile time requiring special post processing of the compiled code to extract this information and store it in the frames. This is not really practical and it ties together the development of the program control code and the authoring

of the database which is what storing the location structure as data was designed to avoid. It is interesting to note however that callbacks would be easy to implement in the LISP version of the Generic Browser as there is no distinction between code and data in LISP and the value (name of the callback function) stored in the callback slot could be executed.

8.3 Presentation Aspects

Presentation is the look of the application both its graphic design and also the views of the data. For instance a path can be viewed both one data item at a time in a simple browser or with the whole path displayed as a map. Other views are indexes to the entire database or to subsets such as only those items marked as interesting by the user. The Generic Browser provides sets of data to the application program. It is up to the application designer to decide how best to display this information to the user; to choose the view.

For some applications it may be desirable to store some display information with the data. The flexibility of the frame representation allows information about the display of a given item to be held in a variety of places depending on its generality. For instance the position of an image could be stored in the picture description frame if that image is always displayed in the same place. If, on the other hand, the image is used in several situations at different screen locations instances of the picture description frame can be used as shown below.

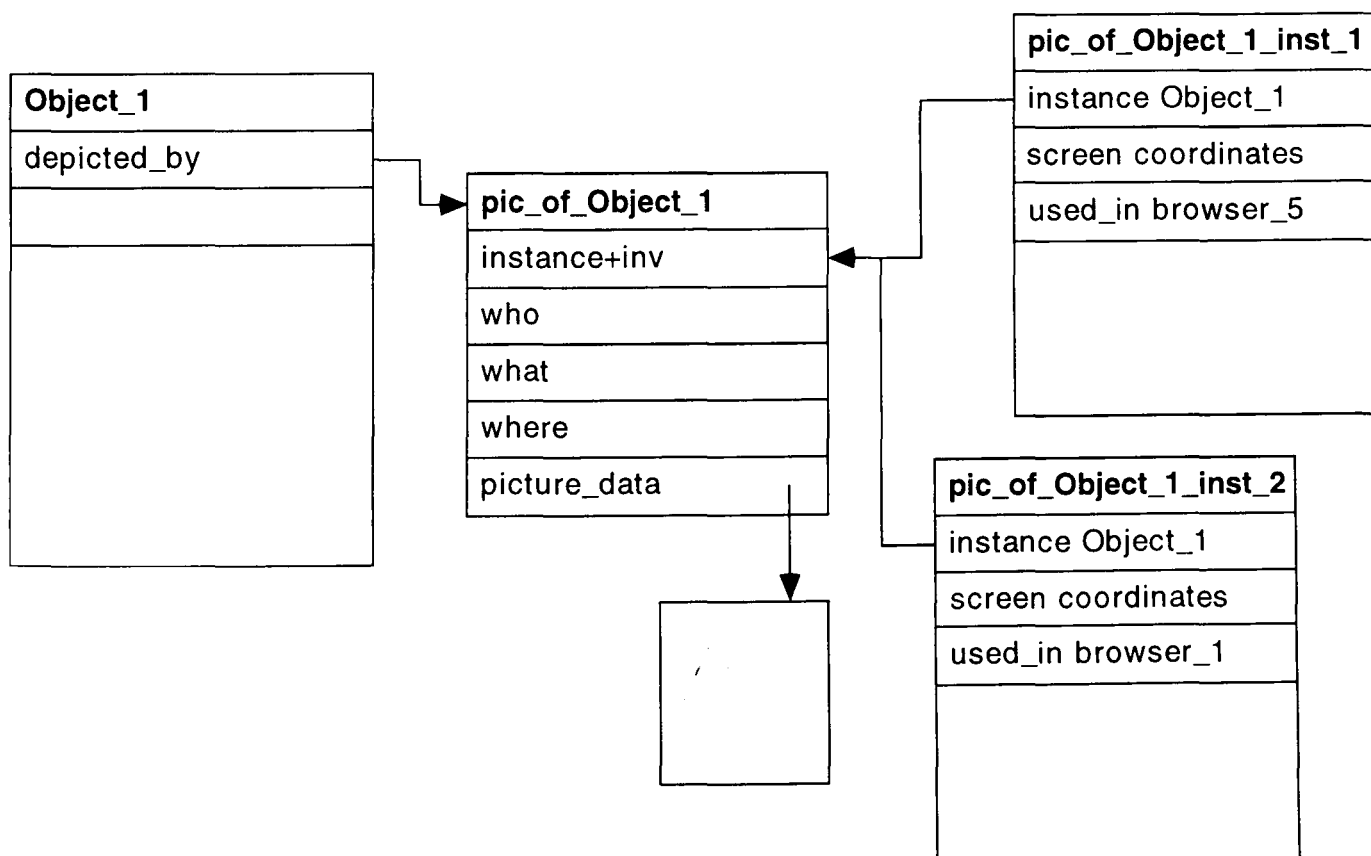


Figure 65. Use of instances of a picture object to store display specific data.

Most information will be inherited but the screen coordinates can be different in each instance.

8.4 Transition effects

As well as information on how an individual item will be displayed transition effect information is also required so that visual effects such as wipes and dissolves can be used as the user moves from one picture to the next. (Similar effects such as fades can be used between audio items.) There are two possibilities in the Generic Browser, transitions between items joined by links and between those on a path.

8.4.1 Transitions Between Linked Items

One approach would be the inclusion of transition effects within link descriptions. To do this in the Generic Browser links would have to become instance frames of a class of the link type. So, for example if there was a *more_information* link which was to be described in this way there would have to be one instance of this link-type for each time the link was used. These could then have extra transition information stored within the link instance frame as shown in the diagram below.

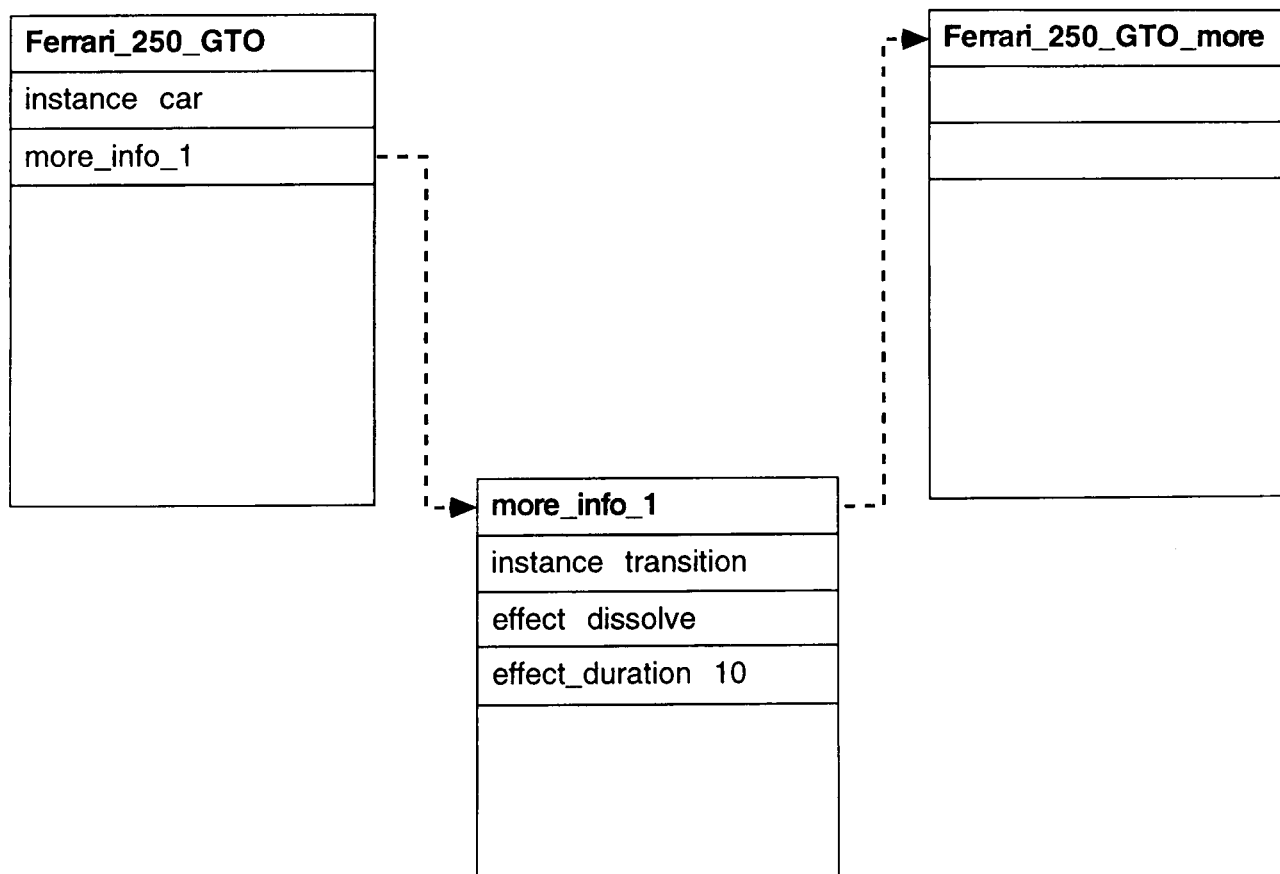


Figure 66. Transition information stored within link frames.

Note that this method only works for one-to-one relations and not one-to-many, a new relation would be used for each item of related information in this latter case. The problem is then how to ascertain all the possible linked items from a given node. One possibility is for the *get_relatives* function in the underlying frame system to be modified so that it can take a class of relations as a parameter, rather than a single relation. *get_relatives* returns a

list of all the frames linked to its first parameter by the relation given as its second parameter. For example, suppose that all the relations linking the *Ferrari_250_GTO* frame to other items of information were members of the class *additional_information*. A call to the modified *get_relatives* would be:

```
get_relatives("Ferrari_250_GTO",  
             "additional_information")
```

This would return not those items linked to *Ferrari_250_GTO* by the *additional_information* relation but by any relation in the class *additional_information*.

8.4.2 Transitions Between Items on a Path

For paths the situation is more straightforward. One solution is to intersperse the assets with the effects in the path list. For example:

```
(capital_tour  
  (is-a path)  
  (path London Effect_1 Paris Effect_2 Rome.....))
```

which give a structure as shown below:

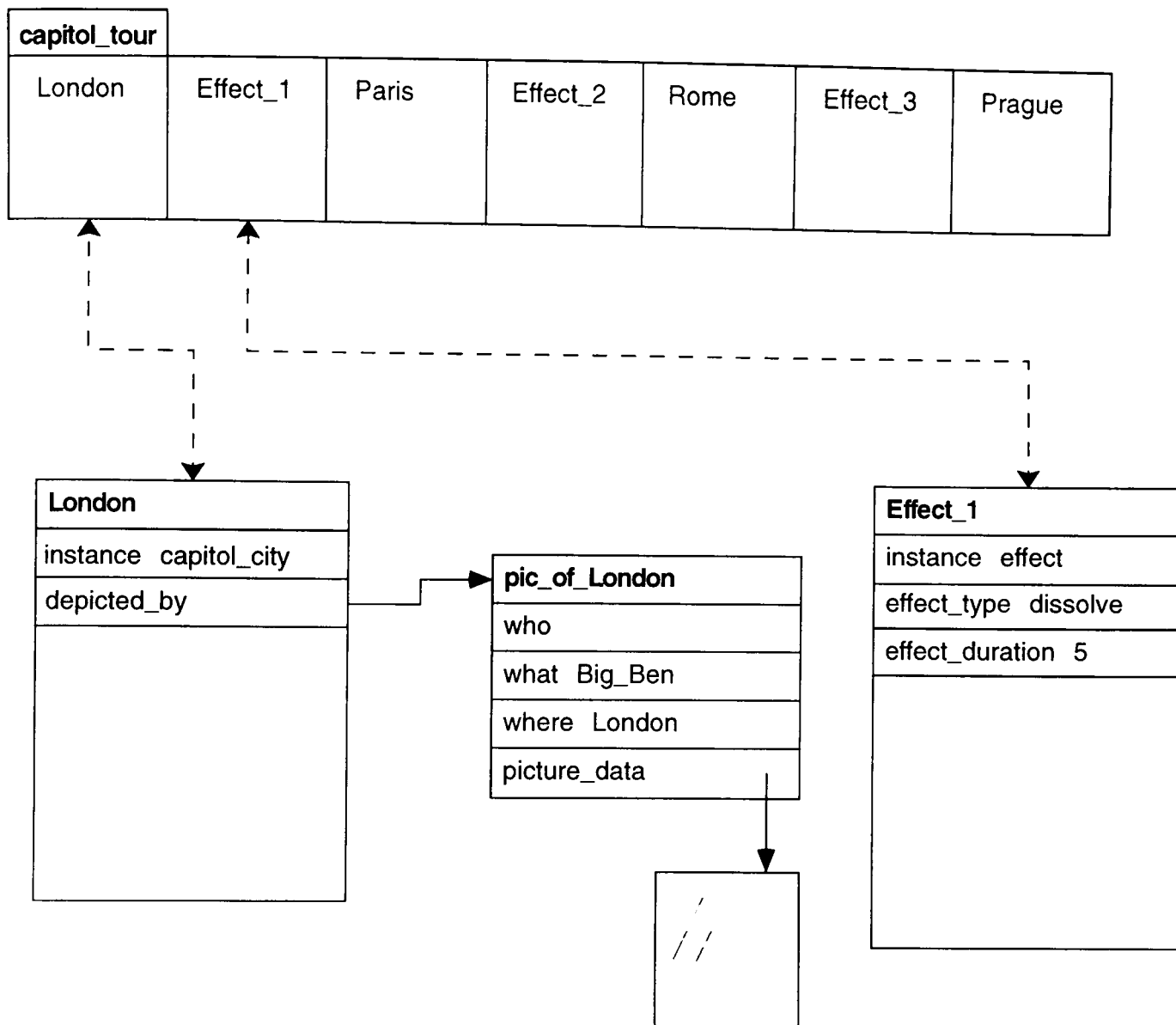


Figure 67. Effects between objects on a path.

Though solutions have been shown for storing display-related information within the database the most tidy solution is the integration of the Generic Browser functionality within an authoring environment such as AMT. The data structure can be constructed by a content expert and the application structure put together by a graphics and interaction designer with little actual programming (in the coding sense) being required.

8.5 Constraints Imposed by the Target System

The final aspect of the user interface which deserves mention here is that of the particular constraints imposed by the system on which it is implemented. The look and feel of an application can be strongly influenced by that target system. This is particularly true of the more modest hardware in the consumer domain. The look is influenced by display facilities such as the number of colours that can be displayed, the screen resolution and the transition effects that are available. The feel is affected by the interaction devices such as keyboards, mice and other controllers. As an example the CD-i player does not have a keyboard, relying on an X-Y position device with two additional buttons. This means that interfaces to applications on this type of system must address the additional problems of, in

particular, the lack of a text entry keyboard. It is therefore important to be able to offer search capabilities without the user having to be able to type search terms into the system as would be the case with database query languages such as SQL. Techniques such as indexes, context sensitive term completion and query by example need to be examined and evaluated to demonstrate their utility or otherwise in such a restricted environment. Resolution of these problems will also benefit the design of applications running on less restrictive hardware if that design allows operation by the use of a point and click type of interface as this should be more convenient for the user than one in which text must be typed in.

8.6 Summary

This chapter has considered how user interface issues raised by the Generic Browser concept can be addressed. In particular special widgets have not be found to be necessary for browse/search application interfaces, nor is there a common “look” to such applications. It is suggested that applications structured round a network of graphical screens, some of which containing browsers or search facilities could be represented using the frames used in the Generic Browser but that a better route would be to integrate the Generic Browser into an application development tool such as Apple’s Media Tool or Macromedia’s Director.

The next chapter looks at the way in which the Generic Browser has been evaluated.

9 Evaluation of the Generic Browser

The purpose of the evaluations described in this chapter is twofold. Firstly continuous testing and evaluation has been possible using the prototype library implementations and applications. This has allowed the progress of the research to be monitored against the evolving specification the final version of which is described in chapter 6. Secondly the final version of the Generic Browser is evaluated to measure whether the objectives set in chapter three have been met by the Generic Browser. In summary these objectives are firstly:

to show that a suitable basis for the construction of information-rich multimedia consumer applications can be provided by a core set of information access functions operating on a flexible data representation and that the function set should integrate browsing and search facilities.

Within this primary objective the following subgoals were defined:

development of a suitable multimedia data model

demonstrate how browsing and search could be integrated

select a core function set

These goals were to be addressed by the construction of a function library that meets the requirements developed from these goals. Thus the implementation of the Generic Browser is the evaluable object in this work i.e. does it demonstrate the premises summarised above?

In common with any software library the Generic Browser presents some problems when it comes to evaluation. With complete applications it is possible to test whether the application fulfils the user's requirements. With a library, which is a component of the final application, this is more difficult. This is largely because shortcomings of the library can be masked by the application - in a sense software can "do anything" given a skilled programmer.

To overcome this problem the testing of the Generic Browser divides into three strands. Firstly there is the basic functional testing of the library to ensure that it performs as specified. Secondly the Generic Browser has been used to build a number of prototype applications. By careful monitoring of the coding process - actively looking for code which is only present to circumvent a shortcoming in the Library, it has been possible to avoid the pitfall of coding around any problems. Each prototype has assisted in refining the specification of the library. The third thread of this evaluation has been a paper study

considering how an existing application could have been built using the final version of the Generic Browser. That is what data structures would be required and how the Generic Browser functions could be used to reproduce the functionality of the application. The aim here has been to ensure that the prototypes applications were not biased to the capabilities of the Generic Browser.

Thus the Generic Browser has been evaluated throughout its development. It has been subjected to continuous evaluation through the development of the prototypes both at the concept level - are these the right functions for the job? and at the implementation level - do the functions perform correctly?

9.1 Formal Test Program

As detailed in chapter 7 a test program was written with the aim of exercising all of the routines in the Library. As well as valid arguments used to test the correct functioning of the routines, invalid arguments were also supplied to each routine to test the error handling built into the functions.

9.2 Prototype Applications

As has already been pointed out it is difficult to assess function libraries alone since their utility can only be demonstrated by using them to produce applications. Accordingly considerable use of prototype applications has been made to evaluate the progress of the Generic Browser. Feedback from the creation of these applications formed the basis for modifications to the library whereupon the cycle was repeated. It must be emphasised that for all of the prototype applications the primary concern is functionality not look and feel. Each of these prototype applications is described below.

9.2.1 Simple picture browsers

This first prototype was built before the full Generic Browser specification was worked out. It was constructed to explore and experiment with preliminary ideas of frame browsing namely traversing a network of frames linked by *previous* and *next* relations.

