UNIVERSITY OF SOUTHAMPTON

Data Integrity Problems in an Open Hypermedia Link Service

by

Hugh Charles Davis, M.Sc., MBCS

A thesis submitted for the degree of
Doctor of Philosophy

in the

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science

November 1995

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE

DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Data Integrity Problems in an Open Hypermedia Link Service

by Hugh C. Davis

A hypermedia link service is system which stores the information describing hypertext links in a database which is separate from the data content over which the links are intended to operate.

One of the first open hypermedia link services was Microcosm, which takes this philosophy to the extreme, storing not only the links in a separate database, but also the information about the endpoints of the links. The most important advantage of such an organisation is that the system remains open so that hypertext functionality may be extended to third party applications.

The first part of this thesis describes the background to open hypermedia link services and describes the Microcosm system, which was developed by the Multimedia Research Group at the University of Southampton.

The major problem with storing all the information about links separately from the content is that such a scheme introduces many opportunities for the introduction of inconsistencies and the loss of integrity of the hypermedia data model. The second part of this thesis examines these problems, and proposes a number of solutions. It concludes that no one solution can resolve all the problems, and that in order to ensure integrity it is necessary to impose some conditions which limit the degree of openness.

Contents

List of Figures

List of Tables

Acknowledgements

The Microcosm project has been a team effort, and there are many people who need acknowledgement in a thesis which, in part, describes this system. First I would like to thank Professor Wendy Hall, whose untiring efforts to provide academic direction and the financial resources have produced an environment in which such research can be undertaken. I would also like to thank her personally for the help and advice she has given me throughout the rather protracted production of this thesis.

Secondly, I need to thank the longer standing members of the Microcosm research team, particularly Dr Ian Heath, Dr Gary Hill and Dr Rob Wilkins, but I should also mention Dr Z. Li, Simon Knight, Dr Les Carr, Mark Weal, Dr Steve Rake, Dr Gerard Hutchings, Roger Rowe, Jon Russell and Nick Beitner, all of whom have helped me by both producing the system and by giving me time to discuss my own research interests.

Within the Hypermedia research community there have been many people with whom I have spent time discussing the integrity issues, but I would like particularly to thank Dr Adrian Vanzyl from Monash University, Dr Kasper Østerbye from Aalborg University, Dr Randy Trigg from Xerox Parc, Dr Antoine Rizk from Euroclid and Peter Nürnberg from Texas A&M University, many of whose ideas are reported here, even if they wouldn't recognise them as such.

I would like to thank my family for sacrificing family holidays for the last year to give me the time to write-up this thesis, and my mother and father for giving over their dining room table so I had the peace and quiet to do it. Dr John Leggett was also kind enough to provide me space at Texas A&M University to complete the final version.

Finally, I would like to give special thanks to Professor David Barron, who has patiently guided me in my research for the past six years.

Chapter 1. Introduction

In 1988 a small group of researchers in the Image and Multimedia Group at the University of Southampton gathered together to discuss the problems involved in extending the functionality of hypertext systems to include multimedia data objects. Experiments that had been conducted with commercially available systems such as Guide (Brown, 1987) and Hypercard (Goodman, 1987) had proved unsatisfactory as there was little or no support in the model for incorporation of new data types. The best one could hope to achieve was to launch external applications to display the required data and having achieved this there was no support for further hypermedia functionality; the user was at a *dead-end*.

The motivations of the Southampton group to improve on this situation were various. Work was being conducted on CD-ROM's and Video Disks, which are read only media, so it was necessary to invent a system that would work without attempting to embed information in the data. My own work on integrating third party software engineering tools implied the same restriction. Work on reducing authoring effort in producing hypertexts pointed to the need for maintaining link information separately from the data, and finally the desire to design a "workbench" system that would be extensible for both new data types and for new functionality indicated a modular extensible design.

Ever since we took the decision to hold all link information separately from the content information, I have been concerned about the implications that this model has upon the integrity of the hypermedia structure, and aside from the management of the development of the Microcosm system, this subject has been my main research interest. When all link information, including the information about the end points of the links (the anchors) is separated from the content, there are numerous opportunities for inconsistencies to occur. The content may be edited, in such a way that the links no longer correctly reference the content (the editing problem); content files may be moved so that the link is anchored in a non-existent file (the dangling link problem), and attempts to edit hypertexts and to keep versions of hypermedia projects lead to problems that are more complex than versioning and concurrency control in standard database systems.

The way that Microcosm has evolved over the past few years makes it very difficult to specify exactly who did what work.

Chapter 2, Background, is my own review of the development and current state of hypermedia research, with a particular slant towards open hypermedia systems, hyperbases and link services.

Chapters 3, 4, 5 and 6 describe the current state of Microcosm and our research in this area. These chapters show how Microcosm solves many of the problems of earlier systems and indicates further areas of research that must be addressed before such systems achieve full industrial strength. Although I have been involved, as one of the original designers and latterly as research laboratory manager, in all stages of the development of this system, the work reported in these chapters is often the result of the efforts of other individual researchers, and I only take credit for this work in so far as I have been part of the design team and have managed much of the research.

Chapter 3 describes the current state of the design of Microcosm, and is a much extended version of the description which was published in Davis et al (1992). Chapter 4 describes work carried out by Dr Z. Li under my supervision, to integrate information retrieval into Microcosm (Li et al, 1992)(Li, 1993). Chapter 5 describes work being carried out by Simon Knight, under my supervision, on the integration of third party applications into Microcosm (Davis et al, 1994b). Chapter 6 describes extensions to the Microcosm model which are the product of current research, including the steps to ensure that the system is truly multimedia (Hall & Davis, 1994)(Davis et al, 1994a).

These chapters have value, both as an introduction to Microcosm for readers, so that they may understand the framework within which the problems and solutions discussed in the remaining chapters occur, but also as the first full description of Microcosm and the results of research on the system. Descriptions of Microcosm in published papers have been necessarily short, and other PhD theses have tended to concentrate on a particular aspect of the system, rather than giving the whole picture.

Chapters 7 to 11 form the most significant part of this thesis and contain my analysis of the data integrity issues in open hypermedia link services, along with my solutions to these problems. These chapters explain how data integrity

problems arise in systems such as Microcosm, and investigate a number of solutions to each problem. A separate chapter on versioning examines the state of the art in applying versioning to hypertext systems, and investigates its use as a tool for ensuring data integrity. The work reported in these chapters is entirely my own, except where specifically credited elsewhere.

Chapter 2. Background

2.1. Early Systems

The earliest hypertext systems aimed to provide speedy access to information by cross referencing and associative indexing. They were inspired by a theoretical design by Vannevar Bush (Bush, 1945), and typified by mainframe based systems offering information frame by frame, such as Augment (Engelbart, 1963) and ZOG (McCracken & Akscyn, 1984). The terms *hypertext* and *hypermedia* are attributed to Ted Nelson, whose book, Literary Machines (Nelson, 1981), describes a vision of a universal hypertext system, Xanadu; the ideas were probably too far ahead of the technology, and Nelson is still striving to produce this system.

In the latter part of the 1980's, workstation and operating system technology had advanced to the stage that it was possible to start conducting research into the ideas originally perceived by Bush, Engelbart and Nelson. Research oriented systems such as Notecards (Halasz, 1988), KMS (Akscyn et al, 1988), Neptune (Desisle & Schwartz, 1986), gIBIS (Conklin & Begeman, 1988) and Intermedia (Yankelovich et al, 1988)(Haan et al, 1992) began to appear. For more information on these systems and basic definitions of hypertext and hypermedia concepts the reader is referred to the classic survey by Conklin (1987) and the more recent book by Nielsen (1995).

The introduction of Guide for the Macintosh in 1986 was followed by the release of the first PC based commercial hypertext systems in 1987, including Hyperties (Shneiderman, 1987), PC Guide (Brown, 1987) and Hypercard (Goodman, 1987). Hypercard is not strictly a hypertext system in itself, but more a toolkit and prototyping environment in which it is possible to create hypertext systems (Hutchings et al, 1993a), but the effect of bundling this product with the Apple Macintosh was profound, and led to much research into the usability of hypertext interfaces, e.g. Hutchings et al (1993b) and Nielsen (1990).

2.2. The Problems with Current Systems

Given the quantity of research, the introduction of commercial systems and the increasing interest in multimedia, it might have been expected that hypermedia would have made great strides forward to the extent that hypermedia systems would be present on most desktops: this has not happened, and a number of commentators have attempted to explain this failure and to indicate the issues that must be addressed in order to make progress beyond the state where hypertext is used largely for static software help systems and relatively small technical documentation applications.

In 1988 Frank Halasz suggested that the research community addressed "Seven Issues for Next Generation of Hypertext" (Halasz, 1988), which included:

1. *Search and Query*

   Link navigation by itself is not sufficient for hypermedia information discovery. Hypermedia applications must also provide query based interfaces.

2. *Composites*

   Composites are groups of nodes and links, possibly defined by some schema, which may be treated as a single entity within the system.

3. *Virtual Structures*

   These are dynamic structures, analogous to views in relational databases, which are constructed at the time that the structure is accessed.

4. *Computation over Hypermedia Networks*

   Internal or external engines should be available, which work over the network to produce new information or modify the existing information.

5. *Versioning*

   It should be possible to maintain and manipulate the version history of the network.

6. *Collaborative Work*

   Collaborative working on hypermedia systems requires that the system allows multi-user access to information, along with suitable concurrency control and notification mechanisms.

7. *Tailorability and Extensibility*

   It should be easily possible to modify or extend the behaviour of the hypermedia system.

On revisiting this subject in 1991 (Halasz, 1991) he concluded that the monolithic hypertext systems of the 1980's were no longer a viable species, and that they were being replaced by "open" systems consisting of independent but communicating components. The challenge was to "end the tyranny of the link" by producing systems which allowed users to navigate using other tools such as information retrieval, relevance feedback and expert systems to determine the association between items of information.

In 1991 Malcolm et. al. (1991) stated that

> "Current hypermedia tools do not support the needs of collaborative work groups in distributed heterogeneous environments and cannot be integrated into the existing and planned computing environments of large enterprises like Boeing. It is in meeting these needs, however, that hypermedia could make its greatest impact."

They suggest that the research community address the following issues.

Interoperability: the ability to access and link information across arbitrary platforms, applications and data sources.

Links and Object Attributes: the facility to attach attributes to links and objects, such as who made the link, where are the link destinations etc., so that it will be possible to filter out the required information.

Link Anchors: all objects must be capable of becoming link anchors within the system, and anchors must support multiple links.

Private and Public Links: It must be possible to create both public and private links and annotations. Group workspaces must also be supported.

Templates: In the same way that many word-processed documents may be described by a template, so may many hypertext nodes. The template would describe the objects that would be in the node and the links required from that node.

Navigational Aids: There must be multiple methods of navigation, such as link following, network level browsing, tables of contents, dynamic query, bookmarks and trails. Filtering of available information must be possible.

Configuration Control: Versions of nodes, links and webs must be possible. Users will require permission rights to elements of the system, and must be able to alter parts without affecting others.

Concurrency Control: In a large multi-user system it must be possible for more than one worker to access the same object.

Programmability: It must be possible to extend the system functionality.

Operating Systems, Storage and Networks: These must improve to allow distribution of hypermedia systems across heterogeneous hardware. They must also deal with the caching of large multimedia objects as well as compression and decompression.

What the authors required was an adaptable environment for the integration of data, tools and services.

In 1993 Irene Greif (1993) set the challenge to the hypertext community to design hypertext systems to *support* other applications. She proposed a lightweight hypertext with multiple navigational routes between pieces of information held in different applications.

2.3. The Dexter Model

By the end of the 1980's the state of the art in hypertext systems was probably best represented by Intermedia (Yankelovich et al, 1988)(Haan et al, 1992). The research community had recognised the need for a hypertext reference model: a workshop

was held in December 1989 on hypertext standardisation, which was attended by the main US research teams and at which various models were proposed. The "standard" that emerged was the *Dexter Reference Model* (Halasz & Schwartz, 1990)(Halasz & Mayer, 1994). This model attempted to capture the important abstractions found in a range of hypertext systems that existed at the time, such as Augment (Engelbart, 1963), Intermedia (Yankelovich et al, 1988)(Haan et al, 1992), Hypercard (Goodman, 1987), Hyperties (Shneiderman, 1987), KMS (Akscyn et al, 1988), Neptune (Desisle & Schwartz, 1986), Notecards (Halasz, 1988) and the Sun Link Service (Pearl, 1989). The intention was also to capture those abstractions that would be found in future systems, but in this respect the model failed, as will be explained subsequently in this section. The goal of the model was to provide a basis for the comparison of systems and for the development of interchange and interoperability standards.

The Dexter model divides hypertext systems into three layers, as illustrated in figure 1. The *runtime layer* contains the basic hypertext functionality: the mechanisms to support the user's interactions such as accessing, viewing and manipulating the hypertext. *The storage layer* contains the network of links and node specification that represents the structure of the particular hypertext. The *within component layer* is the content of the particular component, node, document or frame. Dexter sensibly makes no attempt to model the structure of the component, leaving this to other modelling tools such as ODA (HyperODA, 1992): as far as the other layers of the Dexter Model are concerned the within component layer might as well contain text or graphics or any other data format. Nor does the model attempt to specify the runtime layer in detail: this is the province of the individual hypertext system.

The interface between the runtime layer and the storage layer is provided by *presentation specifics*: these are details held within the storage layer that inform the presentation layer how components should be handled, for example whether a component should be displayed for viewing or for editing and the size and position of windows.

The interface between the storage layer and the within component layer is provided by anchors. The within component layer is responsible for maintaining the position of anchors within the content of the component itself, and will know these anchors by some unique identifier. The storage layer contains links which

refer to end points by a specifier which consists of a component specifier (which may be resolved to the UID of one or more nodes), the anchor ID, the direction of the anchor (source, destination or bi-directional) and any presentation specific information.

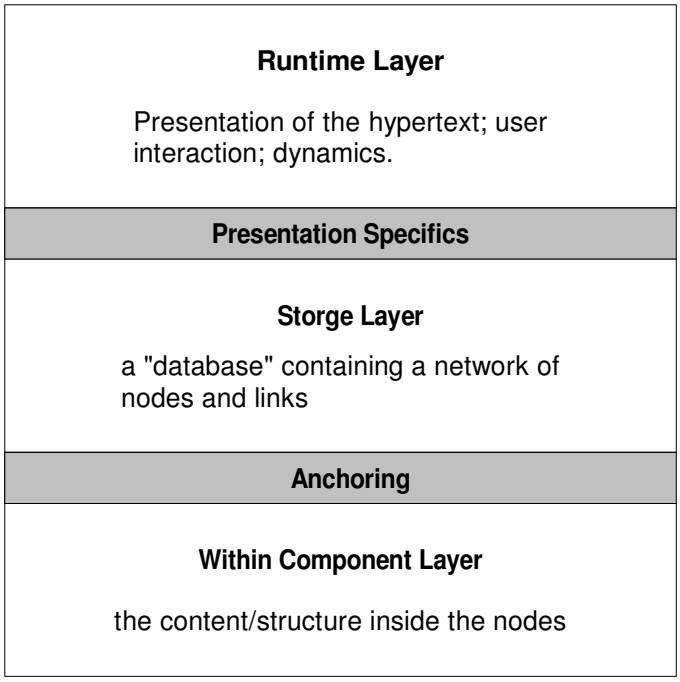

Figure 1: The Three Layer Dexter Hypertext Reference Model.