This browser was implemented in XLISP and XKC on a Sun and subsequently ported to the CD-i player. The library required no changes to run on the player, only the display/interaction part of the prototype needing to be recoded. This is in line with the requirement that the Generic Browser be platform independent. This prototype application consisted of a simple linear browser of ten frames, each a member of the class *item*, linked together by *next/previous* relations. Each frame had a picture associated with it and the display portion of the program used the filename held in a slot in each frame to load the

picture associated with the current frame. As a trivial demonstration of the data independence of this application, a single command modified all the picture file names from those of a set of vintage cars to that of a set of picture of places in Italy. Thus the frame structure remained constant but the content of the frames being browsed changed.

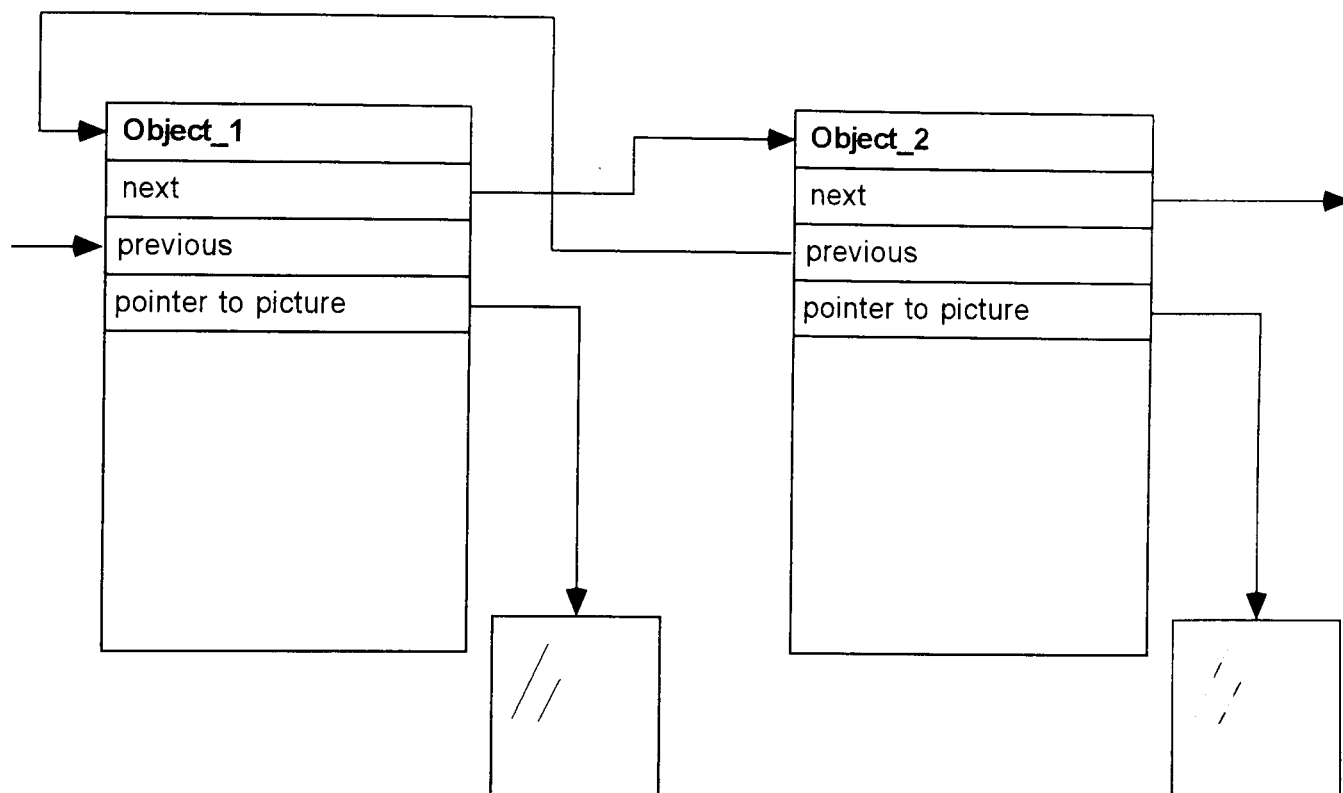


Figure 68. Linked frame structure of the picture browsers.

This prototype highlighted the need for a path structure. Specifically, it was not found to be possible to create a variety of different sequences using the previous and next relations. A different approach was required and the path structure was developed as a result. Evaluation of this new component in the Generic Browser was started in the next prototype.

9.2.2 The F1 Browser

To provide demonstration materials for an internal workshop, where Cframes and the Generic Browser were to be demonstrated, two further browser applications were planned. The first was to be a browser of Formula One motor racing events. The envisaged use of such a browser would be for use in selling motor racing holidays. The second browser was to be one with information about guitars, their players and their manufacturers. Time constraints led to one prototype containing two sets of data being built, but it is useful to distinguish between the two browsers although they were, in fact, just different options within one larger demonstration program.

The F1 browser extended the simple picture browser to include text fragments and multiple pictures associated with each item.

Though the F1 Browser was initially coded in XLISP/XKC performance problems forced a reappraisal of the use of XLISP on the CD-i player. A new version of the Generic Browser written in 'C' and using the Cframes frame library was then written. The F1 Browser was re-written to test these new libraries. A simple terminal-based text menu interface with pictures on the CD-i player screen allowed all the Generic Browser functions to be tested.

The browser offered the following options on its main menu:

print all frames
print all marked
print all path
print slideshow
print all tagged
print current frame

These functions allow the current state of the browser to be investigated. The user can check that, for instance, marking has taken place correctly by printing the current frame followed by all the marked items to see if it is there.

print history list

This option prints the contents of the history, node by node, to the terminal.

backup through history list

This is the function that is usually found in the callback of the *return* or *go back* button of an application.

goto frame

Go directly to any frame in the database. This option calls

```
goto_frame(select_from(get_relatives("item",  
                                "is-a")));
```

where *item* is the root node of all the information nodes in the database. The *get_relatives* call finds the frames connected to *item* by an *is-a* relation. The *select_from* function simply allows the user to choose from a menu.

goto marked frame
goto frame on path
goto tagged frame

These allow the user to go directly to particular frames in the marked set, on a path or in the tagged subset. In the very first version of this application there was only one marked set with the members linked by *next marked* and *previous marked* relations. This required the following code for this option:

```
goto_frame(select_from(get_relatives("marked_items",  
                                  "previous_marked")));
```

With the development of paths it was decided to treat marked items as members of a path rather than items linked by special relations. This means that navigating a marked set and a path use the same mechanism allowing all subsets to be treated in the same way.

```
goto_frame(select_from(append(reverse(get_values(  
                                "my_path", "back")),  
                              get_values(  
                                "my_path", "forward"))));
```

The complexities introduced by the use of *forward* and *back* slots in the path frame can also be seen here. As explained in chapter seven these were superseded by a slot with a list of the path members and an index. Again, acquiring the members of a path becomes a provided function in later versions of the Library so this type of code would be replaced by, for example:

```
goto_frame(select_from.gb_get_path("my_path"));
```

At this stage proper object oriented techniques had been only partially observed in the Generic Browser. In particular the *current frame* was a global variable. This was, of course extremely dangerous because modification of the *current frame* was possible by the application, circumventing the Generic Browser. Later versions of the Library have functions for accessing the *current frame* ensuring that all access to this variable is through the Generic Browser (which can then record changes in its history).

It can also be seen that the concept of tagging remains distinct from marking at this time. Later versions of the Library combine these under the term marking, allowing an optional value (tag) to be associated with a mark.

mark

Marks the current frame with a specified mark, e.g.

```
mark ("my_mark" );
```

mark path

This option added the current frame to a path. In a demonstration a path could be built up by navigating around the information nodes adding them to a path at will using this option. The result could then be played back as a slideshow.

next

previous

These options exercised the traversal of frames linked by *next* and *previous* relations. In this version of the Library there were special functions (called “next” and “previous”) for this purpose. These were included because it was thought that the basic navigation of a browsing application would revolve around a basic architecture of next and previous-linked nodes. Further studies revealed that this is a simplistic view. Applications typically have a number of subsets of nodes which need to be accessed by next/previous-type browsing. These are catered for in a much more suitable manner by the use of paths.

next marked

previous marked

Previous/next browsing for the members of a marked set.

next path

previous path

Previous/next browsing for the members of a path.

next tagged

previous tagged

Previous/next browsing for the members of a tagged set.

slide show of history

This option simply allowed all of the nodes visited to be re-displayed as a slideshow.

slide show of path

This option presented the members of a path as a slideshow.

slide show

This option was a pre-programmed slideshow (it also used a path for the members of the show).

tag

untag

These options allow marking with a value and the removal of same.

unmark current frame

Strips off the mark from the current frame. In this application only one mark is used. Full generality would require the user to select from a list of all of the marks associated with the current frame. This again demonstrates the need for the unification of marks and tags because the user may well not make any distinction and want all marks (tags included) to appear in this list.

remove from path

Takes the current frame off the path. In this application only one path is used but the Generic Browser allows as many as required.

use F1 pictures

use guitar pictures

no pictures

These allowed different databases to be tested within the same application framework.

The main contribution of this application was as a testbed for the re-implementation of the Generic Browser in 'C'. It was possible to compare the operation and performance of browser functions in the LISP and 'C' versions of this prototype.

The path structure was also implemented in the version of the Generic Browser used for this prototype. The combination of the practical exploration of the path concept in this implementation and the results of the parallel literature study characterising browsing led to the addition of the path as a browsable object in the Generic Browser specification.

9.2.3 The Guitar Browser

With the conceptual structure provided by Phil Lloyd [Lloyd 1991], this was the first (if small) demonstration that the Generic Browser library could be used to build a realistic application - the previous example being more of a testbed.

The aim of the Guitar Browser was to have pictures, text and sound associated with each item, to have more than one browsable subset (manufacturers, guitars and players) and to demonstrate content browsing using links such as makes/made by, plays/played by as well as the normal next/previous. The Guitar Browser further explores the problems of previous/next relations by having these three browsable subsets.

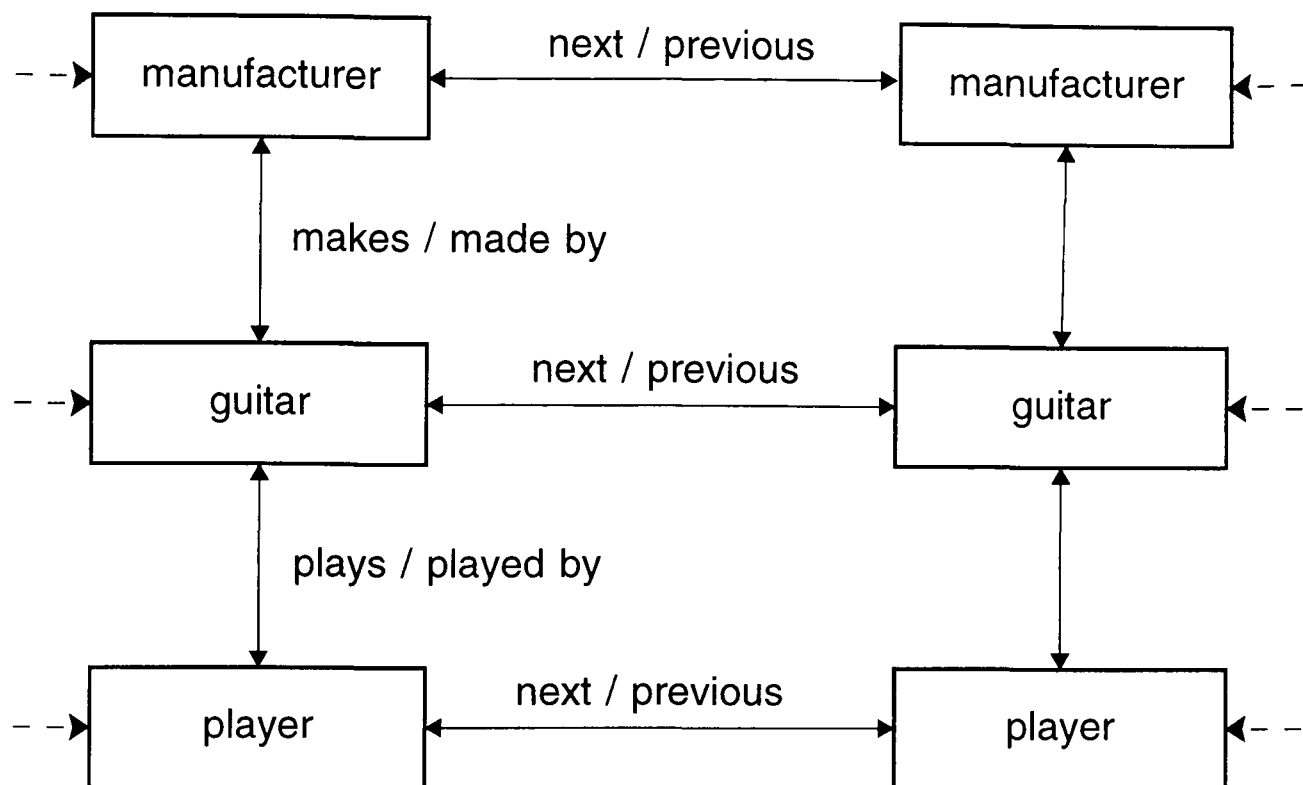


Figure 69. The Structure of the Guitar Browser.

No significant modifications to the Generic Browser library were required to support this application. However the use of next/previous links to define browsable sets as used in this example was recognised as being too restrictive. It works here because each of the sets, manufacturers, guitars and players is completely separate from the others. So a *next* relation can be used. If, however, an item needed to be in two browsable sets (if a player were also a manufacturer, for instance) the *next* relation would have to have two destinations. This is within the syntax of relations but has no meaning since there is no way to know when traversing the links in one set (manufacturers, say) whether the next item is the player or manufacturer just based on the information encoded in the relation. This resulted in the move to the use of paths for subsets. So in this example each of the subsets manufacturers, guitars and players would become a path.

This example, whose development overlapped with that of the F1 Browser, provides concrete evidence of the need to use paths for subsets and not the *previous/next* relations which had originally been thought to be adequate.

9.2.4 Ski Browser

The addition of search together with on-screen interface elements via the PRL GUI library⁴⁰ marks the Ski Browser as an important prototype. From this exercise, a valuable insight into the problems of presenting search in a multimedia application on a CD-i player was gained.

The Ski Browser was designed as a multimedia ski brochure and is described in detail in chapter 7. To recap the idea was that a user of the system would locate a skiing holiday by choosing a holiday area then tell the system what sort of skiing holiday attributes were important to him. The system would then search the relevant resorts for those which most closely matched the specification. The user would then browse the resorts. At any time the user would be able to ask the system to find “similar” resorts (searching the database with the current resort’s characteristics). Once a resort was chosen the user could browse the hotels available there.



Figure 70. The Ski Browser control panel.

From an interface perspective the Ski Browser is a browser-based application. The current item is always being shown. A control bar offers the following options:

⁴⁰The PRL GUI library provided GUI facilities on the CD-i platform and was developed at PRL by Richard Allen.

previous and next buttons

These buttons allow the current path (resorts or accommodation) to be browsed. The current item is displayed.

previous marked and next marked buttons

These buttons allow the browsing of marked items. Items are marked either by the user (via the mark menu) or by the search process. In this application no distinction is made between user and search created marks although this would be possible if desired.

mark menu

This menu allows marks or tags (included for demonstration purposes) to be added to the item on display. An option for clearing all marks is also provided.

browse menu

This menu allows the user the choice of browsing resorts or the accommodation offered at a particular resort.

search pop-up

This dialogue box has options for specifying the search query (using the simple text menu shown below), setting the search query to be the set of characteristics of the current resort to allow similar resorts to be found and a button to perform the search. A slider control is also provided to vary the match threshold so that the degree of similarity required for an item to be a hit can be controlled - the displayed number of hits is updated as this control is operated.

slideshow menu

Marked items or the accommodation of the current resort can be displayed as a slideshow.

print menu

A number of diagnostic printout options are provided on this menu. This can be used to demonstrate the effects of, say, doing a search on the database.

The search process in this prototype was not integrated into the frame representation. The search characteristics were held as a separate data structure (a 'C' array) representing the matrices of the vector search method. This application allowed the vector search method to be explored in more realistic surroundings. It highlighted the need for full integration of the search process with that of browsing. Extra code was required to "bridge the gap"

between the browsing offered by the Generic Browser and the results of the search process. This was done by marking each hit from the search.

The feedback from this application was the use of marking to integrate search hits. This method for integrating browsing with search was added to the specification of the search component of the Generic Browser.

9.2.5 Cruise Directory

As part of the Esprit Homestead project a prototype application was built to demonstrate a version of the Generic Browser that was part of Philips' contribution to the work. Of the three retail partners in the consortium, Page & Moy, a specialist travel company, had requirements which were ideally suited to solution using the Generic Browser. Of the other two, the Barclays application was banking services and was not browse/search oriented, and the Freemans application, a clothing catalogue, was less advanced from a design point of view at that time.

Page & Moy's requirement was for a CD-i version of their (paper) Cruise Directory. This listed the cruises offered by all the major cruise companies, grouped by area of the world. Page & Moy's requirements were crystallised into an interaction design by designers at Barclays TV and Video Unit.

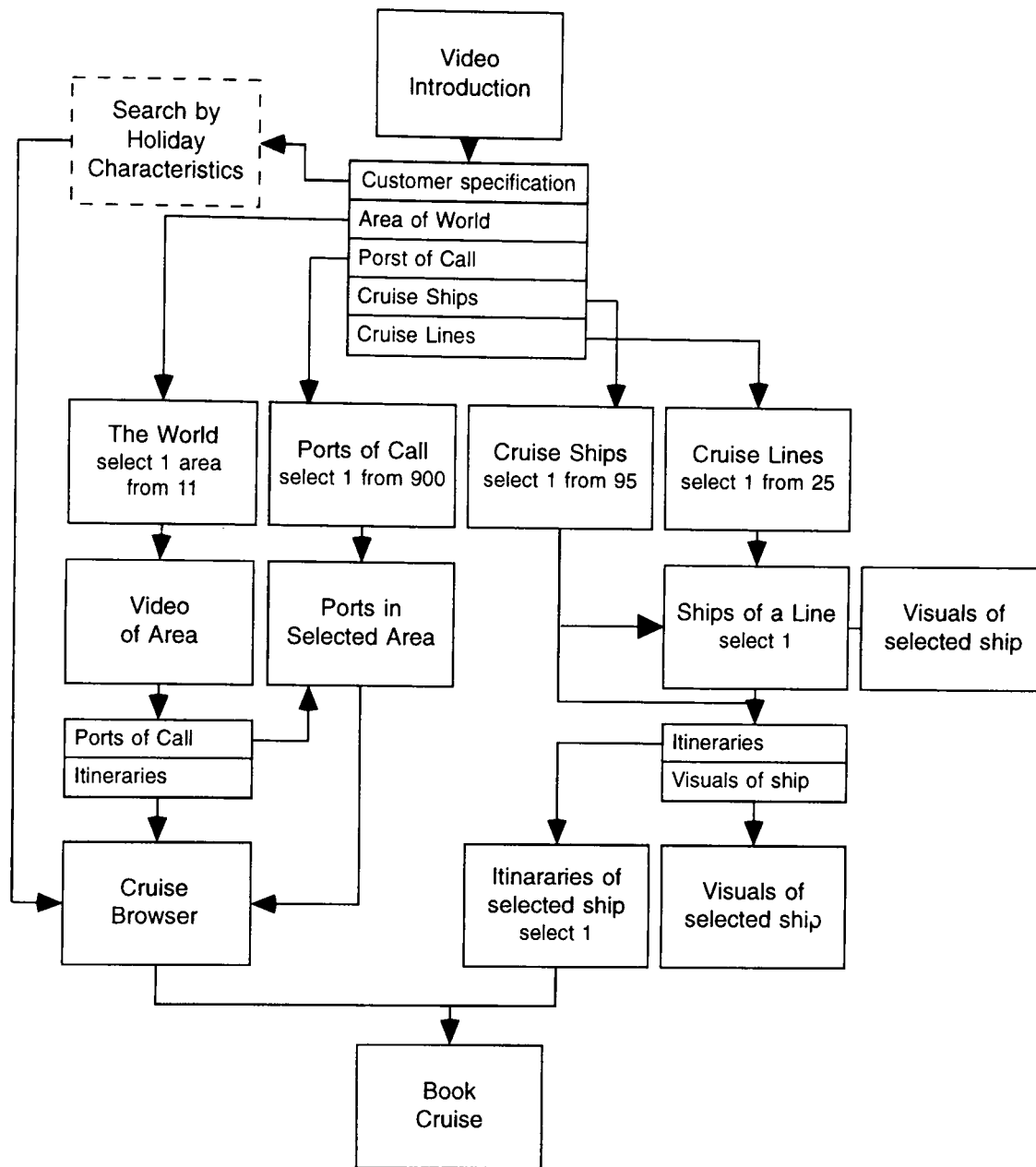


Figure 71. Cruise Browser structure.

With this specification the author built a first prototype using placeholder assets and with the CD-i support library BALBOA providing the hotspot mechanism. The structure of “locations” was hard coded into the program because this aspect of the design was the responsibility of another partner in the Homestead project. This Cruise Browser prototype was constructed using the final version of the Generic Browser.

The Cruise Browser structure consists of a hierarchy of “locations” as shown in the diagram above. Following an introductory video the user is given a choice of ways in which he might like to choose his cruise holiday.

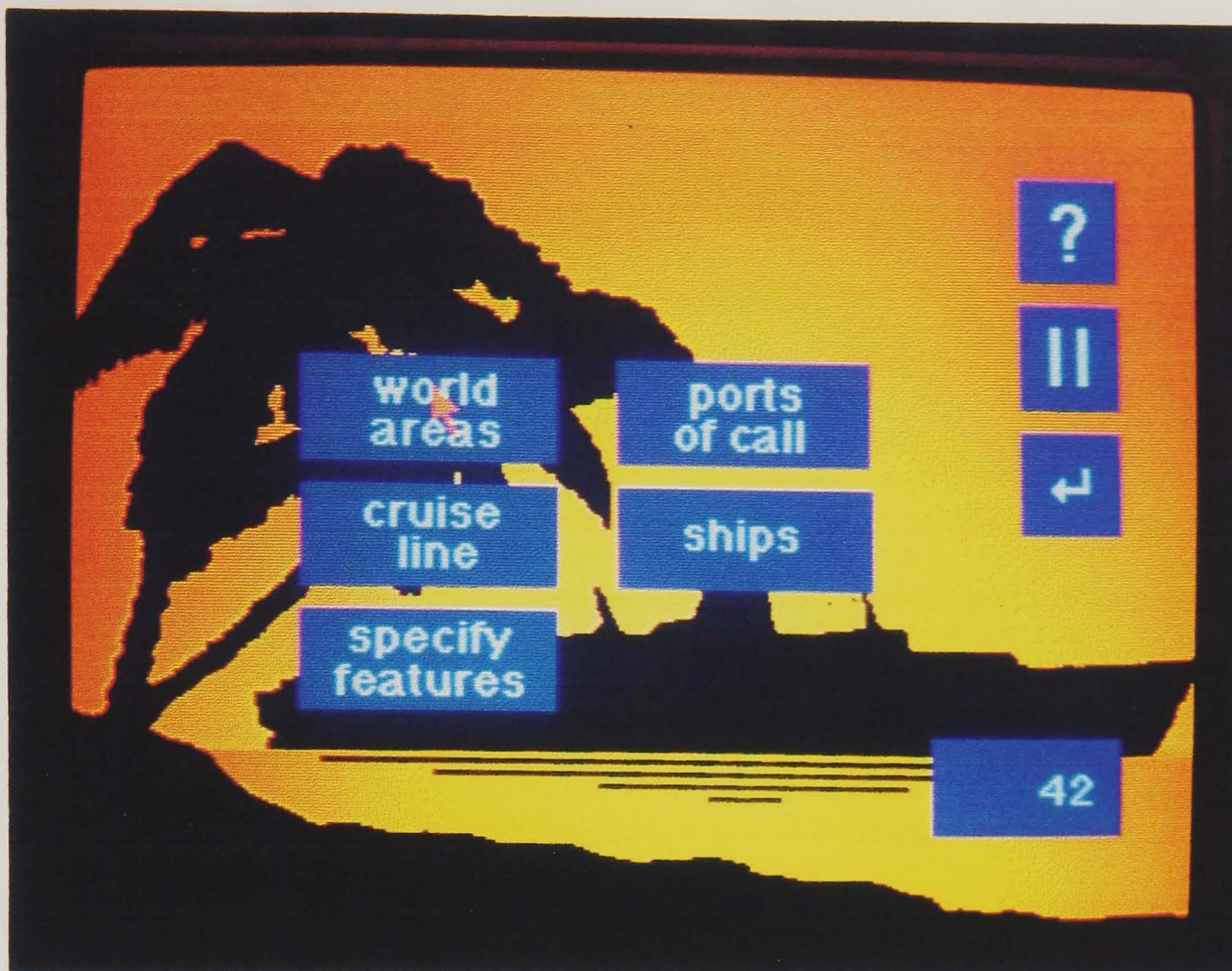


Figure 72. Cruise Browser main menu.

He can select to search for cruises with particular characteristics or browse the cruise directory in a number of different ways. He can select the area of the world in which he wants to cruise. The system then allows either all itineraries in that area of the world to be browsed or offers a list of all the ports of call in that area. The user can then select ports of interest and the system then finds all the cruise itineraries which pass through those ports. This search uses a best match process. The user then selects his preferred itinerary in a browser and makes his booking. Alternatively the user can select a port and the area of the world is found for him.

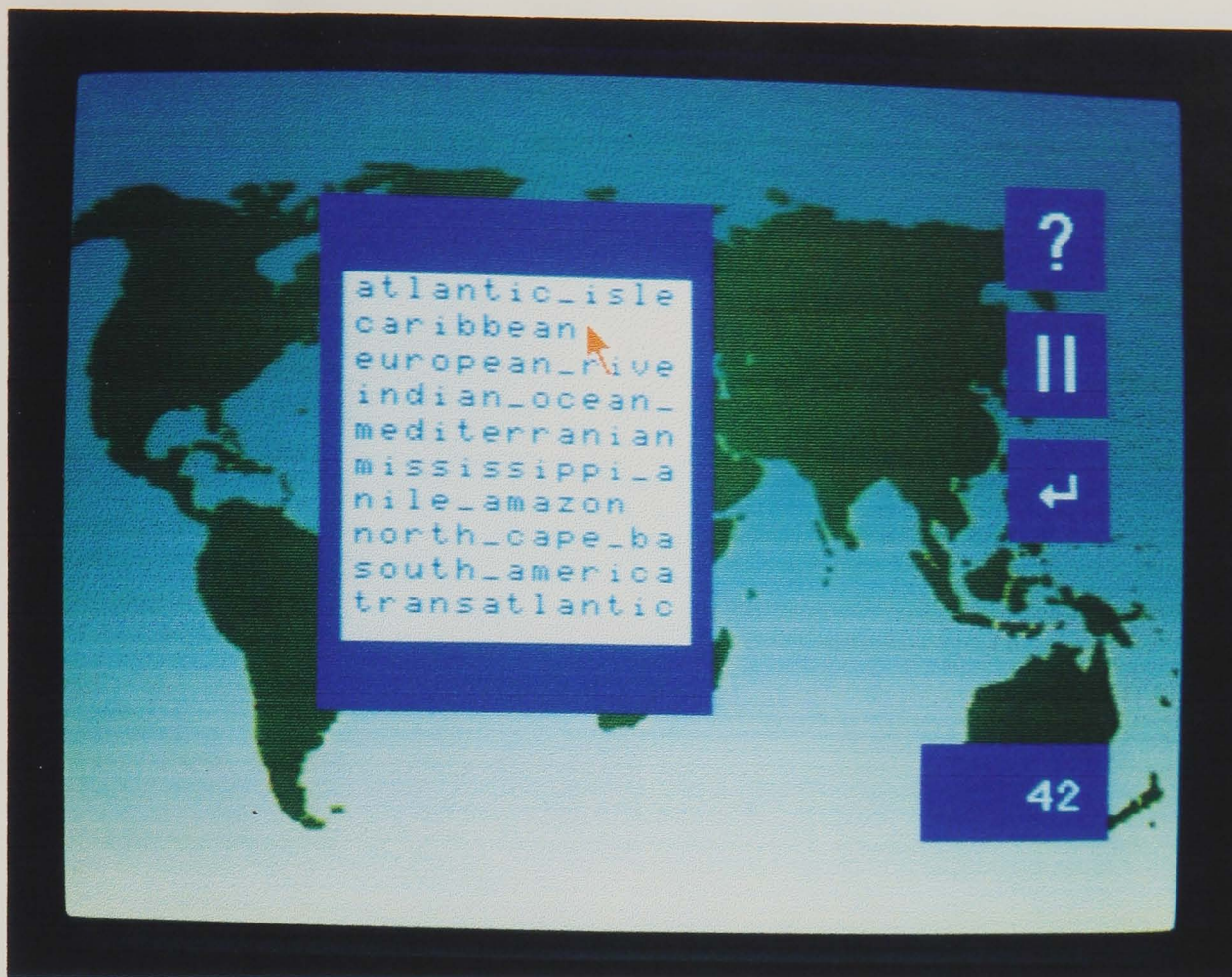


Figure 73. Area of the World menu.

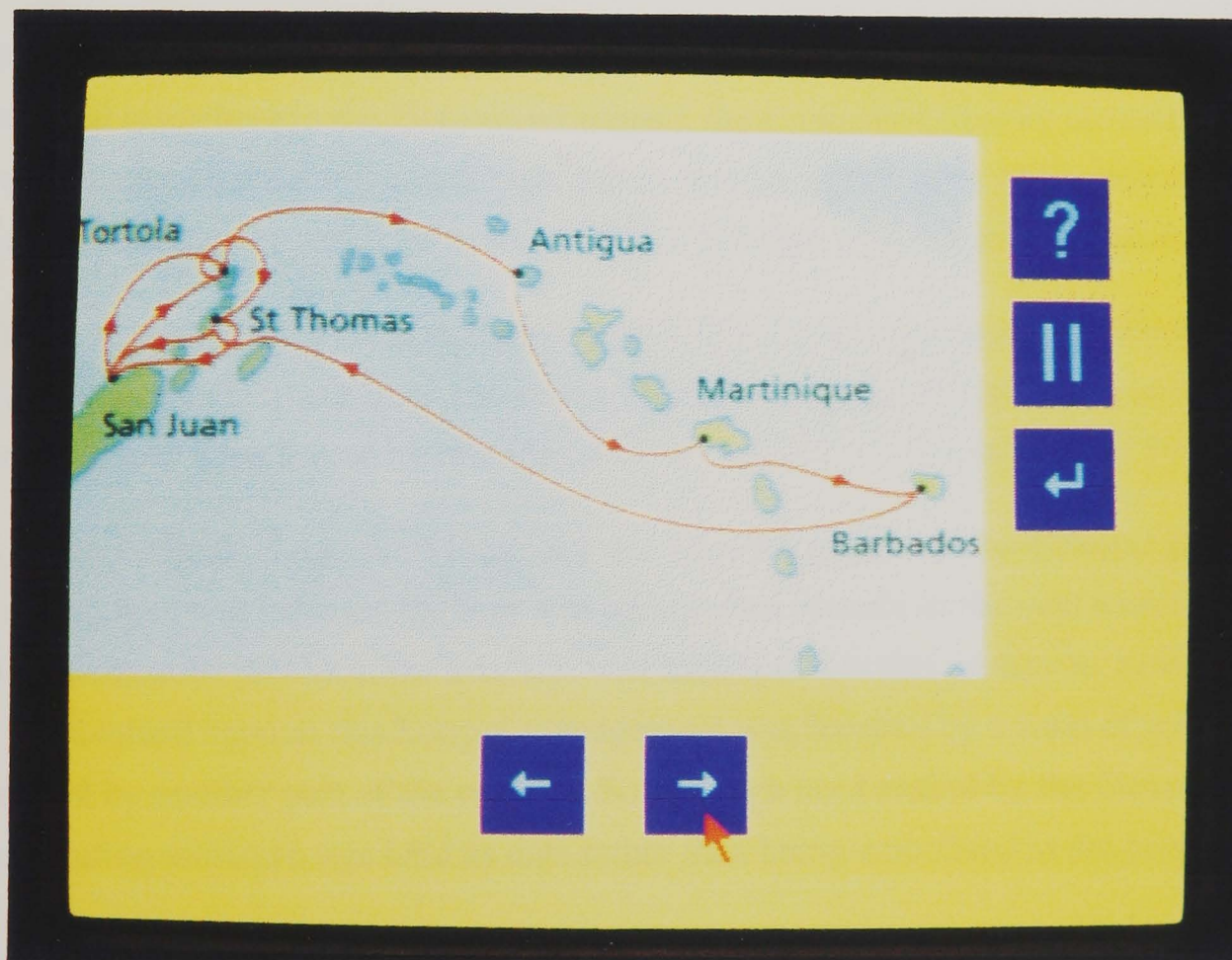


Figure 74. Itinerary browser.

The user may prefer to choose his cruise based on a particular cruise line or ship and these are both supported; the user either selecting a cruise line followed by a ship from that line or a ship from a list of all the ships. He is then presented with a browser of the itineraries being sailed by that ship, makes his choice and books his cruise.



Figure 75. Cruise Line menu.

In this prototype, which served as a demonstrator prior to a final implementation which was constructed elsewhere in the Homestead project, the choice of cruise by characteristic was not implemented because of a lack of interface designs at that stage of the project. However the search mechanism was used in the choice of itinerary by ports of call option.

The basic data structures were straightforward. Frames were created for areas of the world, ports of call, cruise lines and cruise ships. Each had pointers to assets where appropriate and were linked by relations such as "has ships", "location" and "visits".

Navigation between the various locations was handled outside of the Generic Browser. When a hotspot was selected the user was transported directly to the next location. As already mentioned this location structure was hard coded in this prototype, however in the final Homestead version this hotspot management was handled by a suite of software developed by another part of the project. So although not handled by the Generic Browser the navigation structure was held as data rather than being embedded in the program code.

This prototype highlighted a missing type of search mechanism - that which allows a function to be used to compare the query with data items. An example was how to find cruises cheaper than a user specified amount. Although the cruises could be categorised by price and the user's request mapped onto a price category it would have been better to be able to translate the request directly into a query.

It is useful to compare the Cruise Browser with the Ski Browser. Both have essentially the same requirements of search then browse. However the Ski Browser has a very simple interface and is basically a browsing application with the facility to change what is being browsed by performing a search. All navigation is within the data space. By contrast the Cruise Browser has a more sophisticated interface and requires the user to navigate around a series of screens rather than having a continuously available menu as in the Ski Browser. This additional navigation is outside the data space and highlights the potential problems if the interface is completely separated from the browsing functionality. For instance the interface needs some form of history mechanism to allow backtracking through the network of screens with the browser offering a similar facility for the data. This suggested the linkage between the browser and the interface described more fully in the previous chapter.

Work on interface design for the Cruise Browser showed how difficult it is to present a search facility within a consumer application targeted at a platform without a keyboard. For the limited freedom to choose search characteristics in this application a solution was found but more general search facilities would probably need some form of text input. Browsing on the other hand seems completely natural in this application. There is a great deal of research currently addressing the interface problems of providing search facilities to consumer-type users. One area of interest is the use of agents which do searches on behalf of the user. This may well be the way that interface design for search progresses. However the underlying search facilities and their link with browsing remain important whether it is the user who controls them directly or via an agent or if the search carried out entirely by the system.

The Cruise Browser was built with the final version of the Generic Browser and as such demonstrates the ability of the Library to support the development of a real application. Aside from reservations on the performance of the search component of the Generic Browser no modifications to the specification were generated during the development of this application and its subsequent integration into the Homestead demonstrator.

9.2.6 Summary of modifications to the Generic Browser Specification

Chapter 3 described the prototyping work as part of a feedback loop which led to the refinement of the Generic Browser specification. The prototyping work contributed two significant additions to the specification:

The first modifications to the specification surround the addition of the path as a browsable object. The original idea for creating browsable subsets had been the use of relations to tie the objects together. The early prototyping showed the flaws in this scheme and the path concept was developed. As described in chapter 7 the path has become the primary browsable object in the Generic Browser as it was also used for recording marked items.

The need for the inclusion of search into the specification came partly from the study of browsing and partly from the requirements for the Ski Browser. This application had a search mechanism added “on top of” the Generic Browser and provided the insight required to use marking as the means to integrate search. This too became part of the specification.

The Cruise Browser demonstrated the need for function search as well as matching search.

9.2.7 Critique based on Prototypes

This succession of prototypes has served two purposes. Firstly it has tested each stage in the development of the Generic Browser in as realistic setting as was practical. Problems thrown up by the prototyping, whether in the conceptual design or simply bugs in the implementation have enabled following prototypes to progress the state of the Generic Browser art. Secondly this testing gives practical credence to the hypotheses advanced by this thesis. Namely that it is possible to produce consumer multimedia applications with a core set of functions. The data representation structures proposed here have been found quite capable of holding the required information. The integration of browsing and search has been demonstrated in realistic applications.

One observation that can be made looking back over all of the prototype development is that, in contrast with each application being written stand-alone each prototype has benefited from the work on previous ones. Techniques of how best to use the Generic Browser facilities are re-usable as are basic data structures. As hypothesised in section 6.10 this is a clear advantage of using this approach over existing methods saving time and effort for the author. A consequence of re-use of techniques and code fragments should also be greater reliability and indeed less time was spent debugging low level code in the construction of the later prototypes.