The Dexter Model has not gone without criticism. Grønbæk & Trigg (1992) found the requirement that it should not be possible to create dangling links, made for restrictive interfaces to link creation and editing. They also point out that there are multiple orthogonal concepts of directionality of links (such as the directionality of a "supports" typed link and the direction of traversal). In DHM they implemented typed anchors: *Whole-Component* anchors use a whole component as the source of the anchor: *Marked* anchors refer to a marked point within a component: *Unmarked* (or *Keyword*) anchors allow the resolver function dynamically to situate an anchor at any point where a particular keyword occurs.

Grønbæk et al. (1993) found it necessary to extend the Dexter model to support long term transactions, locking and event notification so that a co-operative hypermedia system could be designed.

Maioli et al. (1993b) cite the existence of systems such as Microcosm and RHYTHM as evidence of the need to extend the Dexter model to include external anchors, and suggest that anchor id's should be replaced with anchor specifiers that may be resolved to internal or external anchors in much the same way that component specifiers are resolved to UID's.

Leggett & Schnase (1994) point to the shortcomings of using Dexter as an interchange standard, particularly its incomplete specification of multi-headed links, the problems with dangling link creation when importing partial hypermedia and the failure of the model to distinguish between separate webs as implemented in Intermedia. They also point out that the model has no concept of the semantics of arrival and departure when following links and that the notion of composites and their consistency is incomplete, as has also been noted by other authors (Grønbæk & Trigg, 1992)(Maioli et al, 1993a). Furthermore they suggest that anchors belong within the *link services* domain rather than within the *application* (or component) domain.

A further problem of the Dexter model is that it contains no specification of how to deal with temporal event specification; this aspect is well covered by the HyTime standard (HyTime, 1992)(Carr et al, 1994a)(DeRose & Durand, 1994), and partly for this reason it is probable that HyTime will be seen as a more complete hypertext interchange standard. The Microcosm team have carried out work on this subject (Carr et al, 1993).

The limitations of the Dexter model have had the result that few systems describe themselves as Dexter systems, but many describe themselves as Dexter *based*. The Microcosm team had designed their system and begun their implementation before the publication of the Dexter model, and it is perhaps due to this that the system has not been constrained by the problems described above. Like Leggett & Schnase (1994), we believe that the NIST standardisation workshop occurred a little too early in the development of hypermedia systems.

At the Workshop on Open Hypermedia Systems at ECHT '94 (Wiil & Østerbye, 1994), Randy Trigg and Kaj Grønbæk, whose DHM system (Grønbæk & Trigg, 1992) is probably the most Dexter compliant system of all, worked hard on updating the Dexter model to include such systems as Microcosm. The result (Grønbæk & Trigg, 1996) is an extension to the Dexter model to include *LocSpecs* and *RefSpecs*. A LocSpec (Location Specifier) is (normally) associated with an

anchor object, and contains attributes which identify how the *EndPoint* of that anchor is to be identified - typically by within-component Object ID, structure descriptor or by content search criteria. Thus, in Microcosm terms, a *LocSpec* contains exactly the information that identifies the source or destination of a link. A *RefSpec* (Reference Specifier) is an object belonging to a particular hypertext component, which contains the *LocSpec* and a presentation specifier. In Microcosm terms, *RefSpecs* are generally transient objects which are built at the time that the user makes a selection and action: in the case of Microcosm buttons, the *RefSpecs* are built at the time that the viewer is notified of the button's existence. Thus by extending the Dexter model it has become possible to represent a greater range of modern systems, including Microcosm.

2.4. Towards Industrial Strength Hypermedia

In attempting to solve the problems of current hypertext systems and to produce systems that will deliver industrial strength hypermedia-in-the-large, the research community has proposed a number of partial solutions. This section describes current progress in these areas.

2.4.1. Open Hypermedia

Much use and abuse has been made of the term "open". There are those who use the term to mean that the application runs on an open system such as UNIX. This is not what is meant in the context of open hypermedia. Various definitions have been proposed.

In Davis et al., (1992) we published our criteria for referring to a system as open. These criteria included.

1. *The hypertext link service should be available across the entire range of applications available on the desktop.* To us this implied that since applications would not normally be capable of manipulating anchor identifiers, as required by the Dexter model, it would be necessary to design a system that held links and their anchors externally from the node content.

2. *The link service must work across a network on heterogeneous platforms.* This implies that hypertext functionality must be provided by a number of communicating processes, and led to the concept of a link service - a framework for routing messages between the various components.

3. *The architecture should be such that the functionality of the system can be extended.* This implies that the design should be modular so that new components may be written to a specified API then added to the system. The API should be kept as simple as possible so that applications may be adapted to conform to the API.

4. *There should be no artificial distinction between author and reader.* Many systems have an authoring mode and a reader mode: such a system is not open from the reader's point of view. We believe that all users should have access to all parts of the system; this does not imply that one user will be able to access or change another user's data, but implies that this aspect should be controlled by the operating system access rights granted. Users should be able to create their own links and nodes within their private workspace, then change the access rights so that other users may view or edit them as required.

Subsequent authors and workshops have attempted to define the term "open hypermedia", and following the Open Hypertext Systems Workshop at Konstanz in May 1994 (Aßfalg, 1994) and the ECHT '94 Workshop on Open Hypermedia Systems at Edinburgh in September 1994 (Wiil & Østerbye, 1994), I have produced the following summary of current thinking.

The term *open* implies the possibility of importing new objects into a system. A truly open hypermedia system should be open with regard to:

1. *Size*

   It should be possible to import new nodes, links, anchors and other hypermedia objects without any limitation, to the size of the objects or to the maximum number of such objects that the system may contain, being imposed by the hypermedia system.

2. *Data Formats*

   The system should allow the import and use of any data format, including temporal media.

3. *Applications*

   The system should allow any application to access the link service in order to participate in the hypermedia functionality.

4. *Data Models*

   The hypermedia system should not impose a single view of what constitutes a hypermedia data model, but should be configurable and extensible so that new hypermedia data models may be incorporated. It should thus be possible to interoperate with external hypermedia systems, and to exchange data with external systems.

5. *Platforms*

   It should be possible to implement the system on multiple distributed platforms.

6. *Users*

   The system must support multiple users, and allow each user to maintain their own private view of the objects in the system.

No one system implements all the aspects of openness as described above, but systems that meet sufficient of the criteria to warrant the term open include Microcosm (Fountain et al, 1990) (Davis et al, 1992), the Sun Link Service (Pearl, 1989), Multicard (Rizk & Sauter, 1992), KHS (Hammwöhner & Rittberger, 1993), PROXHY (Kacmar & Leggett, 1991), SP3 (Leggett & Schnase, 1994), HyperTED (Vanzyl, 1993), RHYTHM (Maioli et al, 1993b), ABC (Shakelford et al, 1993), DHT (Noll & Scacchi, 1991) and the World Wide Web (Berners-Lee et al, 1992).

2.4.2. Hyperbases

In October 1992 an NSF sponsored workshop was held in Washington, D.C. (Leggett et al, 1993) and this was followed up by a second workshop at Seattle,

Washington in November 1993. These workshops were motivated by the belief that the current generation of hypermedia systems would not scale to deal with the very large information repositories such as those that would be found in digital libraries and the purpose was to separate the area of research that concentrates on hypermedia system implementation issues from other issues such as user interfaces and evaluation and usability studies. The areas that these workshops concentrated on were models and architecture, node, link and structure management, version control, concurrency control, transaction management and notification control.

The workshop participants considered hypermedia system architecture to consist of three layers as shown in figure 2. Typical applications are text and graphics editors, mail tools and multimedia presentation tools. The hyperbase layer provides the hypermedia data model to the applications and interacts with the storage layer which provides persistent storage for the data model abstractions (nodes, links, anchors, contexts etc.).

This model is similar to the Hypertext Abstract Machine (HAM) (Campbell & Goodman, 1988) which was used by both the Neptune CAD system (Desisle & Schwartz, 1986) and Dynamic Design (Bigelow, 1988) for providing the lower layers.

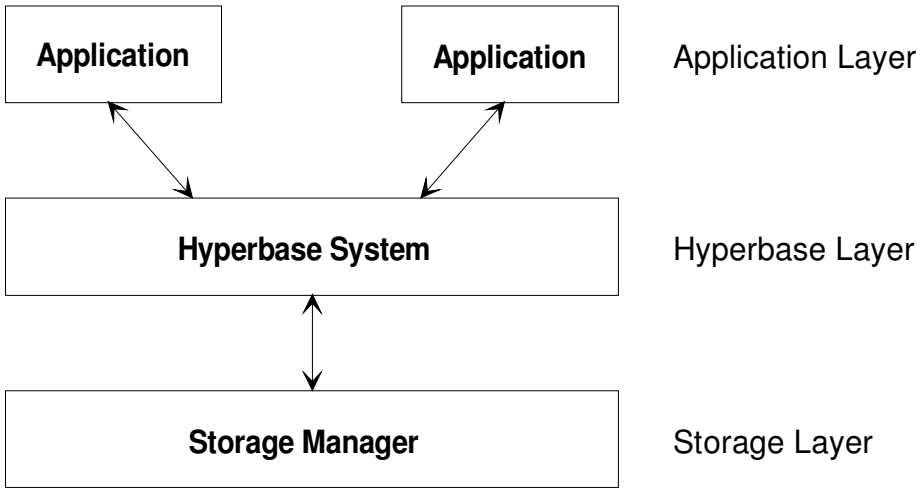

Figure 2: Three Layers of Hypermedia System Architecture

Hyperbase and hypermedia system architectures were characterised by seven dimensions:

1. *Scale*

   The size of the hyperbase, measured in such quantities as bytes of information, number of nodes, links, anchors, versions, contexts and views, and also the number of simultaneous users.

2. *Openness*

   The extent to which participating applications are free from restrictions imposed by the data model.

3. *Distribution*

   The extent to which the hypermedia processes may be distributed over multiple machines.

4. *Heterogeneity*

   The extent to which different hypermedia data models are provided by the hyperbase systems(s).

5. *Extensibility*

   The extent to which the data model may be extended to deal with new data abstractions and operations.

6. *Computational*

   The extent to which the system supports computation of data model elements such as nodes, links, anchors and virtual structures.

7. *Method of Interaction*

   The extent to which the model supports multiple users and their co-operation.

Many participants were using traditional database systems, typically object oriented, to provide the storage layer, and in theory this would allow different hyperbase systems to access any given stored hyperbase. In practice this is not

possible as there is no agreed data model: there was a majority agreement that the Dexter Model had had its day, and was insufficiently flexible for the purposes of the hyperbase community.

Systems that are making progress in the area of hyperbase infrastructure implementation include GMD-IPSI's CHS and CoVer (Schütt & Streitz, 1990), SP3 and HB2 (Leggett & Schnase, 1994) (Kacmar & Leggett, 1991) (Schnase et al, 1993), ABC and DGS (Shakelford et al, 1993) (Smith & Smith, 1991), Hyperform (Wiil & Leggett, 1992), DHM (Grønbæk & Trigg, 1992) and DHT (Noll & Scacchi, 1991). In effect, the overlap between the open hypertext system research community and the hyperbase research community is very large.

An interesting problem that the Hyperbase community is attempting to resolve is how to enable such systems to provide Hypertext services to third party applications. There are two problems: hyperbase systems expect to store the node content, whereas third party applications expect to address the filestore to retrieve their data, and secondly hyperbase systems expect the editor to handle anchors (or persistent selections) in order to ensure integrity. Current systems that address this problem have had to accept that such cases are exceptions and that the data will be allowed to reside on the filestore. In order to avoid integrity problems, ABC (Shakelford et al, 1993), only allows node to node links in such cases and HyperDisco (Wiil & Østerbye, 1994) uses a search engine to locate a given string.

2.4.3. Link Services and Lightweight Link Services

A link service is a process which provides hypertext functionality to applications which elect to communicate with this process. The link service provides an interface to which applications may connect in order to store and retrieve links in any object. Link services may also be coupled with object stores which hold all hypertext objects, possibly including the nodes themselves.

The term *link service* has recently become widely used to refer to any system which holds the links separately from the data. As such, any true Dexter model application and any hyperbase system may be regarded as a link service.

When we first started using the term link service to describe Microcosm's underlying architecture we were referring to something subtly different. A criticism of the three layer hyperbase model shown in figure 2 is that the interface

between the application and the hyperbase layer must be handled entirely by the application, generally using a client server model for communication. This has the disadvantages that each application that is hypertext enabled is required to implement all the code to interface to the hyperbase layer API, and that the applications must be active hypertext applications capable of collecting queries from the user, asking about such aspects as links and display characteristics and dispatching other applications as required.

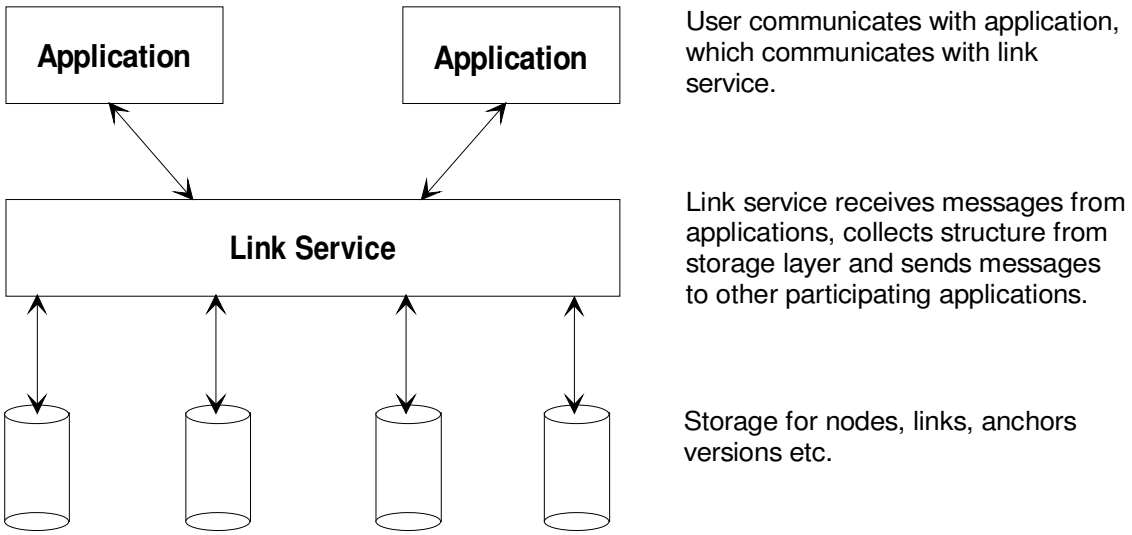

Figure 3: A link service model.

These problems, combined with concern to keep systems open and extensible, have led to the introduction of the concept of a *lightweight* link service layer.

Essentially the link service behaves as if it were a single application sitting above storage layer, and this application takes full responsibility for handling communications with all applications which the user wishes to employ. The link service is the active component which contains as much as possible of the functionality for communicating with both the storage layer and with the applications. I define the link service as *lightweight* if the participating applications may elect to use only a subset of the full link service API to provide a usable subset of the full hypertext functionality.