The disadvantages of this approach are firstly that new prototypes take longer to design and build than it would to evolve a single prototype. This must however be offset against the greater freedom to test different application designs which is more difficult if one application must be evolved. A second problem which is common to all multimedia prototyping is limited availability of assets. It takes time to scan pictures and record audio tracks and at the prototype stage only a modest set can be put together. This makes performance testing a problem particularly where a process is non-linear with time such as the search routines described in this thesis. The solution here would be to duplicate assets where appropriate and automatically generate large databases for testing purposes. The third problem encountered while building and testing the prototypes was the needs of the interface. These simple prototypes only test the tie-up between the Generic Browser and the interface to a limited extent. As this work has progressed the importance of a stronger

link has been realised. To explore this connection in more detail would require a greater interface design contribution in the prototyping phase.

9.3 Reimplementation of Existing Browsers

One of the problems with creating prototype applications to test the library has been that the amount of design in the test applications has been small. In any but the very simplest of applications the rôle of the designer becomes dominant. The structure of the application is dictated by the design rather than being a consequence of the data structure and the interface becomes very important. Though the prototypes described above have proved invaluable in the development cycle it was felt that it was necessary to demonstrate the generality of Generic Browser architecture and functionality showing that it is capable of supporting the structures present in existing multimedia applications. The approach taken has been to perform a paper study analysing the structure of two applications from the CD-i catalogue to see how these would be implemented using the Generic Browser. Both the data structuring aspects and the access functions have been considered. The principal difference between the prototypes described above and these real applications is one of scale. The prototypes have made do with relatively few assets (tens rather than hundreds). The real applications have hundreds of assets. The size of the navigable or searchable domain has increased accordingly. However the complexity is not significantly greater than the prototypes.

The two examples have been chosen to cover a wide range of Generic Browser functionality and to be as different from each other as possible. While both discs require basic browsing The World of Impressionism has the type of location structure highlighted in chapter 8 and the ACT College Search disc includes search.

9.3.1 World of Impressionism

This disc presents the art of the 19th Century as a sort of interactive documentary. It is basically composed of a hierarchy structure of menus the leaves of which are slideshows with accompanying narration and or music. The menu structure is as follows:

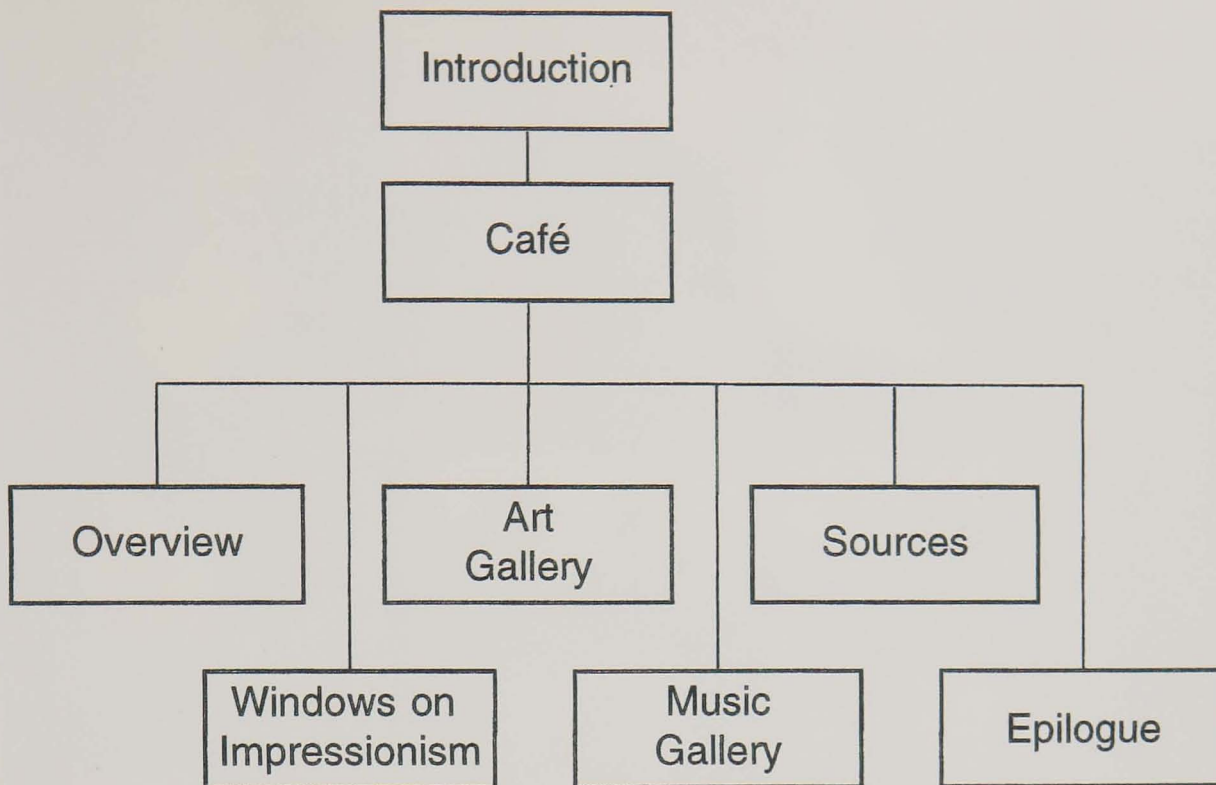


Figure 76. The World of Impressionism top level structure.

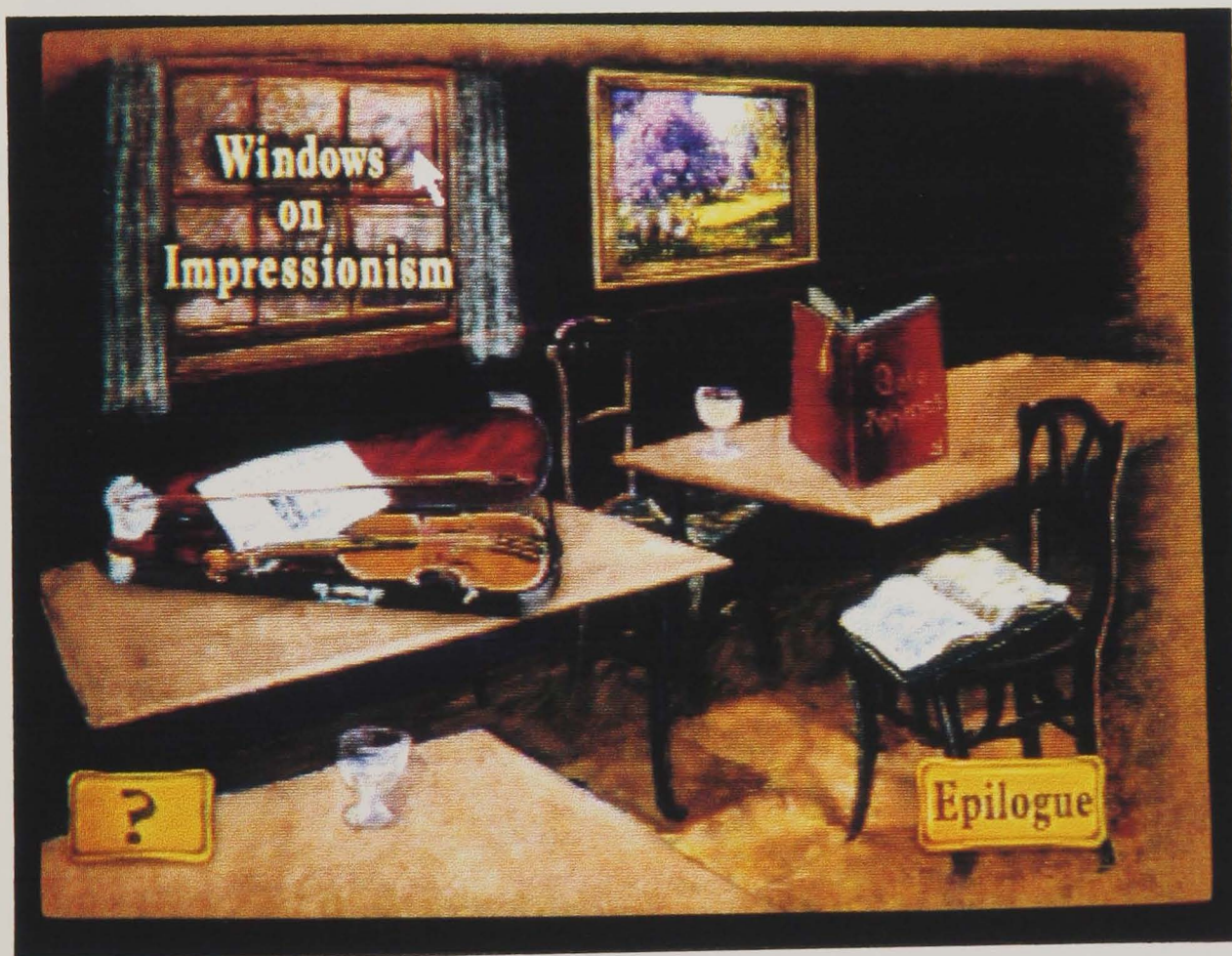


Figure 77. World of Impressionism main menu.

The Windows on Impressionism option leads to the following menu screen:



Figure 78. The Windows on Impressionism menu - the Paris option is centre of bottom row.

Each of the options in the window leads to another level. That for the Paris options is shown below:



Figure 79. The Paris map.

This map is preceded by an introductory sequence of stills with a narration and background music. Choosing a location on the map again leads to a narrated sequence of stills.



Figure 80. Selecting an option on the Paris map.

From a Generic Browser perspective this application can be constructed as a hierarchy of screens with paths (the slideshows) as the leaf nodes. For example:

```
(main_introduction
  (has_path intro_path)
  (leads_to cafe))

(cafe
  (has_path cafe_intro)
  (leads_to overview art_gallery sources
    windows_on_impressionism music_gallery
    epilogue)
  (back_to main_introduction))

(cafe_intro
  (is-a path)
  (path title1 title2 three_girls girl_CU
    girl_painting ...))
(soundtrack main_intro_soundtrk)
(display_time 3 seconds))
```

```
(three_girls
  (is-a picture)
  (pic_file three_girls.d))
```

The main challenges here are twofold. Firstly, in this application the user navigates a set of screens i.e. it is not the data but the interface structure that is being browsed. Secondly the slideshows have a narration that runs while a number of pictures are being shown.

The first problem is dealt with by using a frame representation for the screen structure, using relations to link the screens as shown above. As the user navigates he/she follows *leads_to* links (using the *gb_traverse_link* function which maintains the necessary history). At each change of screen the intro sequence (a path) is played, followed by the enabling of the menu hotspots (which are outside the Generic Browser representation, being embedded in the application).

The problem of the narration is eased by the fact that each image in the slideshows is shown for the same length of time with the same transitions between each one. Thus the soundtrack and picture duration information can be placed in the path frame. However there is no link between the soundtrack and the picture sequence, both just play from a synchronised start position.

A notable advantage of the frame representation used above is that assets which are used in several presentations (common in this disc) are stored once, pointers to them being stored in the paths. This contrasts with a possible alternative which would be to use video clips for the presentations. Here each use of a picture is stored separately (as part of each video presentation).

Once the basic application has been developed it would be straightforward to replace the World of Impressionism screens and assets with art from another era. The basic browsing application would remain the same since it traverses a structure held as data.

9.3.2 College Search Disc

The ACT College Search disc is a CD-i applications that makes use of search. The disc is aimed at school leavers in the United States who must choose their college. The search criteria are the courses and facilities offered by the various colleges. The structure of this disc is as follows:

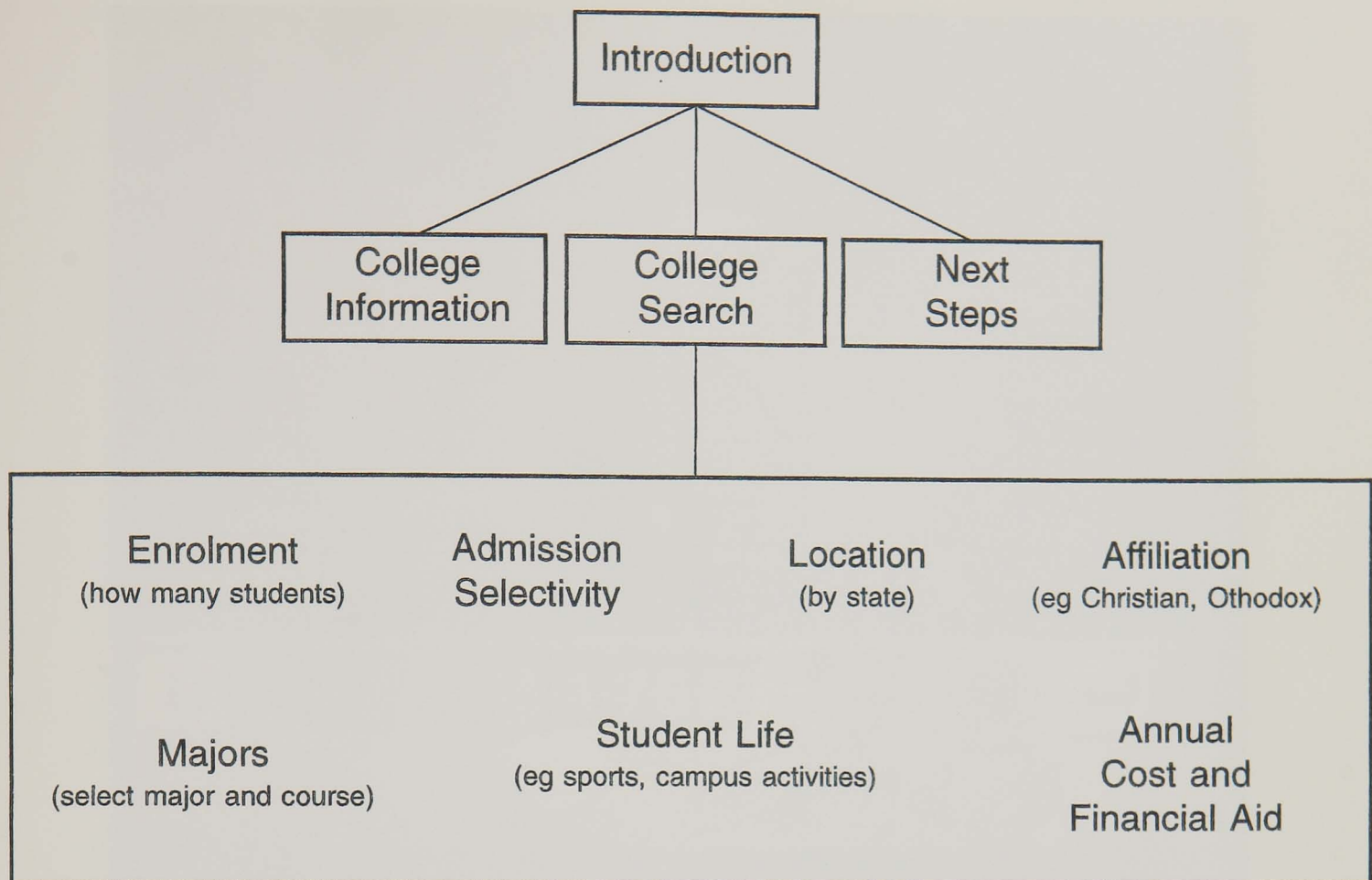


Figure 81. Structure of the College Browser disc.

The main part of the disc is a hierarchy of menus allowing the user to build a search specification using the seven main characteristics shown in the diagram above.



Figure 82. The Location option.

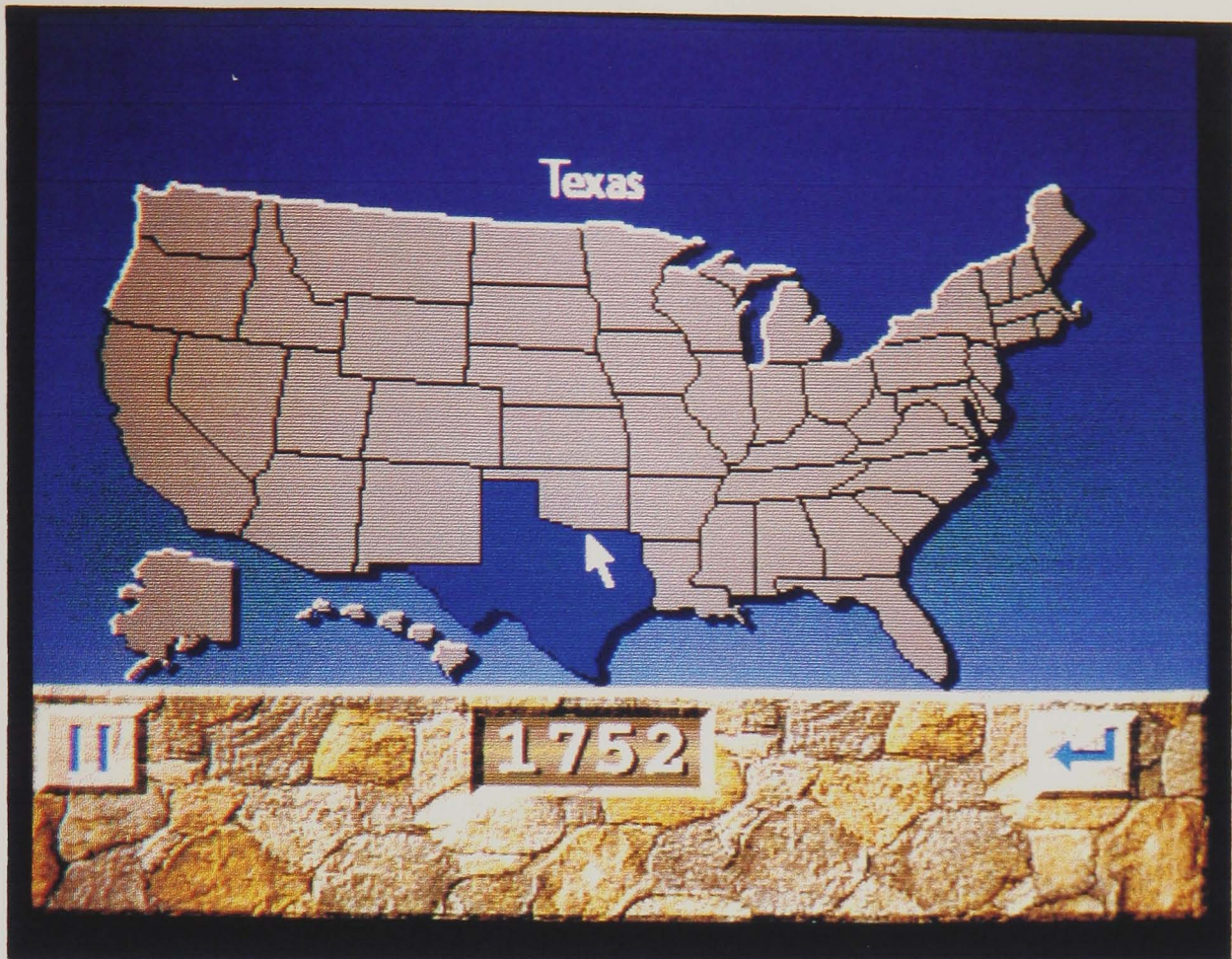


Figure 83. The State menu.



Figure 84. Selecting Texas - note the reduced number of universities (82).

A search is made and the results presented as a menu allowing browsing of the search hits. Individual colleges can be marked. This subset is also available for inspection at any time and is used to hold possible candidate colleges. When a college is selected to be examined

in further detail a slideshow presentation is followed by a menu of further information items such as admission information and sports offered.

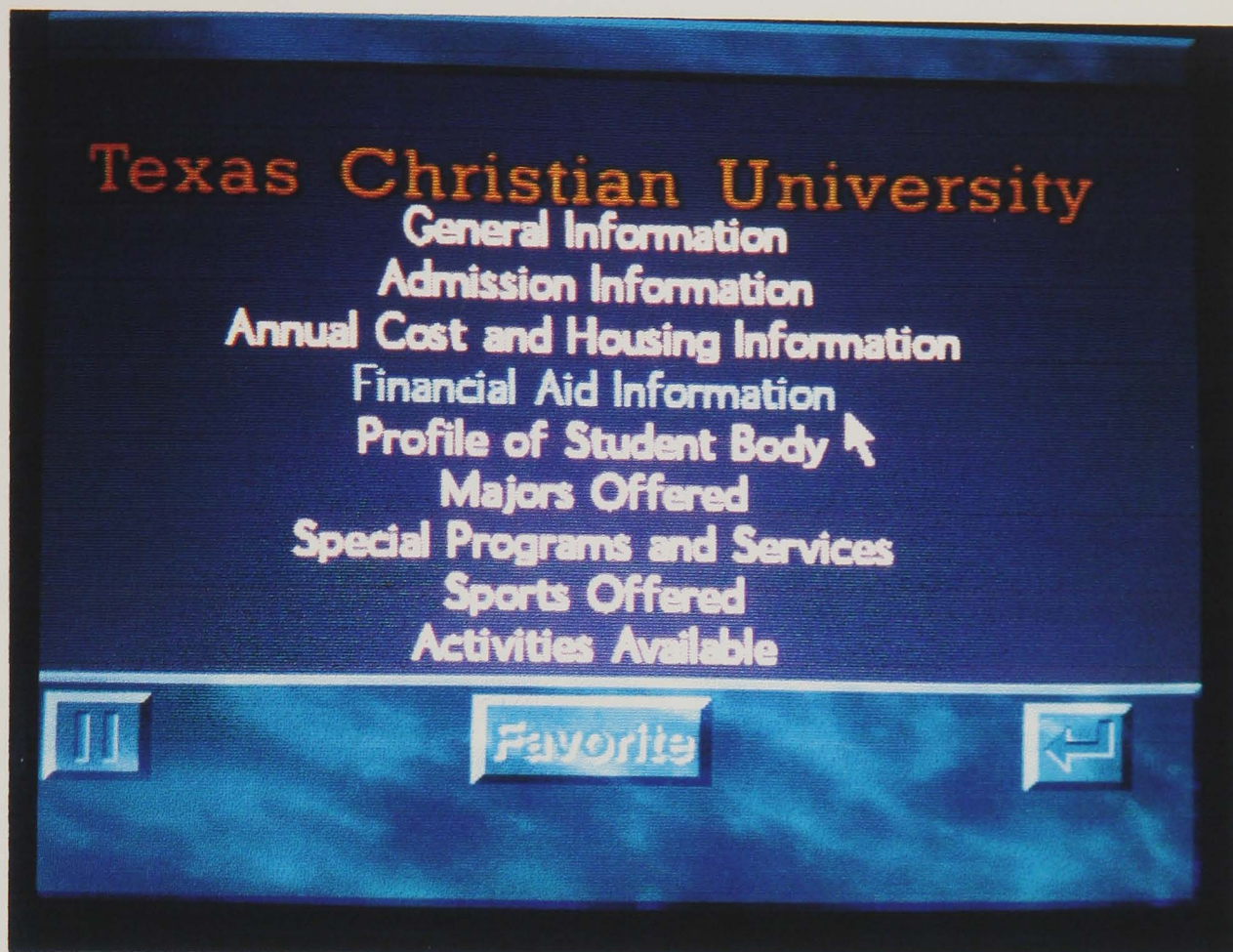


Figure 85. University Information menu.

In a Generic Browser implementation, each college would have a frame. This would contain slots for the characteristics such as courses offered and location, the extra information about the college and links to the "slideshows" that advertise each one. For example:

```
(MIT
  (instance college)
  (majors_offered food_sciences architecture
                  city_planning ...)
  (sports_offered baseball basketball golf ...)
  (ACT code 1858)
  (address "77 Massachusetts Avenue, Room 3-108,"
           "Cambridge, MA 02139")
  (admissions_office_phone "(617) 253-4791"))
  :
  :
  :
)
```

The search characteristics appear in the *majors_offered* and *sports_offered* slots while the extra information is in the *address* and *admissions_office_phone* slots.

The application would consist of a mechanism to present information extracted from the frames as menus and update the search query based on the user's selections. The user's favourite colleges form a marked set. Again, like the World of Impressionism example the hierarchical menu structure, though simple, could be represented as frames allowing a similar disc for example for the UK UCCA system to be produced easily.

In this example the search mechanism only uses exact matches. This could be duplicated by the Generic Browser or, greater freedom could be added by allowing a variable threshold which would prevent the "no hits" situation which is possible with the ACT disc.

9.4 Summary

The evaluation of the Generic Browser has taken on several forms. Basic functional testing has formed an integral part of the implementation of the Library. The construction of prototypes has allowed evaluation to form part of the refinement loop used in the development of the Generic Browser. This has prevented this development going off course and forced it to address real-world problems of application design. At the same time the prototypes demonstrate the main thrust of the thesis - the provision of integrated browse and search functions on a consumer platform. By doing a paper implementation of existing applications the possible pitfall of only evaluating possibly "Generic Browser friendly" prototypes has been avoided. Two examples have been shown. The next chapter reviews the work reported in this thesis and draws conclusions from it.

10 Summary & Conclusions

The main aim of this thesis was to show that a suitable basis for the construction of information-rich applications in the consumer domain could be provided by a core set of information access functions operating on a flexible data structure. The Generic Browser demonstrates that this objective has been realised as a working implementation on consumer hardware and tested with prototype applications.

The domain of consumer multimedia was chosen to limit the scope of the work to manageable proportions and to consider an area where little theoretical work has been done - the field being almost entirely technology led.

Having identified the key features of the processes of browsing and search, real world examples were also studied to augment the theoretical analysis. The resulting characterisation divides into two aspects. Firstly the data structures required for browsing and search have been considered and secondly the basic function set required to build information-rich consumer applications. A specification for a test implementation was developed and a function library built. In keeping with the desire to keep a firm grasp on the practicalities of the real world the implementation was performed on a current generation consumer multimedia platform, CD-i.

10.1 Contribution

This work has contributed to knowledge about browse and search interactions and the software systems that underpin them. It has done so by developing an integrated model that views browse and search as two extremes of the information access continuum. It further shows that the ratio of browse and search in a particular application can vary as the interaction proceeds. This means that an application does not have to be categorised as either browse or search but can be a dynamic combination of the two thus alleviating the problem of classifying multimedia applications that exhibit both browsing and search behaviour.

The theoretical basis for this model, distilled from the relevant literature, has been given a grounding in the real world by the results of a study of existing browsing applications. The characteristics derived from this study having been incorporated in the model. In particular the basic simplicity of the browsing process has allowed the specification of a core set of functionality.

This has been implemented as a 'C' function library running on the CD-i player. Implementation on a real consumer multimedia platform has been crucial to this work as it demonstrates that this approach is viable on this type of system within the constraints that it

imposes. The minimalist approach taken for the function library implements only what is strictly necessary to achieve the functionality that is actually of use to an application designer. A small, efficient kernel still looks attractive even when the platform is more capable as would be the case if the Generic Browser Library were to be used on existing PC or Macintosh machines, or if it were used in a server on a networked multimedia system.

The frame-based data model that complements the function library has been chosen to be flexible enough to cope with the requirements of representing multimedia data and also the principle browsable structure of the Generic Browser, namely paths. The requirement for a "sequence" object or path to complement the node-link data representation commonly found in hypermedia applications has been shown and its realisation in the Generic Browser demonstrated. Indeed the path structure has also been used within the Generic Browser itself as the representation of marked subsets.

These marked subsets have been developed in the Generic Browser from a simple facility for tagging nodes as the information space is traversed into a critical component of the browsing system. Apart from providing very flexible marking facilities for the user, marking is used as the mechanism for representing the results of a search. By this device search hits are immediately available to be browsed and browsing and search are effectively integrated.

10.2 Conclusions

To conclude this thesis it is necessary to reflect on what has gone before, to step back and consider what has been achieved and to question the decisions made during the work in the light of their later effects.

The choice of the consumer domain was a key decision early in this work. Though clearly influenced by the fact that this work was carried out in a commercial research laboratory this would have been a totally inadequate justification on its own. However, examination of the consumer domain revealed it to be under researched. It is currently a technology-led field with a high turnover of ideas. These ideas are not generally based on a bedrock of research and are tested by releasing them onto the market, the rapid rate of new software releases masking those applications that fail. By targeting this research at this domain it is hoped that it has been possible to redress this balance to a small degree. However it is perhaps unrealistic to expect to see change in the way that browse and search applications are constructed during the lifetime of this PhD project because of the time that it takes for research to be absorbed into industry practice.

The biggest hurdle is how to get the results of such research taken up. The commercial multimedia industry is composed of many, often small, companies where a combination of designers and programmers produce one-off or short series of applications. It is unlikely that the existence of research such as that documented here will cause an immediate change of working practice. It has become clear during the course of this research that the use of application development tools such as MacroMedia Director and Apple Media Tool is the direction that authoring for multimedia applications is going, particularly since these tools are already in use on PC and Macintosh platforms. These tools raise the level of the programming task, even allowing non-programmers to develop applications almost unaided. Adding functionality to such tools seems the most likely route for the infiltration of Generic Browser techniques into mainstream application development.

This was not foreseen at the start of this work when it was anticipated that the Generic Browser would become part of the application programmer's armoury in the way that GUI toolkits have. The Generic Browser's core function set without bells and whistles is ideally suited to integration within an environment such as MacroMedia Director demonstrating that this approach was the right one for this work.

Considering the choice of CD-i for the implementation work, again the Philips connection played a part but also when this work was started CD-i was the only suitable consumer platform. The trend for the number of PCs in the home to reach levels where a PC should be regarded as a consumer platform was not anticipated. If this work were being carried out in today's environment PCs should be considered as the target platform for this type of research. However it has still to be proved that PCs in the home are really consumer machines rather than work machines used to play games and use edutainment CD-ROMs.

This highlights one of the problems of researching in an environment that is evolving fast as the technology changes. To some extent this causes the goal posts to move as the research is done. Another example is the rise in importance of video. When this work was started there was no full-screen, full motion video available on Macintosh or PC let alone home entertainment systems so video was not considered as important as it undoubtedly is now.

The use of frames as the data representation scheme has been shown to be valid from the point of view that frames are clearly flexible enough to represent multimedia information and the path structures central to the Generic Browser. That very generality and in particular the typeless data used by the frame system used here does impose a performance penalty. Experience with the prototypes showed that this is only significant where large numbers of frames must be processed such as in the search routines. This problem is addressed in detail in the next chapter. The use of frames to store hypertext as opposed to

hypermedia would not be ideal. The problem stems from words in a text not being primary objects in the Generic Browser, rather a fragment of text is treated as an asset. This is similar to the way in which the video is dealt with.

The development of the path structure from just a sequence for display purposes to a primary browsable object is important. The process of authoring imposes structures onto a collection of data nodes and it was found that just using links between the nodes was not flexible enough. The path answers this problem by having the structure stored in its own right. This allows multiple structures without interference between them. This is similar to the concept of views used in the database field. The use of paths for marked subsets and histories has allowed a single unified representation to be used for all structure imposed on the basic data nodes (with the exception of the use of relations as hyperlinks which are still possible). Further research into more complex path structures than the linear ones implemented in the final version of the Generic Browser is suggested in the next chapter.

A goal of this work has been to find a way to integrate browsing and search and this has been accomplished through the use of marked subsets. The Generic Browser methodology also uses the same representation for the data for both browsing and search. If completely separate subsystems are used for browsing and search the different data representations would have to be linked in some way and synchronisation maintained.

An important question arises out of this choice of solution; what compromises, if any, have been made to either the browse or search mechanisms to enable this integration to take place? Because examination of consumer applications shows that browsing interactions dominate access to the information (and is preferred to search) the search component has been tailored to fit within the framework developed for browsing.

This type of integration of the search process has placed some constraints upon it. In particular search systems are usually optimised, often by the provision of indexes. These, once calculated, speed up the search process. Such optimisations have not been possible with the dynamic data structures used by the Generic Browser as indexes only work with static data. However much of the data in consumer applications *is* static. If static data were treated separately within the frame-based system the search mechanism could be optimised for accessing this static data (using indexes perhaps).

A second aspect of the tailoring of the search process is the choice of search algorithm. In the consumer domain inexact matching is preferred as it allows the application to find the closest matching items rather, for example, returning no hits at all if the search query is too strict. The chosen use of marking to integrate search does, however, mean that alternative or additional search mechanisms are easily added to the Generic Browser. What has been

demonstrated is the mechanism for integration with an example search function. Other search functions would use the same marking mechanism.

The prototyping of the library has been complemented by the building of prototype applications in a feedback loop where the results of the application building process have been used to refine the library. This strategy for working has been found to be very beneficial as it brings shortcomings and problems to light as soon as possible. However the bulk of the testing of the Generic Browser has been carried out by the author. Though this was unavoidable it would seem desirable to extend the testing by having others define and perform tests. This would allow verification that the functionality that has been provided is useful and highlight those areas where changes need to be made.

As observed in the previous chapter the prototypes demonstrated that considerable technique and structure re-use was possible from one application to the next. This is important as it demonstrates that the Generic Browser has indeed shown that it can facilitate the authoring of multimedia applications.

Standing back from this work the most significant shortcoming is the performance issue. The final version of the Generic Browser Library is still a research prototype with performance taking second place to demonstrating the concepts of the Generic Browser model. To be used in real applications performance would have to be improved.

The Generic Browser library allows authors of information-rich applications to avoid developing the browsing and search facilities from scratch thereby reducing code development and debugging time. In its current form it is still a research prototype but since it is already implemented on a representative multimedia platform all that remains to be done to make it generally useful is to refine the efficiency of the implementation. This and opportunities for further research arising from this work are discussed in the next chapter.

11 Further Work

Any piece of work such as this has to draw to a close from the perspective of what should be written-up as a thesis so that it can be reviewed and evaluated and for others to benefit from what has been done. This chapter considers in what ways this work might be continued and built upon.

11.1 Further Work on The Generic Browser

This section looks at improvements which could be made to the Generic Browser in the relatively short-term, in particular those issues which would allow it to taken up as part of an application development tool.

11.1.1 Improving Performance

In terms of making the results of this research available to the application development community the issue of performance would have to be addressed. Certainly for CD-i, the prototype library, and in particular the search process, is not fast enough when handling large amounts of data.

The most effective way to address these performance issues is to re-implement the Generic Browser functionality as part of the frame-based system layer, in this case Cframes. This would be advantageous because the frame data structures would be accessed directly rather than via access functions.

The Generic Browser is constructed using Cframes functions for data access. A browsing application therefore accesses information through two layers, the Generic Browser and Cframes. Because the Generic Browser also uses Cframes for its internal storage requirements inefficiencies also arise because of the representation scheme used by Cframes.

Cframes represents all data in its frames as character strings in a table structure. This confers the important advantage from the user's perspective that there are no data types in Cframes, all data is treated the same and different types can co-reside as values in particular slots. From the Generic Browser perspective, though, this very flexibility is a disadvantage since it carries an overhead. Cframes must determine the type of the data item and, if necessary convert from the character representation into the appropriate type for the required operation.

Take for example the use of a counter by the Generic Browser. Since this is a numeric it is clearly very inefficient to store this as a character string since every time it is used it must be converted to and from its numeric representation for the purposes of manipulation.

Similarly the access method used for frames where a symbol table converts from the character string, the name of the frame, into the address of that frame's information in the table. From the Generic Browser perspective it would be more efficient to work with this lower level representation - the frame (or slot) address and only convert back to the character form when absolutely necessary.

For these reasons it would be very desirable to integrate the Generic Browser into Cframes, allowing all operations to be carried out internally with the most efficient representations. As an indication of the gains to be had by this integration Project member Martin Shiels has added the Generic Browser search mechanism to Cframes leading to an order of magnitude speed increase (though no marking took place).

Further research into the types of search algorithms and functionality is necessary to broaden the scope of the search offered by the Generic Browser. The basic mechanism of using marking to integrate search with browsing has been shown to be effective. The challenge is to improve the efficiency of search within the Generic Browser data structures and to examine different types of search.

Additional speed improvements may be achieved by differentiating between static and dynamic data in the frame system. This would then allow the static portion of the database to be indexed to improve search efficiency. This optimisation would, of course, only work for those situations where the data in the final application was fixed, on a CD-ROM or CD-i disc for example.

A more drastic alternative would be to choose to implement the Generic Browser philosophy in, say, C++. Rather than implement a frame system to hold the data, the browsing mechanism could be adapted to browse C++ objects.

11.1.2 Integration with an Authoring Tool

As this work has progressed it has become increasingly clear that, particularly with the increasing importance of PCs for multimedia applications, the integration of Generic Browser functionality within the authoring environments preferred for those platforms would offer an excellent opportunity to make the Generic Browser available to a wider audience. The lack of Generic Browser functionality within such tools as MacroMedia Director and Apple Media Tool has already been noted and, for using either of these tools to construct information-rich applications, this omission would have to be circumvented.

Extending either of these tools would be relatively straightforward. In the case of MacroMedia Director there is a documented mechanism for calling external functions (written in 'C' for example) called XFNs. To add Generic Browser functionality the Library would have to be modified so that the individual functions conformed to the XFN

standard. The Cframes library would have to be similarly modified (to allow the manipulation of the data held in the frames). To use the Generic Browser in MacroMedia Director, the XFN's would simply be called from LINGO scripts.

11.2 Other Research Issues

As well as refining the existing Generic Browser, which should be mainly a development process, research issues have been raised by the Generic Browser which it has not been possible to address within the scope of this thesis.

11.2.1 Branching Paths

Linear paths have been shown to be an extremely useful and flexible structure. Some thought has also been given to a representation for branching paths. Such structures are now receiving attention within the area of multimedia content research, specifically as a representation for multipath story structures. These are already used in interactive fiction and are one possible structure for interactive movies. This latter avenue is being explored by a new project at PRL, the Hands-on Movies Project of which the author is a member. Development of branching paths within the Generic Browser would allow it to be used to support the navigation of a rich variety of non-linear story structures. Detailed exploration of branching paths would be one direction of further research based on the Generic Browser.