Although the idea of the link service was first published by Sun in 1989 (Pearl, 1989) and by the Microcosm team in 1990 (Fountain et al, 1990), it is only recently that the importance of the idea has been realised by the research community. Poltrock and Schuler's trip report on Hypertext '93 (1993) observes that link services are an idea whose time has come, while commenting on the importance of enabling third party applications for hypermedia use. In retrospect it is fair to comment that a number of systems have in effect used such a layer in their model, but have not specifically identified it as such.

A major research issue, and possibly a hindrance to the universal adoption of this concept, has become the degree of cohesion that should be expected between the applications and the link service. The Sun Link Service (Pearl, 1989) required that applications were adapted to co-operate with the link service, so that they could provide unique ID's for every object in the application's data, could display icons or *glyphs* to indicate the presence of an anchored object and provide some mechanism for selecting and following links when they were indicated.

The Microcosm Link Service (Davis et al, 1992)(Hall et al, 1993a) takes a less restrictive approach, allowing applications to participate in all of the link service protocols (fully aware viewers), some of the protocols (semi-aware viewers) or even none of the protocols (unaware viewers).

A further question is how the storage layer is implemented. Both the Sun Link Service and the Microcosm Link Service use the host operating system file system to provide this layer. This leads to potential problems as users may move and edit files directly using tools that are not link service aware, leading to inconsistencies. Intermedia (Yankelovich et al, 1988)(Haan et al, 1992), which in effect did have a link service layer, circumvented this problem by requiring that all objects in the hypertext were accessed via Intermedia applications or via the Intermedia File Browser; thus the hypermedia service was able to maintain hypertext integrity in a similar manner to hyperbases.

A link service may be viewed as an integrating technology and as such it is possible that in the future the need for this layer will become redundant as object request brokers such as CORBA (Corba, 1991) become integral within the operating system environment, and provide underlying system services for inter-application communication and integration. Industry standards such as OLE 2.0, AppleEvents (Apple Computer Inc., 1994) and Sun's ToolTalk (Julienne & Russell, 1993) are

already emerging, and need careful investigation. Preliminary studies by the DHM team on using OLE 2.0 as a link server have proved unsuccessful: they found that it was only possible to produce a very chunky style of hypertext, and that in any case its operation was too slow to be usable. However, operating system enhancements such as Apple's OpenDoc which allow applications to make the structure of their data visible through the operating system show great promise for open hypermedia systems, which need to be able to point to regions within the data as persistent selections.

Examples of applications that have implemented a link service layer, whether explicitly of implicitly include the Sun Link Service (Pearl, 1989), Microcosm (Davis et al, 1992)(Hall et al, 1993a), Intermedia (Yankelovich et al, 1988)(Haan et al, 1992), HyperTED (Vanzyl, 1993), Knowledge Weasel (Lawton & Smith, 1993), ABC (Shakelford et al, 1993), PROXHY (Kacmar & Leggett, 1991), Multicard (Rizk & Sauter, 1992) and SP3 (Leggett & Schnase, 1994); also Tompa et al. (1993) describe a prototype for such an architecture.

2.4.4. Wide Area Distributed Hypertext.

The original concept of Nelson's Xanadu (Nelson, 1981) was based on the premise that the world's literature (and pictures and videos etc.) would shortly all be available on-line. (This was in the early sixties!). Nelson was aware that this situation would not occur until the issues of copyright of electronic material was resolved, and to this end Xanadu was intended to allow writers to make special links (known as transclusions) to other writers' material in such a way that, whenever material was accessed, the original authors would be financially credited for their contributions. Nelson had a vision of a world wide network supplied by Xanadu servers.

Nelson's vision has not yet been realised, but we do have the World Wide Web (WWW) (Berners-Lee et al, 1992). WWW, in hypertext terms, is a simple second generation tool that fails to address many of the issues discussed above: components are written in a hypertext mark-up language known as *html* (HTML, 1994) which is an application of SGML, and links are embedded within this mark-up as explicit references to destination documents. So why has the Web become such a popular system, now accounting for a significant proportion of all Internet traffic and with a server at almost every site? The success of the system appears to

spite all the efforts of the hypertext research community to provide sophisticated systems, and to contradict all those wise men and women who have attempted to chart the course for the future developments of hypertext. The answer, we suspect (Carr et al, 1994b), lies in the availability and simplicity of the system and its highly successful and almost transparent network access.

WWW allows access to documents on a remote system via Universal Resource Locators (URL's) (Uniform Resource Locators, 1994). We have observed (Carr et al, 1994b) that the major use of the system is not in the form of link following from buttons on text spans within documents, but in practice the system is largely used as a method for navigating filestores on remote machines: the system is far more interactive than using such protocols as *ftp* and allows the user to browse in a more satisfying manner than *gopher*.

In spite of its success, there are a number of problems with the Web.

- The fact that links are embedded within source documents means that it is currently quite literally impossible to maintain these links if the destination of the link is moved or deleted. Any user will be familiar with the dangling links that result: considering the amount of data that is now mounted on Web servers the effort of maintaining Web integrity manually is unthinkable.

- The server maintains no attributes for the documents, so it is not possible to query servers to ask about the documents.

- It is not possible for users to make links within documents unless they have write access.

The majority of hypermedia systems that have been discussed so far have limited themselves to operating within defined network boundaries so that the system is able to exercise control of the hyperbase integrity. However, the success of WWW indicates that what users want hypertext for, more than any other factor, is wide area network access and browsing. Many of the hypermedia systems mentioned are addressing this area but it would appear that the first commercial hypertext successes will go to companies producing commercial Web servers such as Netscape and to those companies that are starting to charge for access to their Web servers.

Another system that provides wide-area network access is Hyper-G (Andrews et al, 1995) (Flohr, 1995) which is a client-server based hypermedia system produced at the Graz University of Technology. The Hyper-G server runs on Unix, and clients are available for X-Windows (Harmony) and MS-Windows (Amadeus).

Each Hyper-G server maintains a document management system (DMS), which keeps the attributes of the documents on the server, a link database which maintains the links, and an information retrieval engine, which can retrieve on both the attributes of the document and also the full text content of the document. Hyper-G servers may be arranged into hierarchies, and all belong to the world wide root server. This provides a scope for searches within Hyper-G. Servers communicate with each other to retrieve documents and links when required and as allowed by the user's permissions. The user connects directly to only one server. Hyper-G can also arrange to collect documents from other servers such as Web and Gopher servers.

The Hyper-G clients provide an interface for DMS browsing, authoring (currently in a format known as *htf*, but shortly in html 3.0, and in the long term in full SGML), and link creation. Hyper-G also provides clients for various other formats such as GIF and MPEG, and links may be made within these formats.

Links are stored as separate objects within Hyper-G. Each document knows the id's of all the links within itself, so when a client loads a document, it is also able to load all the links. The client is then able to edit the document (or move it or delete it) without causing integrity problems, since at the client end all links are effectively embedded within the document. Hyper-G provides a number of tools for working with links, such as a 3-D information space browser, a tool for changing link access rights, a tool for exporting part of the server information for use with a stand-alone runtime or to another server, etc.

A user may also connect to a hyper-G server using an html browser such as Netscape. In this case the server will not be able to send the links to the client. Instead it pre-compiles the htf and the links from the linkbase into an html document. Thus Web clients may browse a Hyper-G server, but will not have the benefits of DMS browsing, document authoring or link authoring.

2.4.5. Digital Libraries

In June 1994 the first Digital Libraries Conference was held at Texas A&M University in College Station, Texas (Schnase et al, 1994) and in June 1995 the second conference was held at Austin Texas (Shipman et al., 1995). The US experience of this technology is approximately a year ahead of Britain's, primarily due to the National Science Foundation's Digital Library Initiative. The attendees of these conferences were a mixture of computer scientists, librarians and social scientists.

The definition of a digital library may be taken as:

> "A Digital Library is a machine readable representation of materials, such as might be found in a university library, together with the organising information intended to help users find specific information." (Gladney et al, 1994).

From the above definition it would appear that a digital library is little more than a conventional library, accept that the information is stored in machine readable form. However, there are certain features of a digital library that distinguish it from the conventional library, most notably:

- it is possible to shield the user form irrelevant inter-library boundaries;
- different views of the same material may be presented to different classes of user;
- cross references to other material may be facilitated by hyper links;
- all material in the library may be searched and queried;
- the speed at which material may be added, browsed, searched and distributed is much higher.

The standard model of a digital library is similar to the hyperbase model. The digital library model in figure 4 draws a distinction between the presentation layer (the application that actually presents the results on the screen) and the application layer, which is actually a layer of services which access the server for the data, such as query systems, CAD systems, editors, hypertext engines etc.

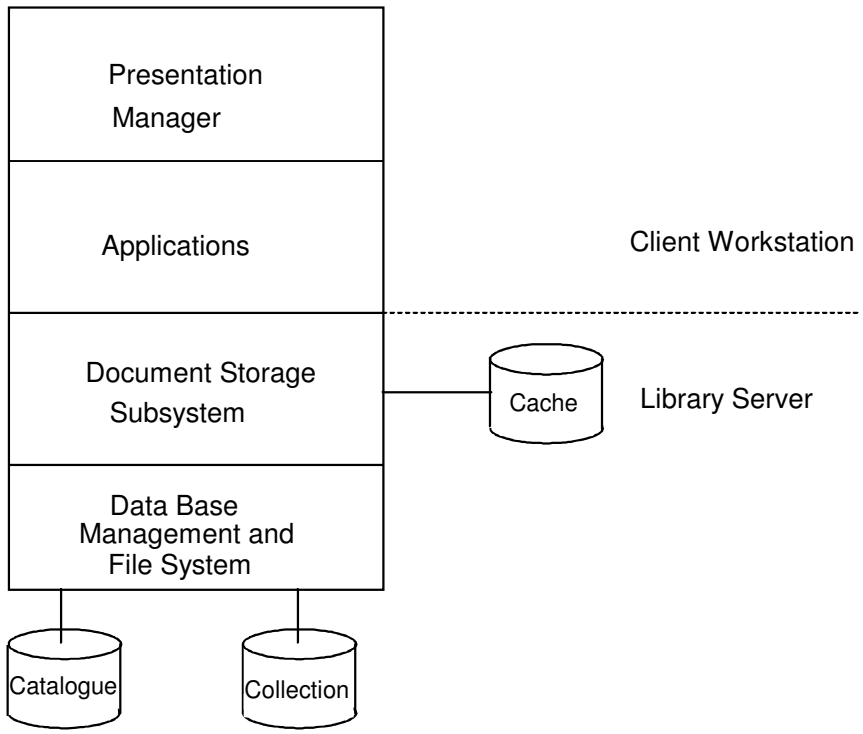

Figure 4: The Model of a Digital Library.

The relevance of this subject to a thesis on hypertext is that Digital Libraries perhaps represent the ultimate goal of hypermedia information systems: to produce wide area distributed information systems that allow seamless access to the world's on-line knowledge. There are those that suggest that the World Wide Web *is* the technology for the Digital Library, but the issues raised in the conferences indicate that this is only the beginning (Fox et al., 1995)

Many of the presentations made at these conferences suggested systems that were beyond the capability of current technology: they assumed enormous network bandwidth, super fast servers, display technologies as good as paper and application architectures that we can only currently sketch on paper. However, whenever this fact was pointed out, the speakers countered by pointing out that the challenge of digital libraries was to be seen in the same light as Kennedy's challenge to the country in the 60's, to put a man on the moon by the end of the decade. Large amounts of money will need to be invested, new technologies will have to be invented, new alliances of academics will need to work together, and the results will change the way we think and work, in ways as yet unforeseen.

Issues that arose out of the conferences tend to have both technical and social dimensions. Some of the active research areas are:

- Naming of Documents on a Network.
- Cataloguing. (who does it? can it be done automatically? what extra information does the digital library require?)
- How can we specify and optimise queries so that they can work across distributed servers holding vast amounts of data?
- How can we utilise "agents" or "actors" to help users to find the information they want? (do we look for information or does information look for us?)
- What sort of networks are required to deliver such quantities of information?
- Payment and security.
- User interfaces.
- How do we eliminate the boundaries between libraries using different systems and data formats.
- Display technologies.
- What social changes will occur as the result of such technologies, and how can we assist the smooth transition of such changes?

The immediate challenge to the hypermedia community is to build systems that improve on the World Wide Web, in allowing users access to multimedia information by numerous navigational methods (Davis, 1994), (Davis & Hey, 1995).

2.5. Standards for Hypermedia

The importance of hypermedia has not been missed by the international standards community, and a number of standards have been produced.

2.5.1. HyTime

Perhaps the most important of the hypermedia standards is the Hypermedia/Time-based Structuring Language, HyTime (1992) (Carr et al, 1994a)(DeRose & Durand, 1994), which attempts to preserve information about scheduling and interconnection of the related components of a hypermedia document. HyTime extends SGML to express cross referencing and temporal relationships between objects, but makes no attempt to specify the run-time semantics of links, or the behaviour of the objects which are linked. HyTime is a large and complex standard, and the reader is referred to the references above for further information. However, an important feature of this standard is the flexibility with which it allows users to express anchors within objects. In HyTime an anchor may be expressed by

- naming - e.g. an SGML entity name or id;
- counting - e.g. the 534th byte in this file, or the 3rd item in this list;
- querying - e.g. the first item with a *type* attribute whose value is 12.

The Microcosm system also uses the three methods above for expressing link anchors. The HyTime standard recognises the possibility of problems occurring when files are edited when they contain anchors expressed in the counting method. This topic is the main subject of this thesis.

2.5.2. HyperODA

HyperODA (1992), is a set of extensions to the Open Document Architecture (ODA) which is a container architecture which represents text and graphics, each according to some external standard, by providing parallel logical and physical views of the document. HyperODA extends ODA by providing extra content architectures for audio and images, as well as link objects and temporal layout. Since HyperODA objects are constrained to a small number of standards, it will always be possible to exchange hypertexts between HyperODA compliant applications, but it will not be possible to extend HyperODA applications to deal with new data types without a new version of the standard.

2.5.3. MHEG

MHEG (Price, 1993) is another container architecture, which allows its objects to be represented in any external standard, along with instructions for their presentation and behaviour. It is intended as an interchange standard addressing the problem of real time data exchange. MHEG differs from HyTime in that it specifies the display and control semantics whereas HyTime devolves this to the controlling application. MHEG differs from HyperODA in that it does not specify that set of standards which may be used to represent the objects.

Although the above standards have absorbed many hours of work in their production, there is, as yet, little evidence of their wide spread acceptance by the user community. There may be a number of reasons for this. The standards themselves, in their attempt to be as generic and flexible as possible, often become so abstract as to become incomprehensible to the user, but perhaps the most important fact is that hypermedia is still a technology in its infancy, and attempts to tie a community down to one particular standard are, as yet, premature. The wide spread acceptance of *html* may well be a testimony to its simplicity and the speed at which it evolves.

Chapter 3. Microcosm Technical Description

3.1. Introduction

Microcosm is an open hypermedia link service that was designed and implemented by a team at the Multimedia Research Laboratory at the University of Southampton, UK. The project started in 1988, when members of the team wishing to experiment with ideas in hypertext and multimedia became disenchanted with the systems that were commercially available and decided to produce a hypertext workbench: the primary goal was to produce a tailorable and extensible system that could support a number of different research requirements.

Initially a third year project student, supervised by Dr Andrew Fountain, produced a system for annotating text on a CD-ROM. This system was called Microcosm, because it was a subset of the much larger system we intended to produce. The name has stuck ever since as we never came up with a better one!

Following this project, Ian Heath and I began building prototypes for a larger version of the system. Ian Heath worked in C, and I worked in an object oriented language called Actor, which is a Smalltalk-like development environment for producing Windows programs (Davis & Heath, 1989).

After evaluating the three prototype systems the research group reformed and specified a design for what became known as Microcosm 1.0. An interesting design decision had to be taken at this stage; whether to produce the system in C or Actor? An important feature of the new Microcosm 1.0 design was that all functionality would be provided by separate processes. At this time we were working with Windows 2, which imposed very severe memory constraints on the programming environment, and it was not possible to have more than one Actor program in memory at a time. For this reason, coupled with the fact that we now had three PhD students who could program in C, we decided to continue the development in C, and this lead to the first realisation of Microcosm 1.0 (Heath, 1991)(Fountain et al, 1990).

Over the last six years the Multimedia Research Group has grown, under the direction of Professor Wendy Hall, from the original group of three academics and one PhD student, to a group of around 30 academics, research fellows, research assistants and PhD students. The Microcosm project accounts for about three quarters of this effort, and for the last four years I have been manager of the Microcosm Research Laboratory.

In 1992 the Technology in Teaching and Learning Programme (TLTP) financed a number of projects to produce teaching materials for use in higher education. Although still in beta test form and available on only one of the three target platforms (Microsoft Windows), Microcosm attracted considerable interest among those who were searching for a method of delivering materials that was more flexible and scaleable than the authoring systems that were commercially available. Microcosm is the delivery platform for two major TLTP projects, and is part of the environment for a number of others.

At the time of writing Microcosm is being used as originally intended, as a research workbench, in three SERC funded research programs, and is being used by around ten PhD students as a tool for their studies. It is also being used in many application areas, including technical documentation, project management, educational resource delivery, electronic publishing, geographic information systems and corporate information systems.

The company, Multicosm Ltd. has recently been formed with the intention of marketing the intellectual property rights in Microcosm to industry and commerce.

Over the last five years the design has been improved upon as researchers, industrialists and educationalists have all made their inputs. In 1992 I managed the production of Microcosm 2.0 from components produced within the research laboratory. This was the first version of Microcosm that was made publicly available. At this stage funding from JISC enabled us to employ a small team of professional programmers, headed my Dr Gerard Hutchings, who produced the first commercial version of Microcosm (version 3.0) in 1994. This team was taken over by Multicosm in 1995, and version 3.2 will shortly become available.

3.2. General Description

Microcosm is a system that allows users to organise and navigate large bodies of information which may come in many different formats.

Users may create their own resources using packages of their own choice, and simply incorporate these resources into the body of information of which Microcosm is aware; users may connect information together using links, or may annotate information; they may navigate through materials by a rich spectrum of navigational link following devices and searching and querying mechanisms; having located appropriate information they may manipulate the data using the tools that created the data in the first place and then they may publish resources and links, or maintain them in a private workspace.

A central feature of Microcosm is that it maintains all information about links, including link anchors, separately from the node component data: the node component data is untouched by Microcosm and may thus continue to be viewed and manipulated by the application that created it. The advantages of this approach are:

1. *Third party applications may be integrated into the hypertext system with minimal effort.*

   Inserting anchor identifiers into the node component data, as required by most other systems, would corrupt the data from the point of view of the source application. Maintaining anchors outside the data makes it possible for the application to continue to act as the data viewer and editor. Systems that keep data in a private format must also provide tools to access and process that data. However, this approach is a closed solution. Arguments amongst users about choice of text editors and drawing packages become almost religious in intensity. Users do not wish to be confined to using a particular package, and in any case it is not possible to predict the facilities that all users will require, nor is it sensible to duplicate existing specialised software packages. Hypermedia systems must allow users to create data in whatever package they choose, and then to make links from data in one application to data in another.

2. *The facility to make generic links.*

   If a link anchor is embedded within the node component data, then it will only be possible to follow that link from the place where its source anchor is embedded. If it is appropriate to make that link available from many different places, then a source anchor must be inserted at each of these places, requiring considerable authoring effort. However, if all link information is held separately from the data, it is then possible for a resolver function at the link service level to define which links may be followed from any user selected source object. A generic link is defined as one which may be followed from any occurrence of a particular object, such as a particular text span, wherever it occurs. This is a very powerful feature.

3. *It is possible to make links in read-only data.*

   In a co-operative environment a user may wish to allow other users to view data, but not to edit it. If link anchors must be inserted in the data then it will not be possible for the other users to create links in this user's data. Keeping the link anchors separately makes it possible to link into data owned by other users. This is also important when the data exists on a CD-ROM, video disk or other read-only media.

4. *It is possible to keep alternative link sets (or webs).*

   If link anchor information is embedded in the data, then *all* link anchors created by *all* users must be stored within the data. An important piece of information on a multi-user system might end up storing a very large number of link anchors. How would the system decide which ones to display to which users? One possibility is that each anchor might have an *owner* attribute, but this scheme would put a large onus on the viewer to filter and display only those anchors appropriate to the current reader. However, by keeping the link information in separate installable link databases (linkbases) it is possible for the user to select which sets of links will be available at any time, making it feasible to have multiple views of any given hypertext.

5. *Link processing tools.*

   In a system where links are completely embedded in the source document (such as World Wide Web) the task of asking the question "which documents

contain links that point to this document?" involves iterating over the entire hyperbase searching each document for anchors that meet this condition - and that is supposing that the user asking this question has file access permission and network access to the entire hyperbase. Keeping all links in centralised linkbases makes link processing operations of this sort, and thus hyperbase consistency, more tractable, and consequently makes such systems more scaleable.

In keeping with the principle of allowing third party applications to continue to access their data, Microcosm does not import node component data: the Microcosm storage layer *is* the file system. Microcosm does, however, keep various document attributes over and above those that are maintained by the file system, such as an extended description of the file, a logical type for the file, the name of the author, the viewer that Microcosm prefers to use for this format of data and any number of keywords and user defined attributes. These attributes are maintained by the Document Management System (DMS). Users may then access any file from the standard file system browser or file manager, or they may access files using the file manager supplied with Microcosm which displays the files using their extended descriptions and organises them hierarchically by their logical types. (See figure 8 in section 3.6).

3.3. Description of the Link Service

Microcosm consists of a number of autonomous processes which communicate with each other by a message passing system (see figure 6). The user interacts with applications known as *viewers*, which are any applications which can display the required format of data. Within the viewer the user may make a *selection*, then pull down the Microcosm action menu, and choose an *action* to take on the selected object. Figure 5 shows an example where the user has selected the text span "bacteria" within the Microcosm text viewer, and is about to select *Follow Link* from the action menu. The same approach may be taken from any application: the selection may be whatever the application allows the user to select, so, for example, it might be an area of a picture or an object in a drawing or a cell in a spreadsheet.

In figure 5 it is possible to observe that two words have been highlighted ("Spirillum" and "diagram"). Some viewers may have the ability to display buttons,

and the Microcosm text viewer is such an example: the user may double click on these buttons, rather than needing to select the span and choose "Follow Link" from the action menu. In Microcosm a button is simply a binding of a selection and action that shortcuts the usual process.

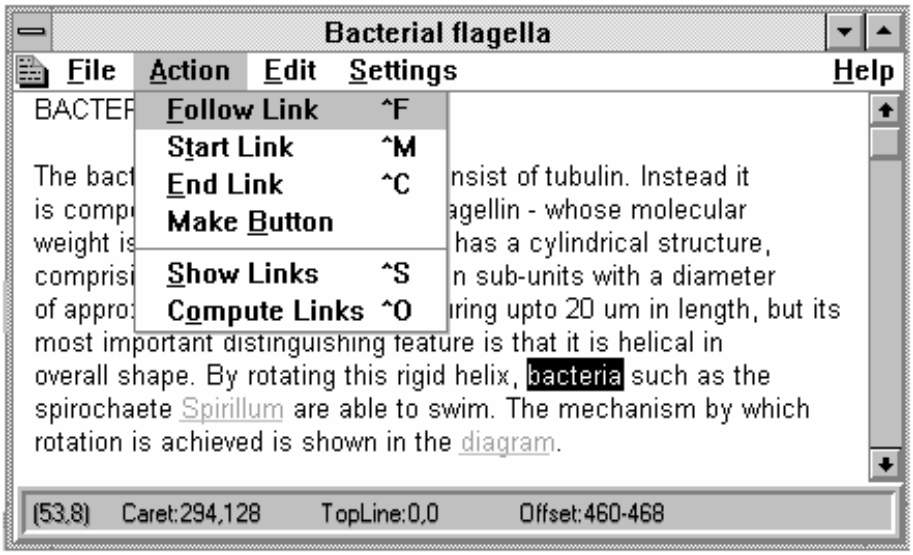

Figure 5. Making a selection and choosing an action within the Microcosm text viewer.

When an action has been chosen the viewer arranges to send a message to the *Microcosm Document Control System (DCS)*, containing the content of the selection, the action chosen by the user, the identifier of the source file and, if possible, some information about the position or context in which the selection was made. This information represents a potential source anchor. Microcosm then dispatches this information in a message to the *Filter Manager* which arranges to pass the message through all the filter programs that are currently installed and that have registered an ability to handle messages containing this particular action.

Filters are programs that have been written specially for Microcosm, and which can handle all the message communication protocols with the filter manager. Each filter in the chain that receives the message now has the opportunity to respond in one of four ways.

- It can block the message, so downstream filters will never see the message;
- It can ignore the message and pass it on to the next filter;

- It can alter the message in some way;
- It can create a new message (for example a message to dispatch a named document).

For example, an important filter is the linkbase filter which, on receiving a *Follow Link* message, looks up the source anchor in the database of links, and if successful it creates a new message asking to dispatch the destination file and display the destination anchor. It also passes the original message on to subsequent filters, unchanged.

The last filter in the chain is usually the *Available Links Filter*. This filter will trap any message asking to dispatch documents and display the link description to the user. If there is more than one choice the user selects which link to follow, and this message is returned to the Microcosm DCS, which consults the *Document Management System (DMS)* for details of the position of the document in the file store and the preferred viewer for this data, and dispatches this application for the user to view.

The above description is deliberately simplified in order to explain the system with as little distraction as possible. More detail is introduced in chapter 5.

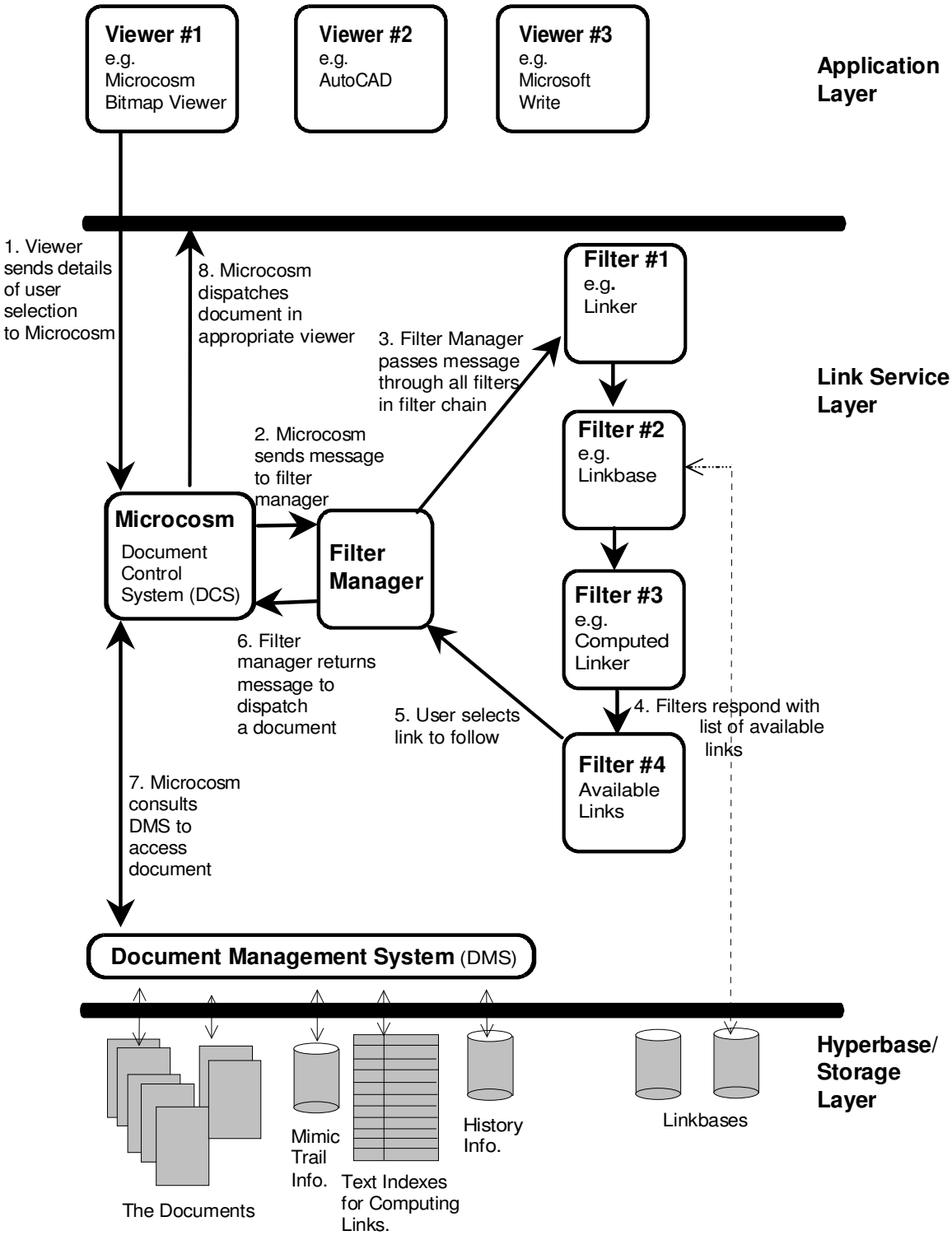

Figure 6: The Microcosm Link Service Model: showing typical message handling.

3.4. Viewers

Microcosm supports viewers with different levels of awareness of the full set of Microcosm protocols.

3.4.1. Fully Aware Viewers

These are programs that were written specifically for use with Microcosm, or applications adapted at the source code level: they are able to participate in the full range of protocols described in chapter 5. This means they will be able, for example to display buttons, they will have an action menu as shown in figure 5, and when dispatched they will be able to display or highlight a destination anchor in some way. Such viewers have been developed for:

- ASCII Text
- Rich Text Format (RTF)
- Bitmaps (BMP and JPEG)
- Sound (WAV)
- Digital Video (AVI)
- Animations (Sets of Bitmaps)
- Object Oriented Drawings (DXF)

3.4.2. Semi-Aware Viewers

These are third party application programs that have been adapted in some way to participate in some subset of the full communication protocol. Typically this is achieved using the application's own macro language: most serious Windows applications programs are now tailorable by some Basic-like macro language, and it is usually possible to add menu options and/or buttons to the tool bar. Using these facilities it is usually only a matter of a few lines of macro code to communicate selections via the Dynamic Data Exchange (DDE) to Microcosm. DDE is a facility for inter-application communication provided by the MS-Windows

operating system. The functionality that can be achieved is discussed later in chapter 5 and will depend on the amount of coding effort one wishes to invest. Some of the applications that have been adapted in this way include:

- Microsoft Word for Windows
- WordPerfect for Windows
- Ami Pro.
- Lotus 123 for Windows
- Autodesk AutoCAD
- CA Project
- Microsoft Access
- Asymetrix Toolbook
- Authorware Pro.
- Freelance

3.4.3. Unaware Viewers

Some application programs that one might wish to use offer no access to source code and have no macro programming language. In spite of this it is possible to obtain a surprisingly high degree of hypertext functionality using such applications: this is made possible by the use of a parasitic program known as the Universal Viewer which sits on top of the unaware program, and handles the communications with Microcosm. This is discussed in detail in section 5.5.