11.2.2 Video

The second area for new research based around the Generic Browser is that of the representation and integration of video. Video is increasingly becoming important in the consumer multimedia arena particularly as MPEG digital video playback becomes prevalent. Very soon the use of video within multimedia applications will become ubiquitous and the Generic Browser will need to be enhanced to integrate it. Research issues such as browsing time-based material (video and audio) and the synchronisation of playback of a number of pieces of video and audio give some indication of the new areas that will need to be addressed. The use of a flexible representation scheme that will allow content description would seem necessary and the research done for the Generic Browser provides a sound starting point. In information-rich applications video will be only a part and there will be a need to integrate other multimedia objects and provide navigational structures for these data. Again the research presented in this thesis provides the basis for information browsing.

11.3 Agents

In the tv programme "Hyperland" [Adams 1990] Douglas Adams presented his view of what the tv of the future would be like. Adams believes that the tv of the future will be interactive and include agents that allow the viewer (or, more accurately, user) to tell the system what he is interested in. The system then offers a variety of information items that conform with this stated interest. The user selects one and views the information. At any time the system agent can be summoned to provide assistance or to change the area of interest. Other information agents (depicted as video icons or "picons") pop up and try to attract the user's attention when they have information that may be of interest to him.

Though Adams was speculating on the future, his ideas are close to what is possible using the multimedia platforms now, or soon to be, available. At CHI '91⁴¹, Apple showed the latest prototype to come out of its GUIDES project. One of the aims of this work is "to integrate browsing and searching activities under one interface umbrella" [Don et al 1991]. The CHI demonstration combined this browsing/searching activity with interface agents (guides) which provided navigational assistance by suggesting possible next topics for the user. Each guide has a different point of view and thus offers selections with different slants. With the guides portrayed as video icons this system is very like that proposed in Hyperland.

Such agents need to search the database and flag items of interest for the user to review at her leisure. An interesting development of the Generic Browser architecture would be the addition of agents. In a sense the search mechanism already in place is a very simple agent and indeed the vector space search mechanism underlies Apple's guides. Further research could be pursued in the direction of the agent mechanism itself. How the user's interests can be communicated to and represented by the agent for example. The frame representation scheme used in the Generic Browser would seem to have the flexibility for supporting research into this area.

11.4 World Wide Web

Since this research was started the World Wide Web has become a consumer product thanks to the growth of PCs in the home and the reduction in price of connection hardware (modems) and the increase in the number of service providers who offer Internet access via phone connections.

⁴¹CHI '91 - The Computer-Human Interaction conference held in New Orleans, Louisiana, April 27th to May 2nd 1991.

The WWW offers a rich area for exploring browsing and search. Information nodes are held in a network data structure with each node having a unique address or URL. Nodes contain a representation of their content written in a text mark-up language called HTML. When the user browses a node the browsing application interprets the HTML and displays the text, links and embedded pictures for the user. HTML offers minimal display formatting options with most characteristics such as font, text size, highlighting etc. dependent on the user's browser application. Existing browsing tools such as NCSA Mosaic and Netscape are fairly primitive offering only the ability to follow hyperlinks with search only available via special database sites. Limited marking is available through the use of bookmarks which record URLs. Video on the Web is treated as it is in the Generic Browser i.e. as indivisible chunks which are "played" by the browser application.

One avenue of research would be to consider how the facilities offered in the Generic Browser could be offered on the Web. For example, paths could be used to form structures of URLs. This would allow guided tours around the Web.

From the Generic Browser perspective HTML techniques could allow hypertext support within the Generic Browser. Rather than URLs embedded in the text, frame names could be used instead. A modified interpreter could then present the text as on the Web and when the user clicked on the hot words another frame of text could be displayed.

Sun Microsystem's newly released "Java" programming language [Sun 1995] offers the possibility of "active" documents i.e. ones which contain code that will be executed on the user's machine rather than just relying on the user's browser to provide access to the information. Perhaps the Generic Browser facilities could be implemented in the "Java" programming language. This would allow self browsable documents which could contain paths and search facilities.

11.5 Summary

The potential for further research work on and stemming from the Generic Browser has been shown in this chapter. The Generic Browser forms a foundation onto which more complex models can be built, to integrate video for example. The world of multimedia and in particular that part which serves the consumer market will not stand still and, if it is to remain relevant and lead the development process, neither can the research activity. The consumer market is a challenging arena because it has a strong tendency to be led by the technology. Research must continue to strive to redress this balance.

References

[Adams 1990]

Adams, D. **Hyperland**. Written by Douglas Adams, produced by Max Whitby, BBC tv.

[Alesandrini 1984]

Alesandrini, K.L. **Pictures and Adult Learning**. Instructional Science 13, 1984 pp. 63-77.

[Andersson 1991]

Andersson, J. **A Design Proposal for a Hypermedia Abstract Machine**. Multimedia - Systems, Interaction and Applications. Kjell Dahl L. (ed), Springer-Verlag 1991 pp. 146-152.

[André & Rist 1990]

André, E. and Rist, T. **Towards a Plan-Based Synthesis of Illustrated Documents**. Proceedings 9th European Conference on Artificial Intelligence, Stockholm, Sweden, August 6-10 1990 pp. 25-30.

[Apple 1993a]

Apple Computer. **Inside Macintosh: Quicktime**. Addison-Wesley 1993.

[Apple 1993b]

Apple Computer. **Inside Macintosh: Quicktime Components**. Addison-Wesley 1993.

[Apple 1995]

Apple Computer. **QuickTime Product Family**. Apple Computer, Inc. 1995.

[Brody 1984]

Brody, P.J. **In Search of Instructional Utility: A Function-Based Approach to Pictorial Research**. Instructional Science 13, 1984 pp. 47-61.

[Bush 1945]

Bush, V. **As We May Think**. The Atlantic Monthly, vol 176, 1945, pp. 101-108.

[Campbell & Goodman 1988]

Campbell, B. and Goodman, J.M. **HAM: A General Purpose Hypertext Abstract Machine.** Communications of the ACM 31(7), July 1988 pp. 856-861.

[Cannan & Otten 1993]

Cannan, S.J. and Otten, G.A.M. **SQL - The Standard Handbook.** McGraw-Hill Book Company 1993.

[Canter et al 1985]

Canter, D., Rivers, R. and Storrs, G. **Characterizing User Navigation Through Complex Data Structures.** Behaviour and Information Technology, 4(2) 1985 pp. 93-102.

[Card et al 1983]

Card, S.K., Moran, T.P. and Newell, A. **The Psychology of Human Computer Interaction.** Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[Carmel et al 1992]

Carmel, E., Crawford, S. and Chen H. **Browsing in Hypertext: A Cognitive Study.** IEEE Transactions on Systems, Man and Cybernetics, 22(5) September/October 1992 pp. 865-884.

[Carrigan 1991]

Carrigan, T. **Fighting for Lounge Control.** Multimedia 4, May/June 1991.

[Cole 1991a]

Cole, R.S. **Modifications to XLISP to Allow Calling from C.** Philips Research Laboratories AI Group Informal Note AI/GEN/0043, March 1991.

[Cole 1991b]

Cole, R.S. **XKC: A Frame-Based System for XLISP.** Philips Research Laboratories AI Group Informal Note AI/GEN/0048, March 1991.

[Cole 1991c]

Cole, R.S. **The CD-i Film Browser.** Philips Research Laboratories Technical Note 3063, 1991.

[Conklin 1987]

Conklin, J. **Hypertext: An Introduction and Survey.** IEEE Computer 20(9) September 1987 pp. 17-41.

[Cortese 1995]

Cortese, A. **Once Again, Software is Seething.** Business Week, January 9, 1995 p. 46.

[Cove & Walsh 1988]

Cove, J.F. and Walsh, B.C. **Online Text Retrieval via Browsing.** Information Processing and Management, 24(1) 1988 pp. 31-37.

[Crawford 1993]

Crawford, C. **Connective Interactivity.** The Journal of Computer Game Design, 6(6) August 1993 pp. 2-11.

[Davenport et al 1991]

Davenport, G., Aguierre Smith, T. and Pinciver N. **Cinematic Primitives for multimedia.** IEEE Computer Graphics and Applications, July 1991 pp. 67-73.

[Davis 1993]

Davis, M. **Media Streams: An Iconic Visual Language for Video Annotation.** Paper presented at IEEE/CS Symposium on Visual Languages, Bergen, Norway, August 1993.

[DeRose 1989]

DeRose, S.J. **Expanding the Notion of Links.** Hypertext '89 Proceedings, Pittsburgh, Pennsylvania, November 5-8 1989 pp. 249-257.

[Drasch 1989]

Drasch, F.J. **The Clisp Programming Environment.** Drasch Computer Software, Ashford, Connecticut, 1989.

[Don et al 1991]

Don A., Oren, T. and Laurel, B. **GUIDES 3.0.** Proceedings of CHI '91, New Orleans, Louisiana, April 27th - May 2nd, 1991 pp. 447-448.

[Duchastel 1990]

Duchastel, P.C. **Examining Cognitive Processing in Hypermedia Usage.** Hypermedia, 2(3) 1990 pp. 221-233.

[Duchastel & Waller 1979]

Duchastel, P. and Waller, R. **Pictorial Illustration in Instructional Texts.** Educational Technology, November 1979 pp. 20-25.

[Ellis 1989]

Ellis, D. **A Behavioural Approach to Information Retrieval System Design.** The Journal of Documentation, 45 (3) September 1989 pp. 171-212.

[Evans 1987]

Evans, A. **TELCLASS: a structural approach to TV classification.** Audiovisual Librarian, 13(4) 1987 pp. 215-216.

[Evans 1993]

Evans, R. **LogBoy and FilterGirl: Tools for Personalizable Movies.** Interactive Cinema Technical Note, MIT Media Laboratory, August 1993.

[Faloutsos et al 1990]

Faloutsos, C., Lee, R., Plaisant, C. and Shneiderman, B. **Incorporating String Search in a Hypertext System: User Interface and Signature File Design Issues.** Hypermedia, 2(3) 1990 pp. 183-200.

[Fox et al 1988]

Fox, E.A., Nunn, G.L. and Lee, W.C. **Coefficients for Combining Concept Classes in a Collection.** Proceedings of the 11th International Conference on Research & Development in Information Retrieval, Grenoble, France, June 13-15 1988 pp. 291-307.

[Garber & Grunes 1992]

Garber, S.R. and Grunes, M.B. **The Art of Search: A Study of Art Directors.** Proceedings of CHI '92, Monterey, California, May 3-7 1992 pp. 157-163.

[Gotel 1990]

Gotel, O.C.Z. **Reader Centred Conversion of Text Books into Hypertext.** MSc dissertation, Department of Computer Science, Queen Mary and Westfield College, London, 1990.

[Grimes 1975]

Grimes, J.E. **The Thread of Discourse.** Mouton 1975.

[Halasz 1988]

Halasz, F.G. **Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems.** Communications of the ACM, 31(7) July 1988 pp. 836-852.

[Halasz & Schwartz 1990]

Halasz, F. and Schwartz, M. **The Dexter Hypertext Reference Model.** Proceedings NIST Hypertext Standardisation Workshop, Gaithersburg, MD, January 16-18 1990 pp. 95-133.

[Hamakawa & Rekimoto 1993]

Hamakawa, R. and Rekimoto, J. **Object Composition and Playback Models for Handling Multimedia Data.** Proceedings of ACM Multimedia '93, Anaheim, California, August 1-6 1993 pp. 273-281.

[Hammond & Allinson 1988]

Hammond, N. and Allinson, L. **Travels Around A Learning Support Environment: Rambling , Orienteering or Touring?** Proceedings of CHI '88, Washington DC, May 15-19 1988 pp. 269-273.

[Hardman et al 1993a]

Hardman, L., van Rossum, G. and Bulterman, D.C.A. **Structured Multimedia Authoring.** Proceedings ACM Multimedia '93, Anaheim, California, 1-6 August 1993 pp. 283-289.

[Hardman et al 1993b]

Hardman L., Bulterman, D.C.A. and van Rossum, G. **The Amsterdam Hypermedia Model: Extending Hypertext to Support *Real* Multimedia.** Hypermedia, 5(1) 1993 pp. 47-69.

[Harrison 1985]

Harrison, H. **You Can Be The Stainless Steel Rat.** Grafton Books, 1985.

[Hobbs 1978]

Hobbs, J.R. **Why is Discourse Coherent?** Technical Note 176, SRI International, November 1978.

[Hooper Woolsey 1991]

Hooper Woolsey, K. **Multimedia Scouting.** IEEE Computer Graphics & Applications, July 1991 pp. 26-38.

[Hovy 1988]

Hovy, E.H. **Approaches to the Planning of Coherent Text.** Paper presented at the 4th International Workshop on Text Generation, Catalina Island, California, July 1988.

[ISO 1988]

ISO 9660:1988 Information processing - Volume and file structure of CD-ROM for information interchange.

[ISO 1993a]

ISO/IEC 11172-1:1993 Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 1: Systems.

[ISO 1993b]

ISO/IEC 11172-2:1993 Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 2: Video.

[ISO 1993c]

ISO/IEC 11172-3:1993 Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio.

[Jerke et al 1990]

Jerke, K.-H., Szabo, P., Lesch, A., Rößler, H., Schwab, T. and Herczeg, J. **Combining Hypermedia Browsing with Formal Queries.** Human-Computer Interaction - INTERACT '90, D. Diaper et al (eds), Elsevier Science Publishers B.V. 1990 pp. 593-598.

[Kehoe 1995]

Kehoe, L. **Four-fold rise in multimedia PC sales.** Financial Times 14th March 1995.

[Kretz & Colaitis 1992]

Kretz, F. and Colaitis, F. **Standardising Hypermedia Information Objects.** IEEE Communications Magazine, May 1992 pp. 60-70.

[Laurel 1991]

Laurel, B. **Computers as Theatre.** Addison-Wesley 1991.

[Laurel et al 1990]

Laurel, B., Oren, T. and Don, A. **Issues in Multimedia Interface Design: Media Integration and Interface Agents.** Proceedings of CHI '90, Seattle, Washington, April 1-5, 1990 pp. 133-139.

[Levie & Lentz 1982]

Levie, W.H. and Lentz, R. **Effects of Text Illustrations: A Review of Research.** Educational Communication & Technology Journal, 30(4) 1982 pp. 195-232.

[Liebscher & Marchionini 1988]

Liebscher, P. and Marchionini, G. **Browse and Analytical Search Strategies in a Full-Text CD-ROM Encyclopedia.** School Library Media Quarterly, Summer 1988 pp. 223-233.

[Little et al 1993]

Little, T., Ahanger, G., Folz, R., Gibbon, J., Reeve, F., Schellig, D. and Venkatesh, D. **A Digital On-Demand Video Service Supporting Content-Based Queries.** Proceedings of ACM Multimedia '93, Anaheim, California, 1-6 August 1993 pp. 427-436.

[Lloyd 1991]

Lloyd, P. **The Guitar Browser.** PRL IS Group Informal Note H91/58, December 1991.

[Lloyd 1993]

Lloyd, P. **Informative CD-I.** Philips Research Laboratories IS Group Informal Note H93/2, January 1993.

[Maguire 1991]

Maguire, J.G. **Interactive Magazines.** Computer Graphics World, August 1991 pp. 33-40.

[Mann & Thompson 1987]

Mann, W.C. and Thompson, S.A. **Rhetorical Structure Theory: A Theory of Text Organisation.** ISI Technical Report ISI/RS-87-190, Information Sciences Institute, University of Southern California, June 1987.

[Marchionini 1989]

Marchionini, G. **Information-Seeking Strategies of Novices Using a Full-Text Electronic Encyclopedia.** Journal of the American Society for Information Science, 40(1) January 1989 pp. 54-66.

[Marchionini & Shneiderman 1988]

Marchionini, G. and Shneiderman, B. **Finding Facts vs. Browsing Knowledge in Hypertext Systems.** IEEE Computer, January 1988 pp. 70-80.

[Marshall & Irish 1989]

Marshall, C.C. and Irish, P.M. **Guided Tours and On-Line Presentations: How Authors Make Existing Hypertext Intelligible for Readers.** Hypertext '89 Proceedings, Pittsburgh, Pennsylvania, November 5-8 1989 pp. 15-26.

[Miller et al 1992]

Miller, G., Hoffert, E., Chen, E.S., Patterson, E., Blacketter, D., Rubin, S., Applin, S. A., Yim, D. and Hanan, J. **The Virtual Museum: Interactive 3D Navigation of a Multimedia Database.** The Journal of Visualisation and Computer Animation, vol. 3 1992 pp. 183-197.

[Minsky 1981]

Minsky, M. **A Framework for Representing Knowledge.** In Mind Design, Haugeland, J. (ed), MIT Press. 1981. Reprinted in Brachman, R.J. and Levesque, H.J. (eds) Readings in Knowledge Representation. Morgan Kaufmann 1985 pp. 246-262.

[Moore 1989]

Moore, J.D. **A Reactive Approach to Explanation in Expert and Advice - Giving Systems.** PhD dissertation, University of California, Los Angeles, 1989.

[Neuron Data 1990]

Neuron Data. **NEXPERT OBJECT™ Product Description.** May 1990.

[Noreault et al 1981]

Noreault, T., McGill, M. and Koll, M.B. **A performance evaluation of similarity measures, document term weighting schemes and representations in a Boolean environment.** In Oddy et al (eds) Information Retrieval Research, Butterworths 1981 pp. 57-73.

[O'Connor 1985]

O'Connor, B.C. **Access to Moving Image Documents: Background Concepts and Proposals for Surrogates for Film and Video Works.** The Journal of Documentation, 41(4) December 1985 pp. 209-220.

[O'Docherty & Daskalakis 1991]

O'Docherty, M.H. and Daskalakis, C.N. **Multimedia Information Systems - The Management and Semantic Retrieval of all Electronic Data Types.** The Computer Journal, 34(3) 1991 pp. 225-238.

[Oren 1990]

Oren, T. **Designing a New Medium.** The Art of Human Computer Interface Design. Laurel B. (ed). Addison Wesley 1990 pp. 467-479.

[Packard 1993]

Packard, E. **Horror House.** Bantam Books, 1993.

[Palay & Fox 1981]

Palay, A.J. and Fox, M.S. **Browsing Through Databases.** In Oddy et al (eds) Information Retrieval Research, Butterworths 1981 pp. 310-324.

[Parsaye et al 1989]

Parsaye, K., Chignell M., Khoshafian, S. and Wong, H. **Intelligent Databases.** John Wiley & Sons 1989.

[Peck & John 1992]

Peck, V.A. and John, B.E. **Browser-Soar: A Computational Model of a Highly Interactive Task.** Proceedings of CHI '92, Monterey, California, May 3-7 1992 pp. 165-172.

[Philips & Sony 1990]

Compact Disc Interactive - Full Functional Specification. Philips Consumer Electronics B.V. September 1990.

[Philips IMS 1992a]

Philips IMS. **Introducing CD-I.** Addison-Wesley 1992.

[Philips IMS 1992b]

Philips IMS. **The CD-I Design Handbook.** Addison-Wesley 1992.