3.5. Filters

A filter is a program which attempts to respond in some way to Microcosm messages. Important filters are:

3.5.1. The Linkbase Filters

These filters provide the primary hypertext functionality. A link database (or linkbase) consists of a set of link descriptions indexed by all the fields in the description. A *follow link* message creates a query to the linkbase which will produce the set of all links in the linkbase with the selected source anchor, and which may be followed from the current context. Microcosm currently supports three primitive pre-authored link types.

1. Specific Links

   These are links which have some information to identify where in the document the anchor is situated, such as a named region, byte offset or structure identifier. Such links may be followed only when the source selection is situated on this anchor, and these links may be presented as buttons if the link author requires.

2. Local Links

   These are links that may be followed from a specific document, wherever the source selection occurs in this document.

3. Generic Links

   These are links that may be followed from wherever the source selection occurs.

If any links are found to be appropriate then the destination documents will be dispatched.

Figure 7 displays a simplified account of how the linkbase handles a message from the text viewer, where the user has selected the text span "kings" located at position 57-61 characters through the file. The linkbase contains a generic link on this source selection, so this will be offered to the user; it contains two local links on this selection, but only one is in the correct context, and it contains a specific link which is in the correct context: these are also offered to the user. All remaining linkbase records are for different source selections.

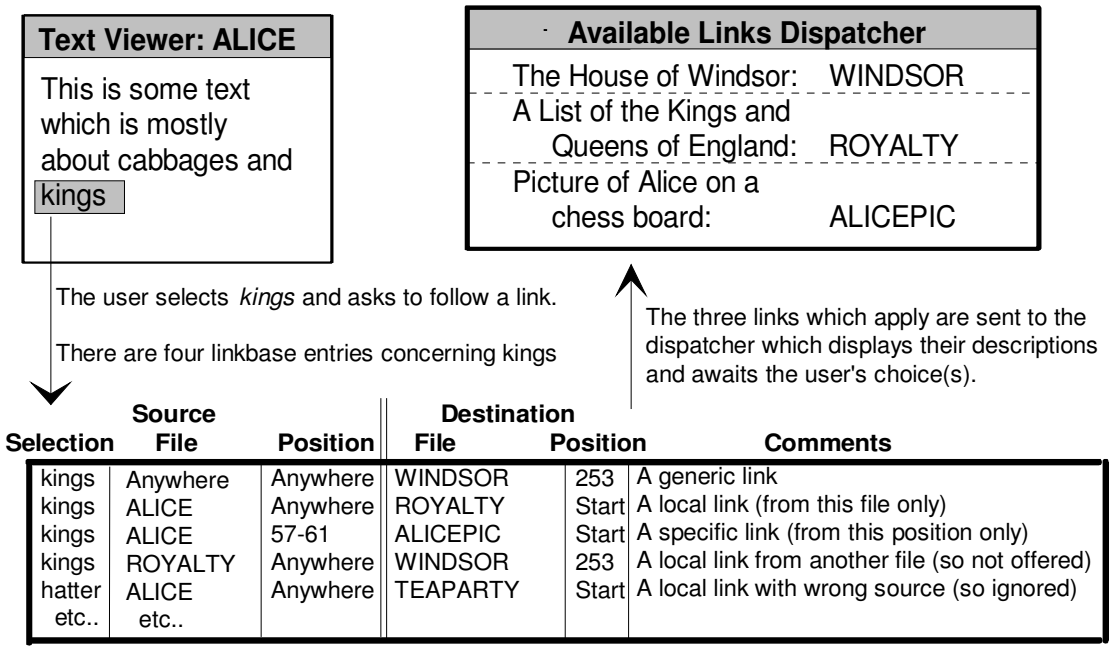

Figure 7: Demonstrating how the link base resolves which links to follow.

The account is simplified as link base entries contain many more details than those shown, and in practice the linkbase does not hold the file name but a unique document identifier which is resolved by the DMS to a real filename. Also, in this example the "position" field is shown as an offset through the file: this is by no means the only way of describing the position of an object in a file.

An important observation is that there is no limitation on the number of linkbases that may be installed at any one time: indeed it is standard practice to have around three, one providing links supplied by an original author, one providing generic links to background materials, glossaries and dictionaries, and one containing private links.

Microcosm also supports whole documents as link anchors, which makes it possible to have node-to-node links rather than the more usual selection-to-selection links. Microcosm will allow the creation of whole-node source anchors when the user asks to *start link* with no current selection, and will allow following of such anchors when the user asks to *follow link* with no selection. In practice this interface is not instinctive, and few users ever discover it. On more recent viewers we have made this explicit, either by providing options to make and follow whole node links, or by showing whole node links as buttons on the toolbar.

3.5.2. The Linker

The linker accepts *Start Link* and *End Link* messages which identify anchors within data, and when two such anchors are paired it allows the user to enter link details, such as the type of the link and the description of the link: this information is then stored in the currently selected linkbase.

3.5.3. Show Links

One of the problems that users perceive with Microcosm is that there are inevitably more links available to the user than those that can be seen displayed as buttons. Indeed some viewers are unable to display buttons at all. The user may make selections and ask to follow links, but unless the selection is a perfect match with the linkbase entry, no match will be found. To overcome this problem a *show-link* action allows the user to make a selection, and attempts to identify all link sources within the selected area, so for example the user may select a line of text. Any links with source anchors within that line will appear in the available links dialogue box.

A restricted refinement of this idea is to have a "show generic buttons" option on a fully aware viewer, which will highlight all source anchors, including generic and local links, within the viewer. This was an idea that I implemented in my prototype of Microcosm, and has recently been re-implemented in the Microcosm RTF viewer.

3.5.4. The Computed Linker

The computed linker is invoked when the user makes a text span selection and requests the action *compute links*. It applies information retrieval techniques to attempt to identify text based documents, or sections of documents, that have the most similar vocabulary to the text in the source selection. To achieve this it uses an index of all the text based documents within the current application area, and this index must be created and maintained by a utility program that is provided. The computed linker sends a ranked list of potential destination documents to the available links filter, from which the user may select. Information retrieval within Microcosm is covered in more detail in chapter 4.

3.5.5. Selection

Not all navigation is achieved by selection and action, as not all text queries that one might wish to make actually appear in the data. The selection filter when popped-up allows the user to type a text entry then select an action, such as *Show Links* or *Compute Links.*

3.5.6. The Available Links Filter

The *available links filter*, if installed, will pop-up and display the description of any links which the user may follow (which by default will be the description of the destination document). Whichever link is selected by the user will cause the destination document to dispatch.

3.5.7. Navigational Filters

A range of aids to navigation are available. A *History filter* keeps a list of all the documents that a user has seen in a given session. The user may view the history list (in terms of either the document descriptions or the method by which they arrived at that document) and click on a document to return to it. Histories may be saved and reloaded between sessions: the saved history may be edited by any text editor.

Mimics are guided tours or trails through a given set of documents. These allow one user to show another a route through a set of resources, but the user is still able to diverge from the trail and return to it at will. The easiest way to create a trail is to load a saved history.

A *local maps* filter may be popped-up which will display a small map with the current document at the centre, and links into and out of this document are shown as connections.

3.5.8. Abstraction

We have designed a replacement for the available links filter, which, as well as showing the description of the available link, will also show a graphic representation of the destination document. In the case of a text based document

this will simply be an icon representing a text document; in the case of a picture this may be copy of the picture itself scaled down to icon size. But this technique is best appreciated when dealing with very large digital video files, especially when they are at the end of a busy network: in this case the abstract is a moving icon, or *micon*, which consists of a small number of important frames from the video shrunk to the size of an icon. This abstract of the video may be all that the user needs to see in order to decide that they do not wish to follow the link. The abstract may be held locally, and in any case will be considerably less expensive to download over the network than the full copy of the video.

3.5.9. Monitoring User Interactions

Since everything that a user does in the way of querying and navigating between different documents appears as a message from the filter manager, it is a simple matter to insert a filter that keeps a log of all messages that are sent. These logs may later be analysed to discover the pattern of a users interactions. This is useful both from the point of view of evaluating the hypertext system, and also for discovering what resources they have accessed and what relationships they have discovered.

3.5.10. Filters that Filter

The original derivation of the term "filter" to describe the message handling components of Microcosm lies in the idea that we believed that as text became increasingly linked it would be necessary to apply intelligent filters to remove some of the links in order to avoid overloading the user with information. In practice this situation has not often arisen because of the practice of organising links into separate linkbases, and allowing the users to load only those sets of links that are currently useful or required.

However, the model allows the implementation of new filters that apply intelligence to removing any links from those offered to the user. When a link is created it is possible to add any user defined attributes that are required, in addition to those that are stored by the linker as default. Based upon the attributes of the link a filter may decide whether or not to block the message asking to dispatch the link. For example, we have produced such filters, implemented in Prolog, which apply rule-based techniques to filter available links and to offer only those which are appropriate to the current user's requirements.

3.6. Document Management

Microcosm maintains a database of all files of which it is aware, known as the Document Management System (DMS). There are two reasons for keeping this database.

First, Microcosm knows all documents by a unique identifier. It uses the DMS to resolve unique identifiers into filenames when it needs to locate the file. The reason for using unique identifiers rather than filenames within Microcosm is so that when a file is moved it is not necessary to search through all the linkbase files changing all references to the file. Even if all the linkbase files were on-line and accessible this would be time consuming. Indirecting through the DMS ensures that the only thing that needs changing when a file is moved or changed is the linkbase entry.

Secondly, the database keeps a number of attributes about the document which are not known by the operating system. These are:

1. *The unique reference number*

   This is assigned automatically by Microcosm when it first encounters the file. This is the name by which the file will be known to Microcosm.

2. *The full filename*

   - so that Microcosm can find the file.

3. *An English description of the file.*

   - because C:\mcm\cbio\cbio34.txt tells the average user little about the contents of the file. If the user fails to supply information for this field, then Microcosm will use the filename as the description.

4. *The physical file type*

   Microcosm will use this information to decide which viewer to use to display the document. For example ASCII text files may be divided into two physical types: the first type might use the Microcosm RTF/text viewer and the second might use the Notepad. Default physical types are associated with filename extensions.

5. *The logical type(s)*

   This will describe where in the Microcosm file manager hierarchy the document will appear. A document may have more than one logical type. The Microcosm file manager allows the user to assign the logical type by dragging and dropping into the appropriate folder in the same way as the Windows file manager.

6. *The application(s)*

   A Microcosm user will normally logon to use one application (or project) at a time. An application defines the set of files, linkbases and other filters that will be available. A file may be part of more than one application. By default the application will be the current application at the time the file was first used by Microcosm.

All the above attributes are required, and if not provided by the user then the system must provide defaults. The remaining attributes are not essential, but may be useful.

7. *The author(s)*

   This differs from the owner (which *is* known by the file system). The reader may wish to investigate all documents *created* by a given author.

8. *The keyword(s)*

   Users may assign keywords to a document. It will then be possible to retrieve all documents with the given keyword. These keywords may then be used as a useful source for the automatic creation of generic links.

9. *The abstract*

   As mentioned in subsection 3.5.8, it is possible to attach abstracts (such as micons) to a document so that when the document appears as the destination of a link, the abstract will appear. This field will contain the filename of the micon or icon to be used as the abstract instead of using the default which will be created automatically, on the fly.

10. *User defined attributes*

    Users may add their own attribute fields then add their own contents to these fields.

Access to the files that the system knows about is via either the Microcosm file manager, or by using a database query tool that allows the user to make Boolean queries based on file attributes. New files may be introduced to Microcosm by dragging them, one or more at a time, from the Windows file manager onto the Microcosm file manager, or by simply accessing the file from the Microcosm file manager's file browser option.

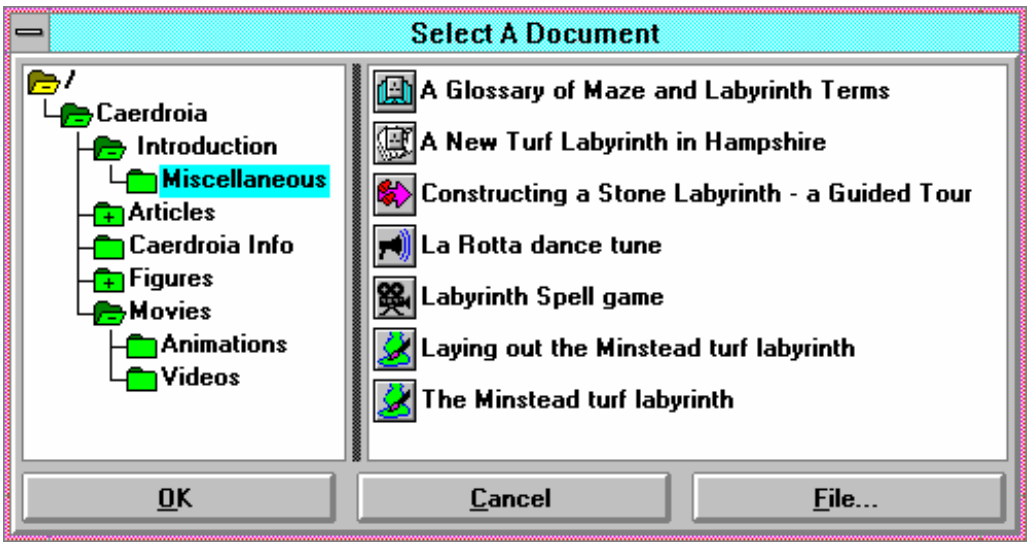

Figure 8: The Microcosm file manager/browser, showing logical types and descriptions.

3.7. The Registry

Since Microcosm is highly configurable it is necessary to maintain a large amount of information about the configurations that are required. This is achieved by maintaining all the information required in a database known as *the registry*. Any component with appropriate permissions may write to the registry, and all applications may read from the registry. The information that is maintained in the registry is typically the class of information that is kept in INI files, but this approach was abandoned due to the hierarchical complexity of the information that needs to be kept.

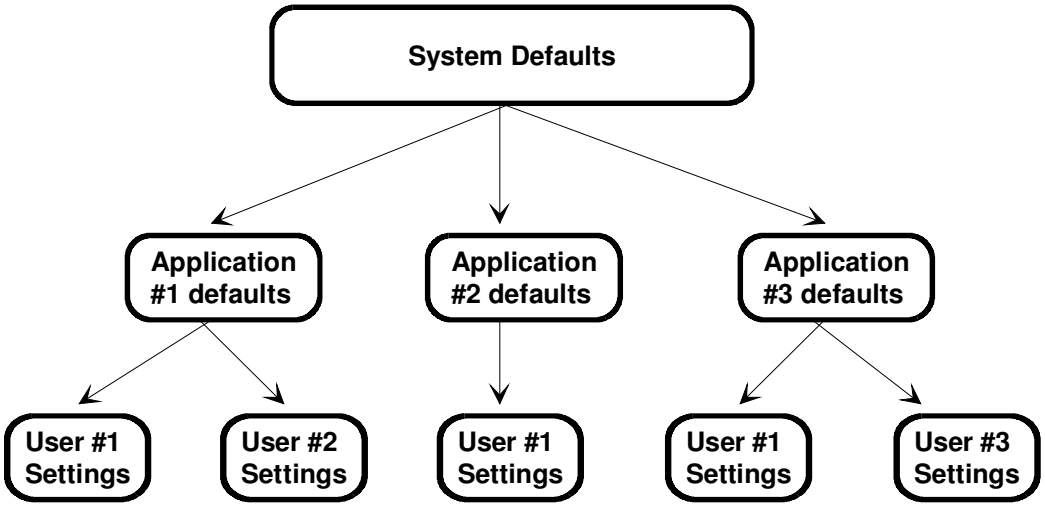

Figure 9: Hierarchy of Settings maintained by registry.

Typical information that is kept in the registry includes the list of filters that are available, the default viewers to use for various data types, the filters and their default settings when the system starts, the list of registered users and their permissions, the font sizes, colours and window sizes for various files, and any other information that might normally be kept in an INI file.

Figure 9 shows how the registry decides which settings to use. There are various system defaults, set by the system manager: these may be overridden or added to by each application that is in use, and then each user may override or add to those settings to arrive at a personalised environment.

Any program may query the registry, for example a viewer may query the registry to enquire which filters are currently installed, and thus build a dynamic menu of actions available: users with appropriate permissions may change entries so that they may maintain their own personalised environment for a given application.

3.8. Navigational Methods and Support

In the early literature of hypertext there was much concern about the issue of "getting lost in hyperspace" (Nielsen, 1990)(Wright & Lickorish, 1990). However, subsequent researchers have now come to the conclusion that this effect was largely due to the fact that link following was virtually the only navigational method in the early systems (Bernstein, 1993). There is little more frustrating than knowing that a particular node exists, and even knowing its subject or title, but being unable to find it because you cannot remember the *route* by which you got there: this is clearly nonsensical. The solution to the problem is to provide multiple navigational methods, so that the user is able to browse information, easily locate new information, recall information that has previously been visited, and follow trails through information left by others. Microcosm provides three levels of access to new information in addition to those methods that involve using historical data, as shown in figure 10.

In general, different methods of navigation will find differing qualities of information: following a button or specific link will take you to a specific piece of information which someone manually authored exactly because they decided that the relationship between the two items of information was important. Such links have a very high *quality*, but there may not be many such links since they require such a high authoring effort: they may be low in *quantity*. On the other hand a generic link may be followed from any place that a particular source anchor occurs. Taking the example in figure 7 in subsection 3.5.1, there is a generic link on the word "kings" which offers the user a file about the House of Windsor, which would generally seem to be a useful connection. However, if the user had been reading a file about the game of chess, then this relationship would have been inappropriate. The generic link once created will be available from many places, but the quality of the relationship discovered may be lower because of the generality of the link. In the extreme, if the user had used *compute links* on the word "kings" the result may

have discovered relationships with files about aspects of royalty, chess, pop stars (Elvis?) and articulated vehicles (kings of the road?). Computed links will be available from any piece of text, so the quantity of such links is effectively infinite, but the quality may be low, requiring the user to decide whether the relationship discovered is actually useful.

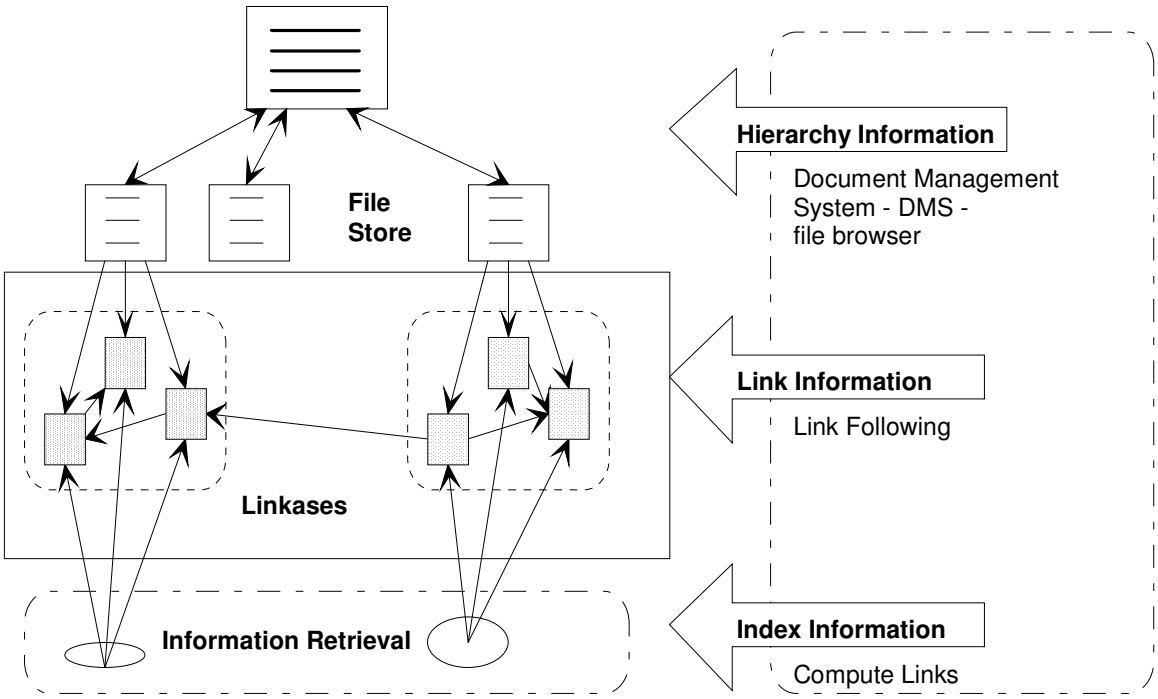

Figure 10: Levels of access to information using Microcosm.

Hierarchically organised file structures and folders have traditionally been the most significant method of ordering and retrieving information. A modern desktop computer might typically contain 750 MB of filestore containing a few thousand files, and quite possibly be connected to a file server offering another gigabyte or so, with tens of thousands of files. So long as the user has a good mental model of the organisation of such filestores they have usually not had difficulty in browsing to discover information as required. No hypertext system should lose sight of this. Hypertext should be seen as adding value to information management, rather than as a replacement to methods that have already been shown to work.

Briefly, in summary, navigation in Microcosm may be through the following methods:

- The Microcosm file browser (figure 8, section 3.6)
- Buttons (figure 5, section 3.3)
- Specific links (figure 7, section 3.5)
- Local links (figure 7, section 3.5)
- Generic links (figure 7, section 3.5)
- Computed links (chapter 4)
- Document to document links (subsection 3.5.1)
- Boolean query on document attributes (section 3.6)
- Mimics (subsection 3.5.7)
- History (subsection 3.5.7)
- Local maps (subsection 3.5.7)

Limited user studies delivering the same set of resources in both Hypercard and Microcosm (Hutchings, 1993) have indicated that Microcosm is successful in avoiding the "lost in hyperspace" syndrome. Users tend to follow a small number of hypertext links at a time, returning regularly to the file browser to reorient themselves and to begin a different line of enquiry.

3.9. Authoring

Hypermedia materials may be constructed either by manual creation of the nodes and links, or automatically. Automatic methods depend upon utilising mark-up that exists within the document(s). An example of an automatic system was Lace (Rahtz et al, 1990) which used Latex mark-up within a document to create a hypertext where nodes were subsections of the original document, and the contents, footnotes, cross references and index were automatically turned into buttons which related the user to the appropriate node within the document. Dynatext performs a similar function with documents marked up with SGML. Another example is the Windows Help system which compiles RTF exported from

Word for Windows to generate hypertext help files. Such systems have an important role to play in allowing text which was perhaps originally marked up for printing as a book to be converted into hypertext versions of the book. But that is all they will be: hypertext versions of the book. Large hypertexts will be created by creating and importing information from many different sources and data formats, and such hypertexts will inevitably require a degree of manual creation.

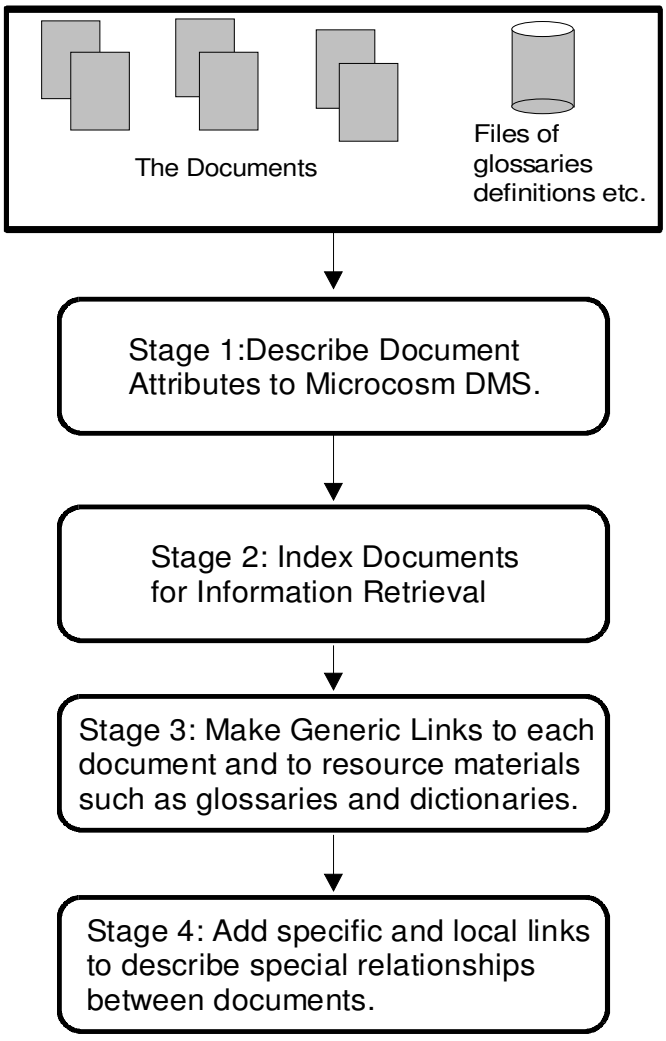

Figure 11: The stages in building a resource based hypermedia application.

Hypertexts are created for many different purposes. Some will be small in size and contain a very rich set of links: these require considerable authoring effort. Others will be very large, possibly containing very large nodes, and because of the cost of authoring, will possibly have a much sparser set of links. The goal of system

developers is to offer the user the greatest degree of information discovery and navigational facility at the least cost to authors. This is the issue Microcosm has addressed.

The Microcosm philosophy is that of taking a *resource based* approach to hypertext application building (Davis et al, 1993). The resources are collected on the machine in whatever format they were created, then on top of this we add facilities, such as generic links, document attributes, and information retrieval indexes (see figure 11) which enable the document to be *located*: they give the document *pull*. Finally authors and users will start to navigate the information space, and as they deduce relationships that they wish to keep or point out to others, they will start to add a layer of local and specific links on top, which they may keep in their private linkbases, or publish for others to view. Furthermore they will create or add new documents to the system, and again these might be private or published. These final stages of adding links and new documents are the most labour intensive, and a typical link making action is shown in figure 12. An important advantage of adopting this approach, is that simply by importing the documents to the system as described above will provide the user with a sufficiently powerful hypertext system with which to begin exploring the information space.

The advantages of the Microcosm approach to hypermedia authoring may be summarised as:

1. *Reduced authoring effort.* Much of the work is achieved by generic links and information retrieval.

2. *Third party applications may be included.* This is because Microcosm does not attempt to store its information within the application data.

3. *Documents and linkbases may be re-used within multiple applications.*

4. *Multiple webs of links are possible.* Each web is stored in a separate linkbase, and multiple linkbases may be installed at run time.

5. *Private links and data may be maintained without making changes to the original data.* Users may keep linkbases and data in their private workspaces, so no-one else will even be aware that they exist.

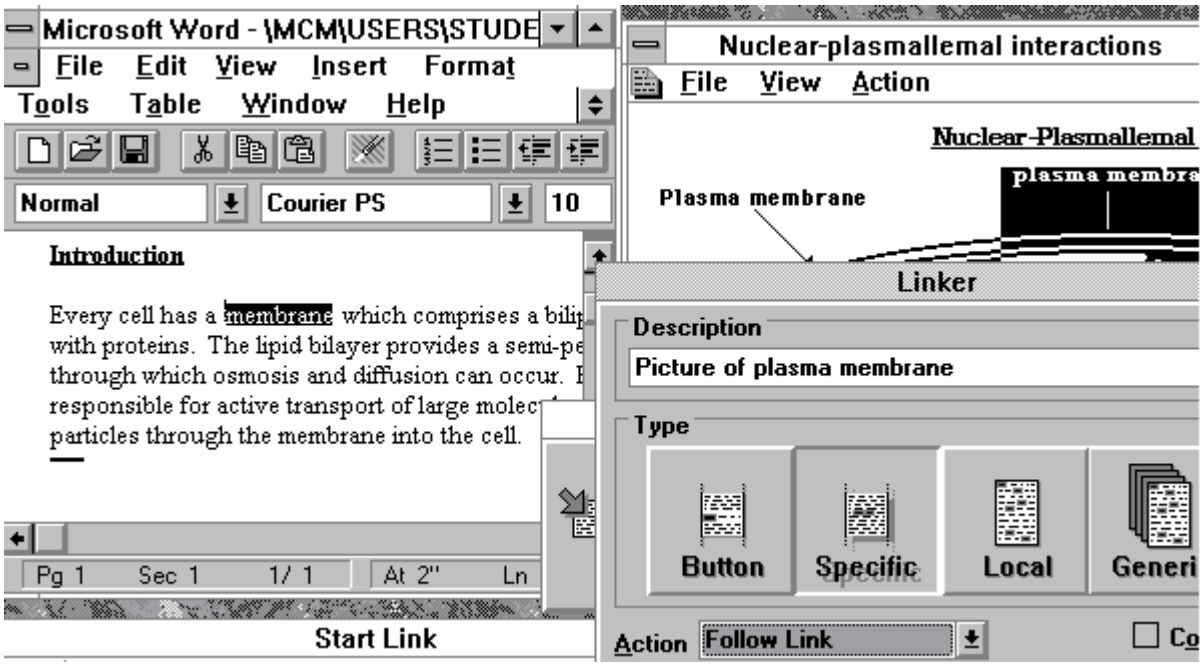

Figure 12: Making a Link in Microcosm. The source of the links was the text span "membrane" within a Word for Windows Document. The destination is part of a bitmap picture. The Author has described the link, has set the link type to Specific, and will press "OK" to complete the link.

3.10. Summary

This chapter has endeavoured to describe the components of which Microcosm is made. We may now reflect on the question "what is Microcosm?". The answer is that Microcosm is an *open hypermedia link service system* which lends itself to a *resource based approach* to the collection and navigation of materials, and as such might be described as an *information management tool*.