[Philips IMS 1992c]

Philips IMS. **The CD-I Production Handbook.** Addison-Wesley 1992.

[Price 1993]

Price, R. **MHEG: An Introduction to the Future International Standard for Hypermedia Object Interchange.** Proceedings of ACM Multimedia '93, Anaheim, California, August 1-6 1993 pp. 121-128.

[Raghavan & Wong 1986]

Raghavan, V.V. and Wong, S.K.M. **A Critical Analysis of Vector Space Model for Information Retrieval.** Journal of the American Society for Information Science, 37(5) 1986 pp. 279-287.

[Rankin 1991]

Rankin, P.J. **Hustlers, Window Shopping and The Information Market - Some Thoughts on CD-i Browsing.** Private communication, September 1991.

[Rudd & Cole 1991]

Rudd, D.A. and Cole, R.S. **CD-I Browser Characteristics.** Philips Research Laboratories Technical Note 3050, August 1991.

[Salomon 1990]

Salomon, G. **Designing Casual-Use Hypertext: The CHI '89 InfoBooth.** Proceedings of CHI '90, Seattle, Washington, April 1-5 1990 pp. 451-458.

[Salton 1989]

Salton, G. **Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.** Addison-Wesley 1989.

[Shatford 1986]

Shatford, S. **Analysing the Subject of a Picture: A Theoretical Approach.** Cataloguing and Classification Quarterly, 6(3) Spring 1986 pp. 39-62.

[Shiels 1991a]

Shiels, M.A. **Performance Measurements for the Supportive Advisor on CD-I.** Philips Research Laboratories AI Group Informal Note AI/GEN/0049, March 1991.

[Shiels 1991b]

Shiels, M.A. **CD-I Frames 1.0 - Reference Guide.** Philips Research Laboratories AI Group Informal Note AI/GEN/0046, September 1991.

[Sun 1995]

The Java™ Language: A White Paper. Sun Microsystems, 1995.

[Walker 1990]

Walker, D.P. **The CD-I Picture Browser.** Philips Research Laboratories Technical Note 3004, November 1990.

[Waterworth & Chignell 1991]

Waterworth, J.A. and Chignell, M.H. **A Model For Information Exploration.** Hypermedia, 3(1) 1991 pp. 35-58.

[Winston & Horn 1984]

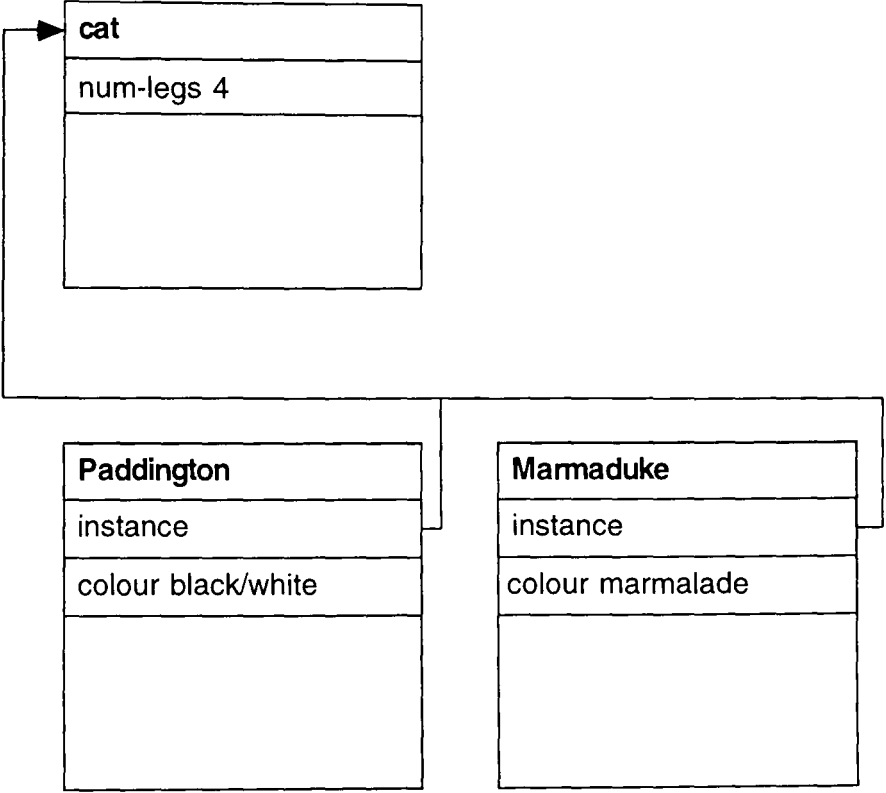
Winston, P.H. and Horn, B.K.P. **LISP Second Edition.** Addison-Wesley, 1984

[Zellweger 1989]

Zellweger, P.T. **Scripted Documents: A Hypermedia Path Mechanism.** Hypertext '89 Proceedings, Pittsburgh, Pennsylvania, November 5-8 1989 pp. 1-14.

Glossary

anchor	An anchor is the source of a link in hypertext/hypermedia systems.
at end	“at end” is the behaviour of the path navigation function <i>gb_next_on_path</i> at the end of the path.
at start	“at start” is the behaviour of the path navigation function <i>gb_previous_on_path</i> at the start of the path.
CLUT	Colour lookup table is a technique used in computer graphics systems whereby a selection of colours (usually expressed as the number of bits assigned to record each entry - thus 8-bit gives 256 colours, 7-bit 128 and so on) are stored in a table. Indexes to the table are stored for each pixel in the display buffer.
current frame	At any one time, a single frame will be the focus of the application and this is referred to as the “current frame”.
current link relation	The current link relation is the relation used by the <i>gb_previous</i> and <i>gb_next</i> functions.
current frame on path	The name of the current frame on the path is known as the “current frame on path”. This can be viewed as a pointer into the path list. The <i>gb_next_on_path</i> and <i>gb_previous_on_path</i> functions move this pointer within the list and hence change the “current frame on path”.
current path	The name of the path currently being browsed is known as the “current path”. If no path is being browsed this is the name of the last one browsed. The current path is changed by <i>gb_goto_path</i> .
DYUV	Delta YUV is a coding scheme used to compress the information required to display tv quality pictures. DYUV is used in CD-i.
frame base	A collection of frames is generally known as a frame-base.

history	A special path (called history) in which all frames visited using Generic Browser functions are recorded.
inheritance	<p>In a frame system, inheritance allows default values in a class structure to be held in parent nodes and inherited by siblings lower in the hierarchy.</p>  <p style="text-align: center;">Figure 86. An Inheritance Hierarchy.</p> <p>In the diagram above information common to all cats is stored in the <i>cat</i> frame and inherited by the instances. Thus <i>Paddington</i> would inherit four legs, as would <i>Marmaduke</i>..</p> <pre>(get-value 'Paddington 'num-legs)</pre> <p>would return 4.</p>
inner product	The inner product is a vector similarity measure used by the vector search function in the Generic Browser.

marking	<p>The facility for putting a mark on an information item is known as marking. It is used to indicate an item that is of interest. The use of different marks allows the database to be divided up into various sets. The Generic Browser allows browsing of such marked sets. For some purposes it is desirable to associate a value with a mark. An example of this might be a preference rating for items being browsed. An application can then make use of this extra information to produce sorted subsets for further browsing.</p>
navigation	<p>Navigation is the process of moving the application's focus around the frame base, changing the current frame.</p>
path	<p>An ordered list of items that can be browsed with previous/next functions (i.e. in a linear fashion) is known as a path. An item can appear in a path any number of times. Paths are used purely for navigation.</p> <p>The Generic Browser maintains various pointers into the network of browsable items. The most important of these are the current frame and the current path. In addition, the position within any marked set, marked set, or on any path, is also maintained and updated when the relevant navigation functions are called. In order that applications can allow their users to retrace their steps, a simple history mechanism is provided with a <i>gb_go_back</i> function that backtracks through the frames already visited.</p>
path frame	<p>A path frame is the actual frame containing the path information such as the actual list of items on the path (the path list), a pointer to the current path frame within that list (current frame on path) and information that enables the navigation functions <i>gb_next_on_path</i> and <i>gb_previous_on_path</i> to know what to do at the start and end of the path.</p>
path list	<p>The actual list of data frames on a path is known as a path list.</p>
pixel	<p>A pixel is single point on a screen display.</p>

weight	A value attached to a characteristic or query term to modify that terms contribution to a search is known as a weight.
---------------	--

Appendices

A CD-i - A Typical Consumer Multimedia Platform

An important objective of this work is to demonstrate its suitability for implementation on the types of multimedia machine sold for use in the home. To this end all the implementation work has been carried out on CD-i hardware. To give the reader a better appreciation of the capabilities of such platforms the specification of the CD-i player is detailed below. The CD-i hardware, disc format and system software are defined in a standard document known as "The Green Book" [Philips & Sony 1990].⁴²

A.1 CD-i Disc Format

Before describing the CD-i hardware it is important to provide details of the disc format which is also part of the CD-i standard.

capacity

A CD-i disc can hold a maximum of 650M bytes of data. This could be

- 100 million words
- 250,000 pages of A4 text
- 72 minutes of MPEG video
- 72 minutes of CD-DA music
- 19 hours of speech quality audio
- 7000 full colour natural images (DYUV)
- 120,000 graphics screens (CLUT)
- or a proportional mix of any of these.

tracks and sectors

A disc is made up of tracks. These form a spiral on the disc surface. There is a maximum of 99 tracks on a CD-i disc and these can be of two types: data or CD audio. Each track is made up of sectors each of 2352 bytes. The data in a sector can be any one of five types - audio, video, data, empty (used for packing in real time files) and message (used to warn CD-DA users that a CD-i disc must not be played in a CD audio player).

The data on the CD is random access but the seek times can be quite long. The Green Book defines a maximum of three seconds to seek from the edge of the disc to the centre.

⁴²For further information on the CD-i hardware and the CD-i authoring process see also [Philips IMS 1992a, Philips IMS 1992b & Philips IMS 1992c].

(In practice most players achieve a better seek time than this). The standard also guarantees that within any 20M bytes the seek time will be less than a second.

The data rate from the disc is 172k bytes per second. This data stream can be divided into up to 32 channels, up to 16 of which can be used for audio.

interleaving

This channel structure is the key to interleaving. Data in different channels can be mixed, sector by sector, on the disk.

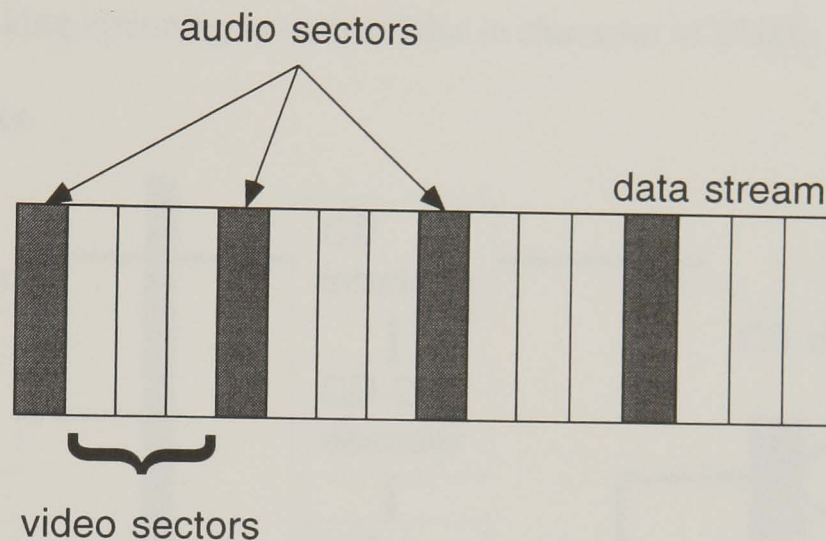


Figure 87. Interleaved data.

This means that, for example, audio can be played as a picture is being loaded. Here the audio, B stereo for example, takes 4 channels. The picture data uses another channel.

Interleaving makes it possible for different audio standards to be used on a single disc allowing lower quality (less disc space) for speech whilst a higher quality is used for music. Parallel audio tracks can be used applications such as multilingual discs and seamless cuts are possible between audio channels.

real-time files

Another important optimisation used in CD-i is the provision of the so-called real-time file. In an ordinary file system such as that used with a CD-ROM [ISO 1988], the disc looks to the system like any other peripheral device and the files on the disc are accessed through a directory structure. For multimedia this has the disadvantage that whenever a different file is accessed the directory must be consulted for the file information to allow the seek to be made. This slows down disc access which for slow devices such as CD drives is significant. Real-time files are not read as in conventional filesystems but are asynchronously streamed into specially allocated memory buffers.

A.2 Player Architecture

The "base-case" CD-i player has a 10MHz 68000-based CPU with 1M bytes of RAM and 8k bytes of non-volatile RAM. The CD can store 650M bytes of data accessed at 172k bits per second. There are four video planes: a background plane (used for digital video), two foreground planes (which can display a variety of different formats including CLUT graphics and DYUV coded natural images) and a cursor plane. CD-i audio is ADPCM coded to a variety of quality levels in either mono or stereo.

The CD-i player runs a version of the OS-9 operating system called CD-RTOS. This is a real-time, multi-tasking operating system similar in character to UNIX.

functional blocks

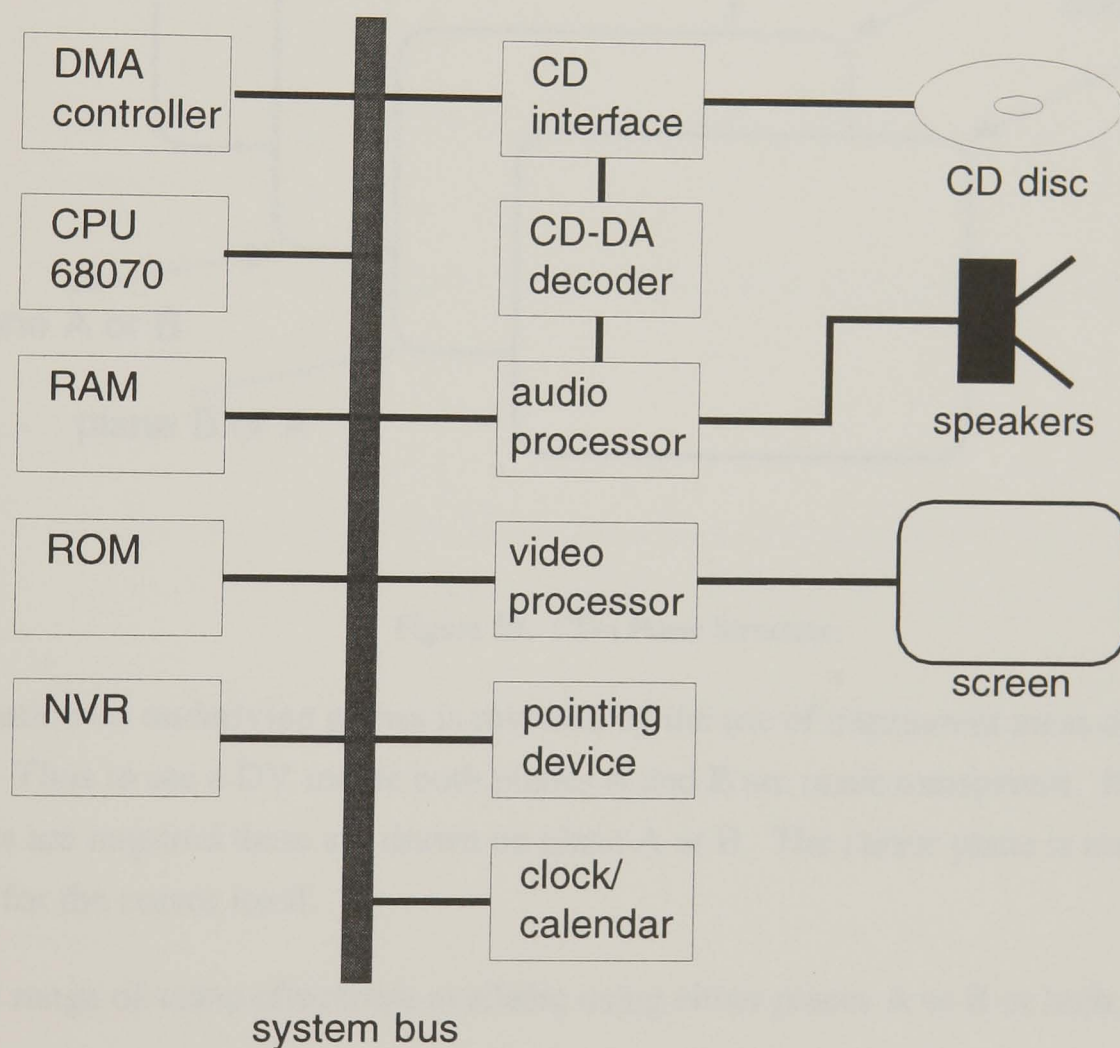


Figure 88. The functional units comprising a CD-i player.

As the data is read off the disc the system determines what type of data is in the sector (from the sector header). The data is then either passed immediately to the audio decoder or DV decoder or buffered in the player's memory ready for use by the application. CD-DA data is passed to the CD audio circuitry whilst ADPCM audio is decoded through its own decoder. The video decoder handles all picture data except DV. The player's RAM is used to hold the application software, sound maps, fonts, video images and graphics.

screens and planes

The CD-i display hardware has four planes. All the planes are displayed simultaneously with only the relative positions of planes A and B being switchable. The plane closest to the cursor plane is known as the front plane, the one behind it, closest to the background plane, is known as the back plane.