As a link service, Microcosm provides a message passing framework which allows users to interact with an application, and from this application make queries and request actions which are passed through a set of link service aware tools (filters) which have the opportunity to service this request. The result of such an action will generally be the presentation of some new information to the user.

When this link service is populated with a set of appropriate viewers (whether Microcosm aware or not) and filters (such as linkbases, information retrieval tools and navigation tools) we have an open hypermedia system.

Perhaps a helpful way of summarising the way that Microcosm may be used as a resource based open hypermedia system or an information management system is to compare it with the way we use a standard library as shown in table 1.

But like any analogy, the comparison with the library is imperfect as there are some things that you may do in Microcosm that are not possible in a library. In Microcosm you may make use of information retrieval techniques to perform a powerful set of searching tasks; you may browse through data at a very much faster rate than is possible when you must physically collect resources; you can add your own resources to the collection (should people wish to include your links and resources within their "view" of the system); you can use dynamic and computational resources; you can provide pre-set routes to travel through a set of resources: you can keep a history list of everything you found, and how you found it, automatically; you can provide users with connections between resources; you can put the same resource (or at least a link to that resource) in as many places as you like.

Function In Library	Equivalent in Microcosm
Enter new resource into catalogue	Describe new resource to Microcosm's Document Management System (DMS)
Decide which classmark to file resource under.	Assign resource to one or more projects, and within projects assign logical type or types
Put resource on shelf	Put resource on computer file system
Use card index or on line system to find out about resources	Make structured Boolean query of DMS
Use abstracting journals or citation indexes to locate new resources	Use Information retrieval techniques to locate articles about appropriate subjects or mentioning relevant authors
Search shelf for book	Use Microcosm file manager to search logical hierarchy to find resource
Discover that the resource that you want is not there as someone else has got it	This doesn't happen! Users may share resources
Look up term, phrase or name in encyclopaedia, glossary, bibliography or dictionary	Follow generic link to on-line resources such as encyclopaedias, glossaries, bibliographies or dictionaries
Follow reference from one article to another	Follow specific link
Photocopy an article	Print an article
Make notes on private photocopy of an article	Make private link to annotation on article.
Search for resource in other libraries	Widen scope of search for information onto Internet servers e.g. WAIS
Make Interlibrary loan request	Read remote resources using viewers of Internet Servers e.g. WWW

Table 1: Comparing Microcosm with the functions of a traditional library.

Chapter 4. Information Retrieval and Computed Links

The idea of integrating information retrieval techniques into hypertext is not new (Bernstein, 1990), and there is no doubt that statistical information retrieval techniques are still more effective than techniques that rely upon semantic analysis (Hutchings, 1993). The problem with many traditional information retrieval packages, such as Status, is that they rely upon the fact that the documents in the information base have been marked-up to highlight structure and keywords, and they rely upon the query being delivered in some kind of structured query language. In order to integrate information retrieval seamlessly into Microcosm we needed a system that would work without requiring the documents to be marked up, and without requiring the query to be structured, so that *any* selection of text could be treated as the query. This chapter describes the technique used, which is based upon techniques attributed to Salton (Salton et al, 1975). The work described in this chapter was carried out by Dr Z. Li, under my supervision, as part of his PhD (Li et al, 1992) (Li, 1993).

Prior to using the computed linker it is necessary to produce an index of the stems of all the significant words in the text documents known to the system. Microcosm maintains a list of all the text documents, and supplies an editable list of stopwords, which contains around 250 of the most popular words in the English vocabulary: these words are removed before indexing takes place. The remaining words are reduced to a common stem using a well established algorithm (Lovins, 1968). The indexer now builds a table which represents the occurrences of each word as shown in table 2. The actual numbers are normalised so they represent the occurrence of each word as a percentage of the total number of indexed words in the file. Note that, for example, the words "compute", "computer" and "computation" will all reduce to the same stem.

Once the text documents have been indexed, the user may install the computed linker with the correct index, then make arbitrary length selections (queries) from any text based viewer, or type into the *Selection* dialogue box, and ask to *Compute Links*. The computed linker will take the query, and apply the same initial stages as the indexer, namely it will remove stopwords from the query and reduce the

remaining words to their stems, and normalise the frequency with which they occur with the length of the query.

Stem of Word	File #1	File #2	File #3	File #4	File #5
compil	0.04	0	0	0	0.02
comput	0.03	0	0.01	0	0.04
disk	0.03	0.01	0.02	0	0.03
print	0	0.01	0	0	0.03
program	0.04	0.01	0.04	0.02	0.02

Table 2: Representation of part of table produced by indexer.

The next stage is to attempt to identify the extent to which the documents in the index share the same vocabulary as the query. This similarity function is achieved by treating the query as an *n* dimensional vector and extracting from the index another *n* dimensional vector (for each document) each of which contains weights representing the frequency of each term. The angle between two vectors is measured by the following formula, and the smaller the angle, the closer the two vectors are.

$$\cos\alpha = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2 \sum_{i=1}^{n} b_i^2}}$$

n is the number of terms in the query.
a_i is the normalised frequency of occurrence of term *i* in the query
b_i is the normalised frequency of occurrence of the term *i* in the document.

The computed linker now sorts the documents in the index into order of proximity to the query.

Finally the computed linker must decide how many of the documents identified by the index should be offered to the user. Since the similarity function does not produce normalised similarity values, there is no fixed threshold that can be used to decide whether a document should be included, so the following rules are designed to produce a dynamic threshold.

1. Calculate the average value of the similarities of all documents. Use the average value as a sub-threshold; only the documents whose similarity values are above the average value should be passed to next step.

2. To avoid passing on too many documents, another sub-threshold whose value is defined as *M* is used to control the maximum number of destinations. But a document collection containing a large number of documents may have more documents relevant to a query than a small document collection, so another sub-threshold whose value is defined as percentage *P* is used to control the maximum number of destinations of a computed link.

3. The documents are ranked according to their similarity values in decreasing order. Then the *max(M,P)* documents that have larger than average similarity values are chosen as the destination documents.

Default values of *M=10* and *P=10%* are used, so for a collection that has 35 documents, a maximum of 10 destinations would be permitted. For a collection that has 200 documents, a maximum of 20 documents could become destinations.

The algorithm as described above allows the user to make a selection and quickly identify the documents that have the most similar vocabulary, but there is no guarantee that every word in the query will also be in any given destination document: nor is there any mechanism for making structured queries of the sort

> "Find me all documents that contain the word ‘language’ and either the word `compiler' or the word ‘interpreter’ but do not contain the word ‘foreign’."

To make queries of this sort requires a quite different algorithm for matching queries, and more importantly would require a special interface, rather than integrating with the selection and action metaphor that Microcosm supports.

There are various improvements that have been made to the system.

1. *Phrase Indexing.*

   If the phrase "laser printer" occurs in both a query and in document A, then one might say that Document A has a higher similarity than document B which contains the words "printer" and "laser" but at different places. Phrase

indexing involves attempting to identify short phrases in the documents and in the query, and treating the phrases as if they were words, adding them to the indexes and vectors. The phrase will now be "double accounted" in the similarity calculation, since the component words will be counted as well. Experiments have shown that this approach has improved retrieval accuracy (Li, 1993).

2. *Linking to document sections.*

The algorithm as described above identifies only whole documents as link destinations. If the document is large it may help if the computed linker points the reader to the best section. If this option is selected the computed linker will dynamically take each document identified and break it up into sections, and then by dynamically indexing each section will offer the user the sections in order of proximity to the user. When the link is followed the viewer will load the document with the section identified at the top of the window. What constitutes a section is under user control, but typically a paragraph or a number of characters will be used.

3. *Document to document links.*

The above algorithm describes a method of finding the similarity between a relatively short query, typically between one word and one paragraph, and other text documents in the *docuverse*. If one wishes to calculate which documents have the closest similarity to the current document, then the vectors become rather large and the time taken to do the calculation increases significantly. An alternative approach is to use only a few words which occur in a given document but do *not* occur much in other documents as a signature for the document. Comparing these signatures is much faster than comparing the entire vocabulary used.

4. *Data Formats.*

Clearly the algorithm described requires access to the ASCII representation of the text in a document. We are currently experimenting with using installable libraries that extract ASCII text for indexing from various text based application data files such as Microsoft's Word for Windows DOC files and Rich Text Format (RTF). Once the text has been extracted it is simple to index

it. It is harder, however, to identify document sections dynamically for indexing and to dispatch the applications at the correct section. Currently this only works with RTF.

The Microcosm computed linker is a tool that users, particularly application authors and readers of large bodies of text have found extremely useful for discovering relationships between documents in the hypertext. The algorithm was optimised for speed at query time, and in general the time taken to locate relationships by this method is actually faster than queries to the linkbase. The selection and action mechanism makes for a very easy to use interface, which is much quicker than typing queries (which is of course also possible through the *Selection* filter). The success of this approach is all the more rewarding, since the original software was not developed for use with Microcosm, and was adapted for this purpose at the source code level in the space of half a day.

However, the computed linker has not gone without criticism. Many users confuse information retrieval with search functions such as *grep*, and are then surprised when the system returns documents that do not have exact matches with the whole string in the query. Users often ask why we do not highlight the terms from the query within the documents when they are displayed. This would be difficult to achieve since the system does not know what the terms were - only the stems of the terms. It would be a time consuming task to identify whether each word in the destination document was one that would stem to one of the stems of the words in the original query.

Perhaps a more serious criticism of the computed linker is that it makes no use of the information that the system has about the structure of the hypertext. For example it is not possible to ask to view all documents that have links into them from documents that contain a given phrase. This would be a different tool, which would need to have greater access to the stored Microcosm information, and would need a much more complex interface. At present we have had no pressure from our users to produce such a tool.

Chapter 5. Application Integration

As explained in section 3.4, it is possible to integrate third party applications so that they may be used as Microcosm viewers. This chapter explains how this may be achieved and examines the quality of hypertext system that results. The Universal Viewer work described in this chapter was carried out by Simon Knight, under my supervision, as part of his PhD studies, and has been reported in Davis et al. (1994b)

There are a number of levels at which system builders have attempted to integrate applications as data viewers for hypertext systems. In increasing order of generality they are:

1. *Tailor Made Viewers.* These applications are written specifically for integration with the hypertext system. Microcosm's RTF and Text viewer was built this way.

2. *Source Code Adaptation.* Where the source code for an application is available it is possible to add the code for communicating with the hypertext link service. For example, we are currently adapting the source code of Mosaic (NCSA Mosaic, 1994) for Windows to become a fully aware Microcosm viewer.

3. *Object Oriented Re-Use.* A basic hypertext viewer class is created and viewers for specific data types inherit from this class. HyperForm (Wiil & Leggett, 1992) and the KHS (Aßfalg, 1994) mail viewer are examples of such systems.

4. *Application Interface Level Adaptation.* Many packages provide flexible interfaces and macro programming languages, via which it is possible to add Hypertext functionality.

5. *Shim or Proxy Programs.* These are programs that sit between the hypertext link service and the viewer, translating actions in one system to actions that the other can understand. The Microcosm Universal Viewer described later in this chapter is such an example.

6. *Launch Only Viewers.* This is the worst case, when all the hypertext system can do is to launch a given program with a given data set, but from the program

there will be no hypertext functionality. In Microcosm we frequently launch the Microsoft Media Player in this mode.

One of the factors that has made it so difficult to integrate existing hypertext systems with desk-top applications has been the adoption of the model requiring that the content viewer/editor is responsible for handling link anchor identifiers. Adapting existing applications to do this involves considerable enhancements which will normally only be achieved by adding code at the source code level as described by Pearl (1989) and Shakelford et al. (1993), although it may be possible to add this functionality using a macro language where this is sufficiently powerful. For example, EMACS has a lisp like programming language with which one can actually tailor the environment and such work has been described by Rizk & Sauter (1992) and Shakelford et al (1993). However, users are unlikely to wish to invest the effort to tailor their applications, and software houses are not likely to wish to add link anchor handling facilities until some standard emerges for doing so.

5.1. Links, Anchors and Persistent Selections

For those hypertext systems which handle anchors that reference objects within the node, there are three approaches to handling the anchors (Davis, 1995).

The first approach is to embed all the information about the link within the node content at the hotspot: this is the approach taken by a number of the earlier hypertext systems. The World Wide Web also embeds its links as *html* mark-up. The advantage of this approach is that the document and its links form a self contained object which may be moved and edited without damaging any of the links that are contained within the document.

On the other hand moving (or deleting) a document involves checking all references to this document within all other documents. In the case of the World Wide Web this involves checking all other *html* documents in the world. Failing to make this check results in dangling links. Furthermore, it is not possible to insert mark-up if the data is owned by another user or is on a read-only media such as CD-ROM.

The second approach is that adopted by classic Dexter systems, which generally require the node viewer application to handle the anchors, as shown in figure 13.

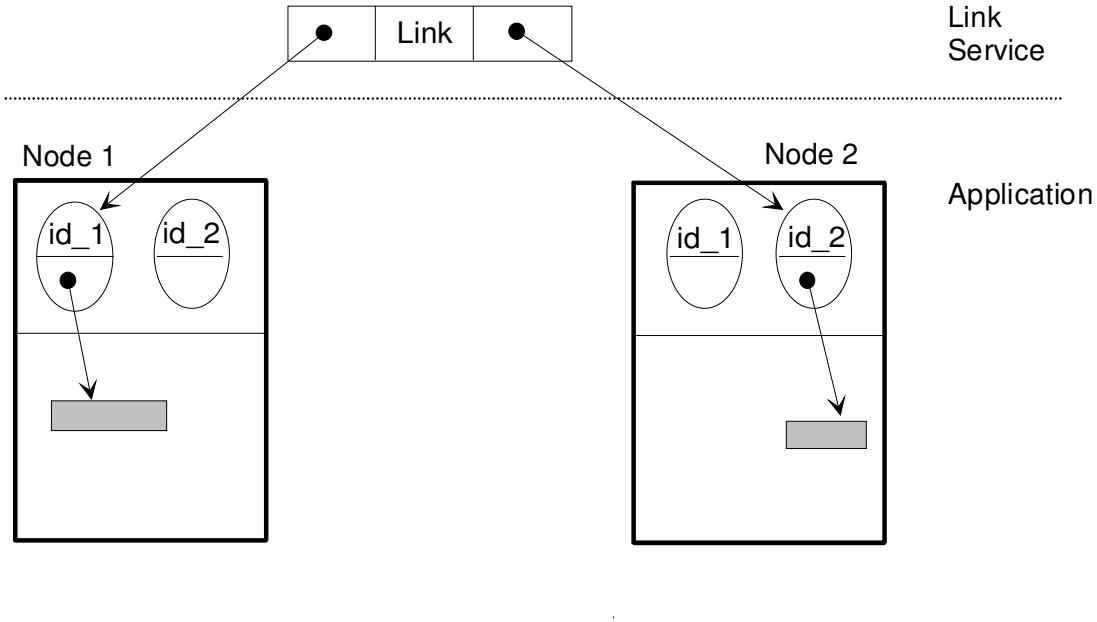

Figure 13: Anchor handling in classic Dexter Systems.