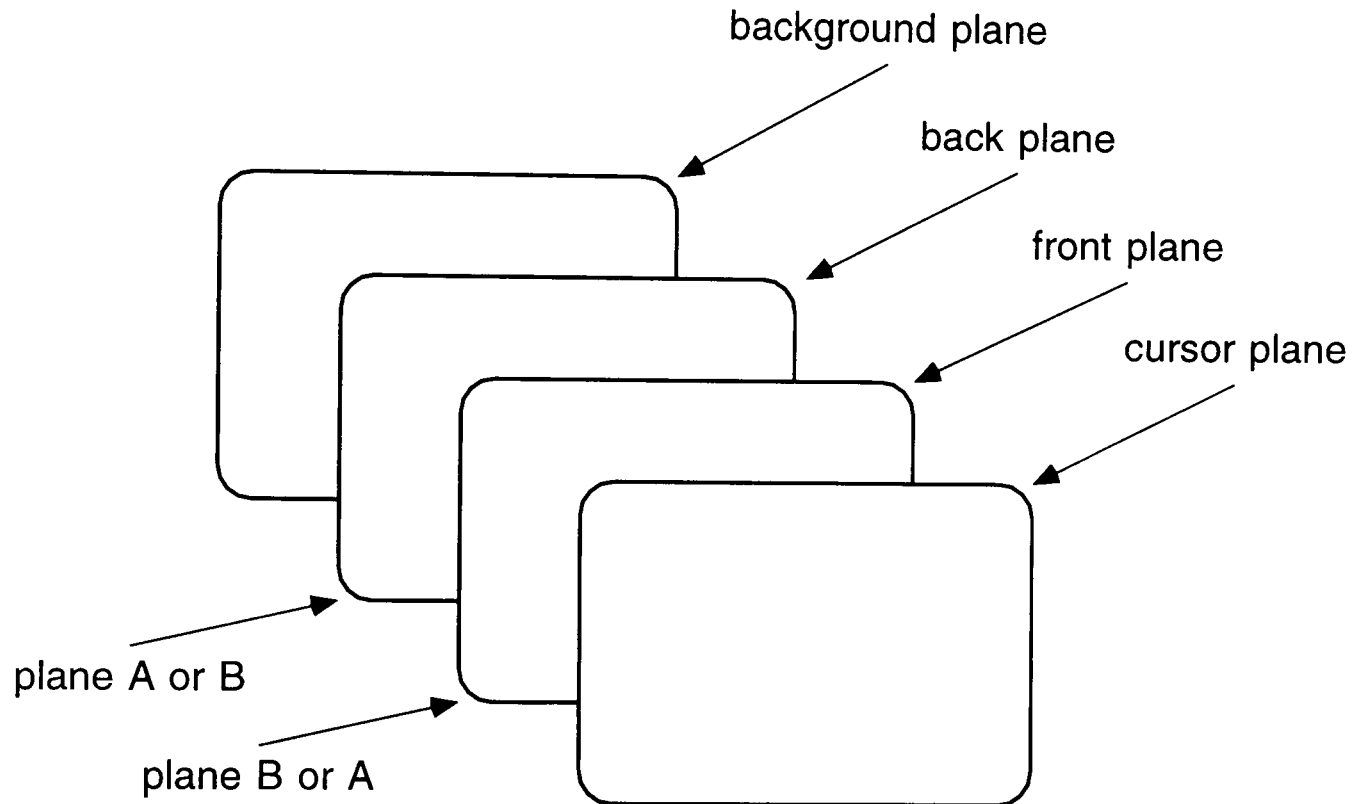


Figure 89. CD-i Plane Structure.

Information on underlying planes is revealed by the use of transparent areas on the planes above. Thus to see a DV movie both planes A and B are made transparent. If VCR controls are required these are drawn on plane A or B. The cursor plane is transparent except for the cursor itself.

A wide range of video effects are available using either planes A or B or both.

Single plane effects:

Cuts - the image is changed instantaneously to another.

Partial updates - not all the screen need be changed.

Scrolling - the image can be larger than the display. In this case horizontal and/or vertical scrolling are provided.

Mosaic effects - an effect where the image can be filtered into larger and larger blocks.

Fade up or fade down - the intensity of the image can be modified allowing fades up from or down to black.

Two plane effects:

Dissolve or cross fade - here the image on one plane is dissolved with the image on the other so if plane A is being displayed, a dissolve will end up with plane B and vice versa.

Cut - an instantaneous swap between which plane is front.

Wipes - the back plane is revealed top to bottom or left to right or bottom to top or right to left. Variations on this theme are:

curtain - here the reveal either starts at the sides and progresses inwards (closing) or vice versa (opening).

blind - the image is divided into horizontal stripes each of which does a top to bottom wipe.

square - the back image is revealed from all the sides inwards or from the centre outwards.

Transparency

chroma key - here a specified colour is made transparent. Wherever this colour appears in the image on the front plane, the image on the plane below is visible.

mattes - here a specified region (of any shape) on the front plane is made transparent.

video

The maximum screen size (normal resolution⁴³) is 384 x 280 pixels (107520 pixels). Within this is a so-called safety area which is guaranteed to be displayed on any properly adjusted television. Important information or interaction objects should not lie outside the safety area. Because CD-i can be used in both PAL and NTSC environments a compromise called compatibility mode is generally used for discs which must play in both environments.

⁴³CD-i can display at "double resolution" which doubles the number of pixels horizontally and "high resolution" which doubles both horizontal and vertical giving 768 x 560 (interlaced).

	maximum screen size	safety area
PAL	384 x 280	320 x 250
NTSC	360 x 240	320 x 210
Compatibility mode	384 x 280	320 x 210

Table 5. CD-i screen sizes.

Considerable flexibility is provided by a range of image coding methods designed to give maximum image compression and quality for differing types of image.

RGB 5:5:5 - this coding method gives no compression. 32,767 colours are available with a colour value being stored for each pixel. Each image consumes about 200k. This method is used for high quality natural images. It requires both planes A and B for a single image.

DYUV - delta luminance colour difference. With this coding method each image is compressed to about 90k. It is best suited to natural images. The differences between pixels, line by line, are stored because the eye is more sensitive to changes in luminance (brightness) than chrominance (colour).

CLUT - colour look up table. This method is best suited to graphics images. It comes in three flavours - CLUT8, CLUT7 and CLUT4 depending on the number of bits used to store the clut index. Thus CLUT8 offers 256 colours, CLUT7 128 and CLUT4 16 colours all from a palette of 16 million. CLUT7/8 images consume about 90k bytes per image. CD-i players can have a different clut for each line of the picture (though this must be traded off against the memory required for the cluts).

RLE - run length coding is used where there are large blocks of a single colour in the image such as cartoons. It records the colour and the number of pixels of that colour. For cartoon-type images the compression leads to images taking about 10k bytes.

QHY - quantised high-resolution Y (luminance). This method uses both planes (A and B) to improve the sharpness of DYUV images.

CD-i allows the image coding method to be specified on a line-by-line basis so that the screen can be split into a number of horizontal bands of different coding types. This might be used to have sub-titles to DYUV encoded images displayed using CLUT7 which is better suited to text.

digital video

A digital video cartridge is available for CD-i players. This has 0.5M bytes of RAM for video buffering and an extra 1M byte of RAM available to the system. The cartridge allows MPEG 1 encoded video to be played from the CD onto the background plane.

audio

To complement the video flexibility CD-i offers a range of audio quality levels. The application designer can trade-off fidelity for disc space.

level	Sampling rate	data rate	channels	fraction of disc bandwidth	total time per disc (hours)	quality
CD-DA	44.1 kHz	20 kHz	1 stereo	1/1	1.2	CD audio
CD-i ADPCM level A	37.8 kHz	17 kHz	2 stereo	1/2	2.4	new LP, no hiss
			4 mono	1/4	4.8	
level B	37.8 kHz	17 kHz	4 stereo	1/4	4.8	FM radio
			8 mono	1/8	9.6	
level C	18.9 kHz	8.5 kHz	8 stereo	1/8	9.6	AM radio
			16 mono	1/16	19.2	

Table 6. CD-i audio formats.

As well as playing audio direct from disc CD-i can place the sound data into a memory buffer called a soundmap. This can then be played on-demand without further access to the disc. One second of level A audio needs 80k bytes of memory, one second of level C audio 10k bytes. Soundmaps are often used for the “clicks” associated with pressing a button.

B Generic Browser Functions

This appendix lists the functions implemented in the final version of the Generic Browser. For each function the syntax is shown followed by a brief description of what the function does.

B.1 Function Arguments

The parameters to all Generic Browser functions must be pointers (char *) to null terminated strings. The function descriptions use the following abbreviations:

c the name of a class,
f the name of a frame,
m the name of a mark,
p the name of a path,
q the name of a query frame,
r the name of a relation,
s the name of a slot,
v a value.

All print functions send their output to stdout. Lists are printed with the start of the list on the left.

B.2 Relation Functions

gb_get_link_relation()

```
#include <gb.h>
char *gb_get_link_relation( )
```

gb_get_link_relation returns the current link relation or NULL if there isn't one.

gb_new_link_relation()

```
#include <gb.h>
void gb_new_link_relation( r )
char *r;
```

gb_new_link_relation makes the current link relation r.

gb_next()

```
#include <gb.h>
char *gb_next( )
```

gb_next makes the current frame the frame reached by following the current link relation. *gb_next* return TRUEVAL if it is successful, NULL if it fails.

gb_previous()

```
#include <gb.h>
char *gb_previous( )
```

gb_previous makes the current frame the frame reached by following the inverse of the current link relation. *gb_previous* returns TRUEVAL if it is successful, NULL if it fails.

gb_print_link_relation()

```
void gb_print_link_relation( )
```

gb_print_link_relation prints the name of the current link relation as set by *gb_new_link_relation*.

B.3 Path Functions

gb_add_to_path()

```
void gb_add_to_path( p )
```

gb_add_to_path adds the current frame to path *p*. The frame is inserted immediately before the current frame on path.

gb_create_path()

```
void gb_create_path( p, path_list )
char *p;
char *path_list;
```

gb_create_path generates a new path frame *p* with path *path_list*.

gb_delete_path()

```
void gb_delete_path( p )
char *p;
```

gb_delete_path deletes the path frame *p*.

gb_get_all_paths()

```
char *gb_get_all_paths( )
```

gb_get_all_paths returns the names of all the paths in the frame base as a Cframes list. If there are no paths in the frame base, NULL is returned.

gb_get_at_end()

```
char *gb_get_at_end( p )  
char *p;
```

gb_get_at_end returns the "at end" value for path *p*. Valid values are "stop" which means that nothing happens when *gb_next_on_path* is called at the end of the path; "restart" causing a circular path, and "goto frame" which makes the current frame *frame* when the end of the list is reached. If *p* does not exist or is not a path, NULL is returned.

gb_get_at_start()

```
char *gb_get_at_start( p )  
char *p;
```

gb_get_at_start returns the "at start" value for path *p*. Valid values are "stop" which means that nothing happens when *gb_previous_on_path* is called at the start of the path; "restart" causing a circular path, and "goto frame" which makes the current frame *frame* when the start of the list is reached. If *p* does not exist or is not a path, NULL is returned.

gb_get_current_frame_on_path ()

```
char *gb_get_current_frame_on_path ( p )  
char *p;
```

gb_get_current_frame_on_path returns the "current frame on path" from path *p*. If *p* does not exist or is not a path, NULL is returned.

gb_get_current_path()

```
char *gb_get_current_path( )
```

gb_get_current_path returns the current path or the last path visited if not currently on any path.

gb_get_path()

```
char *gb_get_path( p )  
char *p;
```

gb_get_path returns the names of all the frames on the path *p*, in order, as a Cframes list. If *p* does not exist or is not a path, NULL is returned.

gb_goto_path()

```
void gb_goto_path( p )
char *p;
```

gb_goto_path makes *p* the current path. The “current_frame_on_path” of the old path is maintained and will become the current frame if *gb_goto_path* is used to return to this path. If *p* does not exist or is not a path, nothing is done.

gb_new_at_end()

```
void gb_new_at_end( p, v )
char *p;
char *v;
```

gb_new_at_end sets the “at end” value for path *p* to *v*. Valid values are “stop” which means that nothing happens when *gb_next_on_path* is called at the end of the path; “restart” causing a circular path, and “goto frame” which makes the current frame *frame* when the end of the list is reached.

gb_new_at_start()

```
void gb_new_at_start( p, v )
char *p;
char *v;
```

gb_new_at_start sets the “at start” value for path *p* to *v*. Valid values are “stop” which means that nothing happens when *gb_previous_on_path* is called at the start of the path; “restart” causing a circular path, and “goto frame” which makes the current frame *frame* when the start of the list is reached.

gb_new_path()

```
void gb_new_path( p, path_list )
char *p;
char *path_list;
```

gb_new_path makes *path_list* the path for path frame *p*.

gb_next_on_path()

```
char *gb_next_on_path( )
```

gb_next_on_path sets the current frame to the next frame on the current path. The behaviour of this function at the end of the path is defined by the “at_end” value for the

path. By default paths are circular i.e. the path is restarted when the end is reached. Returns TRUEVAL if successful, NULL on failure.

gb_previous_on_path()

```
char *gb_previous_on_path( )
```

gb_previous_on_path sets the current frame to the previous frame on the current path. The behaviour of this function at the end of the path is defined by the “at_start” value for the path. By default paths are circular i.e. the path is restarted at the end when the start is reached. Returns TRUEVAL if successful, NULL on failure.

gb_print_all_paths()

```
void gb_print_all_paths( )
```

gb_print_all_paths prints out a list of all the paths in the frame base.

gb_print_at_end()

```
void gb_print_at_end( p )  
char *p;
```

gb_print_at_end prints the “at end” value for path *p*.

gb_print_at_start()

```
void gb_print_at_start( p )  
char *p;
```

gb_print_at_start prints the “at start” value for path *p*.

gb_print_current_frame_on_path()

```
void gb_print_current_frame_on_path( p )  
char *p;
```

gb_print_current_frame_on_path prints the name of the current frame on path *p*.

gb_print_path()

```
void gb_print_path( p )  
char *p;
```

gb_print_path prints the names of the frames on path *p*.

gb_remove_from_path()

```
void gb_remove_from_path( p )  
char *p;
```

gb_remove_from_path removes the current frame from path *p* provided that the application is “on path” *p* i.e. the “current frame on path” is the current frame. If *p* does not exist or is not a path, NULL is returned.

B.4 Mark Functions

gb_delete_all_marked()

```
void gb_delete_all_marked( m )  
char *m;
```

gb_delete_all_marked deletes the mark *m* (and its values) from all frames.

gb_delete_mark()

```
void gb_delete_mark( m )  
char *m;
```

gb_delete_mark deletes mark *m* (and its value) from the current frame.

gb_get_mark_value()

```
char *gb_get_mark_value( m )  
char *m;
```

gb_get_mark_value returns the value of mark *m* in the current frame. If no such mark exists on the current frame, NULL is returned.

gb_get_all_marked()

```
char *gb_get_all_marked( m )  
char *m;
```

gb_get_all_marked returns a list of all the frames marked with mark *m*. If there are no such frames, NULL is returned.

gb_mark()

```
void gb_mark( m, v )
char *m;
char *v;
```

gb_mark marks the current frame with mark *m* and associates value *v* with this mark.

gb_next_marked()

```
char *gb_next_marked( m )
char *m;
```

gb_next_marked makes the current frame the next in the list of those with mark *m*. Note that this list is “circular”. Returns TRUEVAL if successful, NULL on failure.

gb_previous_marked()

```
char *gb_previous_marked( m )
char *m;
```

gb_previous_marked makes the current frame the previous in the list of those with mark *m*. Note that this list is “circular”. Returns TRUEVAL if successful, NULL on failure.

gb_print_all_marked()

```
void gb_print_all_marked( m )
char *m;
```

gb_print_all_marked prints the names of the frames with mark *m*.

gb_unmark()

```
void gb_unmark( m )
char *m;
```

gb_unmark removes mark *m* (and its value) from the current frame.

B.5 Basic Search Functions

gb_find_all_frames()

```
char *gb_find_all_frames( s, v )
char *s;
char *v;
```

gb_find_all_frames returns a list of all frames in the frame base which have a slot *s* where the value *v* appears as one of the values in this slot. If there are no such frames, NULL is returned.

gb_find_all_frames_in_class()

```
char *gb_find_all_frames_in_class( c, s, v )
char *c;
char *s;
char *v;
```

gb_find_all_frames_in_class returns a list of all frames in the frame base which are in class *c* and have value *v* as one of the values in slot *s*. If there are no such frames, NULL is returned.

gb_find_all_marked_frames()

```
char *gb_find_all_marked_frames( m, s, v )
char *m;
char *s;
char *v;
```

gb_find_all_marked_frames returns a list of all frames in the frame base which are marked with mark *m* and have value *v* as one of the values in slot *s*. If there are no such frames, NULL is returned.

gb_find_all_frames_with_slot()

```
char *gb_find_all_frames_with_slot( s )
char *s;
```

gb_find_all_frames_with_slot returns a list of all frames in the frame base which have slot *s*. If there are no such frames, NULL is returned.

gb_mark_all_frames()

```
void gb_mark_all_frames( s, v, m )
char *s;
char *v;
char *m;
```

gb_mark_all_frames finds all frames in the frame base which have value *v* as one of the values in slot *s* and marks them with mark *m*.

gb_mark_all_frames_in_class()

```
void gb_mark_all_frames_in_class( c, s, v, m )
char *c;
char *s;
char *v;
char *m;
```

gb_mark_all_frames_in_class finds all frames in the frame base which are in class *c* and have value *v* as one of the values in slot *s* and marks them with mark *m*.

B.6 Vector-Space Search Functions

gb_vector_search_class()

```
void gb_vector_search_class( c, s, q, m)
char *c;
char *s;
char *q;
char *m;
```

Does a vector search on all the frames in class *c* using query *q* and marking the result with mark *m* (marked with the similarity measure). Note that search terms are held as values in slot *s* (in both the data frames and the search query frame *q*). *s* may be a list, in which case each slot in the list is used for searching. The inner product similarity measure is used. Weighted terms are not supported.

gb_vector_search_marked()

```
void gb_vector_search_marked( m1, s, q, m2 )
char *m1;
char *s;
char *q;
char *m2;
```

Does a vector search on all the frames marked with *m1* using query *q* and marking the result with mark *m2* (marked with the similarity measure). Note that search terms are held as values in slot *s* (in both the data frames and the search query frame *q*). *s* may be a list, in which case each slot in the list is used for searching. The inner product similarity measure is used. Weighted terms are not supported.

B.7 History Functions

gb_add_to_history()

```
void gb_add_to_history( )
```

gb_add_to_history adds the current frame to the history path. The following functions update the history when they are called:

```
gb_goto_frame()
gb_next()
gb_previous()
gb_goto_path()
gb_next_on_path()
gb_previous_on_path()
gb_next_marked()
gb_previous_marked()
```

gb_clear_history()

```
void gb_clear_history( )
```

gb_clear_history deletes the contents of the history path.

gb_get_history()

```
char *gb_get_history( )
```

gb_get_history returns the history path as a Cframes list.

gb_get_history_status()

```
int gb_get_history_status( )
```

gb_get_history_status returns TRUEVAL if history recording is switched on, NULL otherwise.

gb_go_back()

```
void gb_go_back( )
```

gb_go_back retraces one step back through the history list. This function treats the history like a stack, removing the last location visited each time it is called. Obviously the changes to the current frame resulting from calling the *gb_go_back* function are not recorded in the history.

gb_history_off()

```
void gb_history_off( )
```

gb_history_off switches off the history recording mechanism.

gb_history_on()

```
void gb_history_on( )
```

gb_history_on switches on the history recording mechanism. All changes to the current frame (except those caused by *gb_go_back*) are recorded in the history path until *gb_history_off* is called.

gb_print_history()

```
void gb_print_history( )
```

gb_print_history prints, in reverse chronological order, the names of those frames visited.

B.8 Miscellaneous Functions

gb_get_current_frame()

```
char *gb_get_current_frame( )
```

gb_get_current_frame returns the name of the current frame.

gb_goto_frame()

```
void gb_goto_frame( f )
```

gb_goto_frame makes f the current frame.

gb_print_current_frame()

```
void gb_print_current_frame( )
```

gb_print_current_frame prints the name of the current frame.