Dexter defines anchors as objects which have a unique identifier and a list of one or more references to something within the within-component layer. These references might be, for example, byte offsets or pointers into an array, or scripts to be interpreted by the application. However, whatever they are, at the end of the day they must resolve to a pointer into the node data, in such a way that the end point of the anchor may be displayed by the application displaying the node. Such a pointer is known as a *persistent selection.* Most Dexter systems require that the application is responsible for effectively maintaining a set of anchors, and their corresponding persistent selections. This may be achieved by embedding mark-up within the application, or by coding the application to handle an external anchor table. Since all the information about the link structure is held centrally it is possible to build tools which analyse the links, for example, network browsers and tools to identify dangling links.

A variation on this approach, which is taken by many of the hyperbase class of systems such as SB3 (Leggett & Schnase, 1994), says that anchors should be entities which belong to the link service as a whole, and are globally allocated, rather than belonging to the specific file. In such systems, the anchor will point to a file (or list of files), and the viewer application will maintain a table which maintains the association from the anchor to the persistent selection, as shown in figure 14.

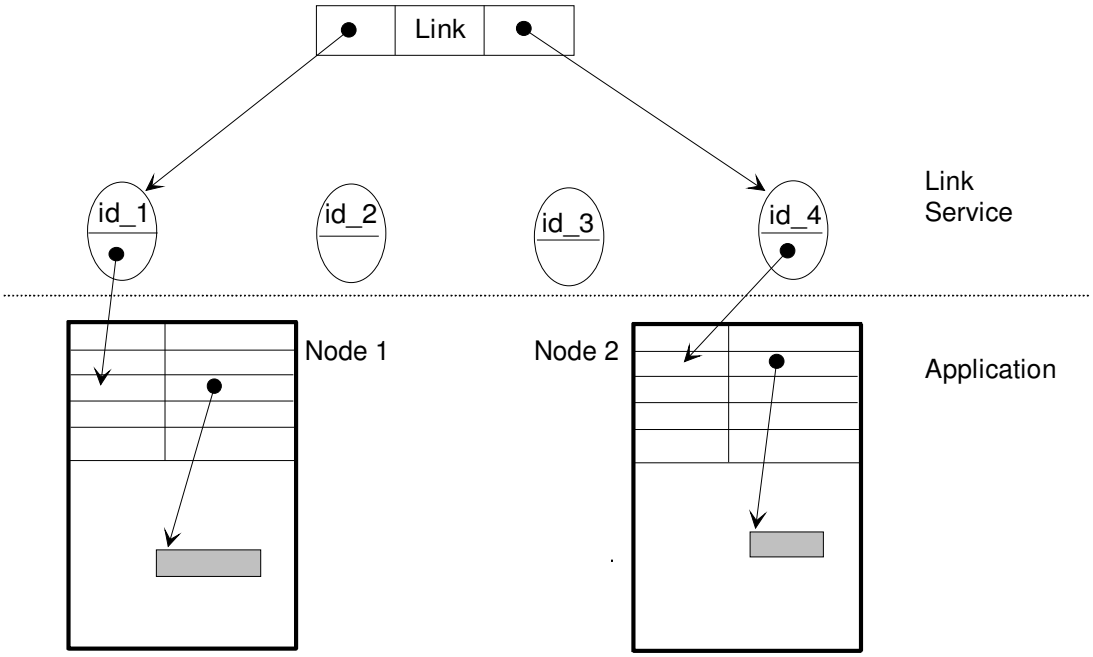

Figure 14: Anchors in Hyperbases.

The advantage of this approach is that, since each anchor is a globally available entity, it is much easier to manipulate the hypertext independent of the content data. It still requires that the application handle the association of the anchors and persistent selections, which requires some coding at the application level, but the onus on the application code is lower, particularly as the code may rely to a large extent on calling methods in the hyperbase application's API.

The final approach is that taken by Microcosm, which is to hold all the information to describe the anchor and its corresponding persistent selection within the link as shown in figure 15. This scheme is the only solution that avoids putting foreign mark-up into application data, and has all the advantages described in section 3.2. However, it also introduces a new problem which occurs when the node content is edited, since there is no explicit binding between the application data and the link structure: this topic is one of the major subjects of this thesis.

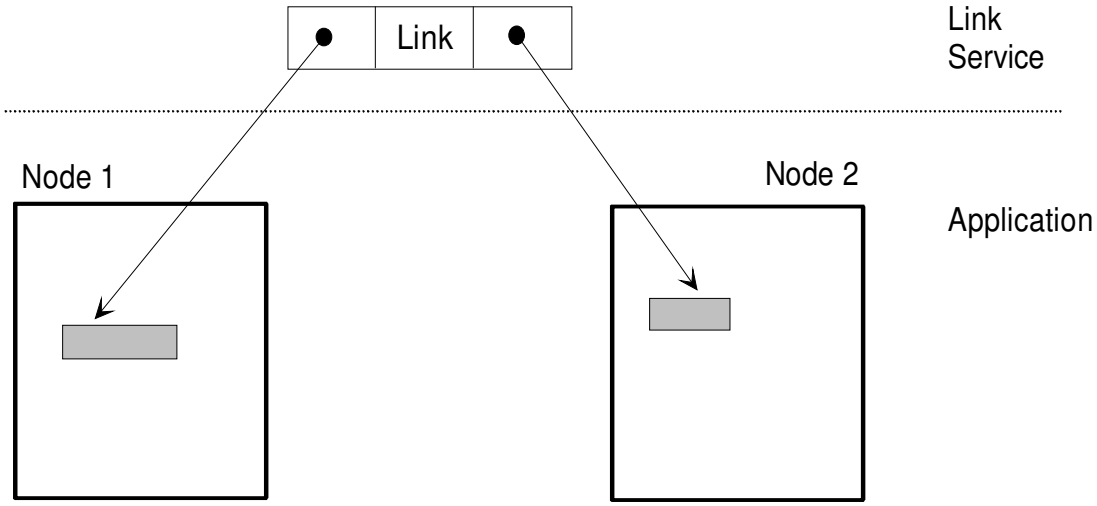

Figure 15: Anchors in Microcosm

Strictly speaking the Microcosm system does not have anchors, since there is no identifiable separate object in the link service which may be identified as an anchor: it has links which contain persistent selections. However, in Microcosm terminology we have used the term anchor to refer to that persistent selection at either end of a link.

5.2. Links and Messages in Microcosm

At this point it is worthwhile investigating in greater detail how Microcosm links work, and how they are stored and activated.

Any viewer may offer the user a Microcosm menu. When a user selects an action from this menu, the viewer will package up a message containing *at least* an *action* field and a *selection* field, representing the users choice of action from the menu and the current selection, if any. The viewer may also elect to put other information into the message, usually including the name of the current document and possibly the position of the current selection within the document.

This message will be sent to Microcosm which will send the message through the filter chain, so that each filter that is able to service the chosen *action* has the opportunity to respond to the message. For example, a user may wish to be

provided with dynamically created links from the computed linker described in the previous chapter, in which case the user will select some text, e.g. “construction of stone labyrinths” and choose *Compute Links* from the menu. In this case the viewer will form a message containing:

\Action COMPUTE.LINKS \Selection construction of stone labyrinths.

Since the Computed Linker is the only filter which understands the action COMPUTE.LINKS, this will be the only filter which will handle this message, and it will use the *selection* as its query.

The *linker* is a filter that will operate on messages containing the actions, MAKE.LINK and END.LINK, which will typically be sent by viewers when the user identifies source and destination anchors. The linker provides an interface through which the user may identify which source and destination anchors are to be joined, and when this happens, it creates a new message with the action CREATE.LINK, and which contains all the other fields that were in the source and destination anchors.

The *linkbase* is a filter that understands, amongst others, two messages; CREATE.LINK, which instructs it to store all the fields held in the message into the link database, and FOLLOW.LINK, which instructs it to treat the fields in the message as a query, and to attempt to locate appropriate links, if any, in the associated link database.

When a link is stored in a linkbase, it is represented by a set of tagged fields. e.g. The set of tags below describes a link form the text "SPC" in a text file (which the system knows by a unique document identifier) and occurs at 214 characters through the file.

```
\SourceFile 100.02.24.93.11.39.54
\SourceSelection SPC
\SourceOffset 214
\SourceDocType TEXT
\DestFile 100.02.24.93.12.40.49
\DestSelection manual
\DestOffset 312
\DestDocType TEXT
\Description SPC Definition
\ButtonAction FOLLOW.LINK
```

A user might later select the string “SPC” from within this file, at the position 312 character through the file, and ask to follow links. In this case a message would be sent as follows:

```
\Action FOLLOW.LINK
\Selection SPC
\DoctName 100.02.24.93.11.39.54
\Offset 312
```

The Linkbase, on receiving this message will apply the following algorithm:

1. Locate all links in the linkbase where:
   \Selection = “SPC” and \SourceFile = “100.02.24.93.11.39.54” and \SourceOffset = “312”
   or
   \Selection = “SPC” and \SourceFile = “100.02.24.93.11.39.54” and the link contains a \LinkLocal tag,
   or
   \Selection = “SPC” and the link contains a \LinkGeneric tag.

2. For each of the links located above, create a message to carry out the action indicated by the \ButtonAction field. Where the ButtonAction is FOLLOW.LINK the action in the message will be DISPATCH, and will contain the description and details of the destination document.

The available links filter will trap these DISPATCH messages, and offer the documents to the user.

Note that offsets here are described in terms of bytes through the file, but this is not pre-requisite. The semantics of the offset are decided by the viewer and can be resolved by the linkbase so long as the method for expressing the offset is used consistently by the viewer for this data type. So for example, a viewer that contains named objects may choose to put the name of the object into the *offset* field, and a viewer that uses a search engine to locate anchors may chose to put some surrounding text into this field.

Note also, that a link need not resolve to a fixed anchor within a document. The (inappropriately named) \ButtonAction field within a link may contain some action other than FOLLOW.LINK, in which case the linkbase will simply package the link up as a message for some other filter to interpret. For example, it is quite common to make a link where the \ButtonAction contains COMPUTE.LINKS, in

which case the action of following this link will be to send a query to the computed linker to dynamically create links using the selection as the query.

Microcosm's very simple method of storing and retrieving links makes it possible to devise a method for storing link anchors to point to almost any sort of data, and makes it possible to extend the system to make any action occur when a link is followed. A record in a linkbase in Microcosm should be seen as a description of the conditions that must be met for a link to be available from any particular selection, and the action that should be taken if this link is available.

5.3. Viewer Communication Protocols

A fully aware viewer is one that can participate in all the Microcosm communication protocols. However it is possible to participate in only a subset of the complete set of protocols. This section informally describes the Microcosm protocols and examines the consequences of ignoring them.

1. *Launch document*

   It must be possible to execute a viewer program with a given data set loaded from external code. This protocol is prerequisite in order to provide destinations to hypertext links as programs that do not allow their data file to be named on the command line do not make suitable viewers.

2. *Display Buttons*

   A fully aware viewer will, as soon as it starts, send a message back to Microcosm asking for details of any buttons (specific links that are to be highlighted in some way): Microcosm will respond by sending back the button information which the viewer will use to repaint the screen. However, if the viewer does not request buttons, the information will not be sent, so buttons will not be painted. This does not mean that the links are unavailable, but simply that the user is made responsible for selecting the source anchor and choosing *Follow Link* from the action menu. This means we might have hypertext nodes without buttons, a viewpoint that might please some (Hall, 1994), but others might feel that it is at times necessary to indicate to the user what options they might have in order to continue browsing.

3. *Start-Up Options*

When a viewer has started-up it may be desirable to indicate the destination anchor in some way. In the case of a text document this might mean scrolling the document so that the anchor text is on the top line; in a video this will mean moving to the correct frame; in a drawing this might mean highlighting the relevant object. Alternatively, it is possible to do away with destination anchors altogether and to simply load the given document which will mean that the resulting hypertext is very "chunky", leading to a note card style hypertext, where the destination of any link should preferably be limited to some sensible size.

Also, when a viewer has started-up, it is possible for the viewer to communicate with Microcosm asking for any display details, such as font sizes, window sizes, window positions, background colours, button colours etc. Although these features are all useful, none of them are essential, and have mostly been implemented to overcome authors' concerns about the use of the vanilla Windows interface for delivering materials to naïve users.

4. *Check Link Integrity*

When a viewer displays a document it is desirable to check that any offsets within the document, used to describe link anchor positions, still correspond to the required objects. This may be achieved by requesting that the linkbases provide a list of all links containing specific source and destination anchors within the current document. If these links are dated after the date that the document was last edited then they are safe, but if the document has been edited since any link was created, then the viewer must relocate the anchor (if it can) and update the linkbase.

If the viewer is incapable of entering into this dialogue with the linkbases, then it is possible that anchors might be incorrectly located. Possible solutions to this problem are:

- Make the document read only, so that it may not be edited. If it must be edited, produce another version.

- Do not use specific link anchors. All links should be generic or local, and have their destinations at the top of, rather than within, the destination file.

- Use a search engine to find link anchors rather than offset / position information. This is the approach taken by HyperTed (Vanzyl, 1993): then as long as the anchor is unique within the file, and the anchor itself is not changed, it will be possible to edit the file.

These solutions to the editing problem, and others, are discussed in detail in chapter 8.

5. *Service User Actions*

Once users begin to interact with the system they will make selections and choose actions from the menu (or click on buttons where they are available). The viewer is responsible for

- providing an action menu

- identifying the selection made

- identifying (in any way appropriate to the application) the position of the selection

- identifying the current data file

- packaging all of the above into a Microcosm message and sending it to Microcosm.

If the application is unable to identify the position of the selection, then Microcosm will only be able to provide local and generic links: specific links would be impossible. If the application does not support selections, then only document to document links will be supported.

5.4. Enabling Applications For Hypermedia Use

Many serious applications these days have a built in macro programming language, and some degree of tailorability so that new menu options may be

added. For example Microsoft Office applications have Visual Basic for Applications, the Lotus Office Suite applications have macro languages and AutoCAD has AutoLisp. Using these facilities it is very simple to add the standard Microcosm menu to such an application, then to add appropriate macros for each action chosen. Figure 16 shows a Word Basic macro for the *Follow Link* action, which uses Microsoft Windows' DDE to communicate with Microcosm.

```
Sub MAIN
    On Error Resume Next
    If Selection$() = "" Then
       MsgBox("No Selection Defined!")
    Else
       chan = DDEInitiate"MICRCOSM","LinkServer")
       If chan = 0 Then
          MsgBox("Could not connect to Microcosm")
       Else
         DDEExecute(chan,
            "[FOLLOW.LINK]
            [" + Selection$() + "]
            [" + FileName$(0) + "]
            [0][DOC]
            ")
       DDETerminate(chan)
    End If
End Sub
```

Figure 16: Word Basic Macro to Follow Link.

In this case no attempt has been made to send any source anchor position (zero was sent) which means that Microcosm will only attempt to follow generic and local links. Word may be persuaded to yield position information (in terms of lines and words through the file) but in this case the macros become more complex.

Other packages may be similarly adapted, for example spreadsheets such as Excel may use a range name to describe anchor positions and AutoCAD allows the author to associate a unique name with each object: this name may be used as the anchor identifier for linking purposes. An example of such a hypertext enabled application is shown in figure 17.

Packages with macro languages may also be programmed to launch macros at start-up which alter the current view so that the destination anchor is made explicit (e.g. by highlighting or scrolling to the appropriate line of text.)

Some of the applications we have enabled using this method are Word for Windows, Word Perfect for Windows, AMI Pro, Lotus 123, Excel, Toolbook, Authorware Pro, AutoCAD, Microsoft Access and Microsoft Project.

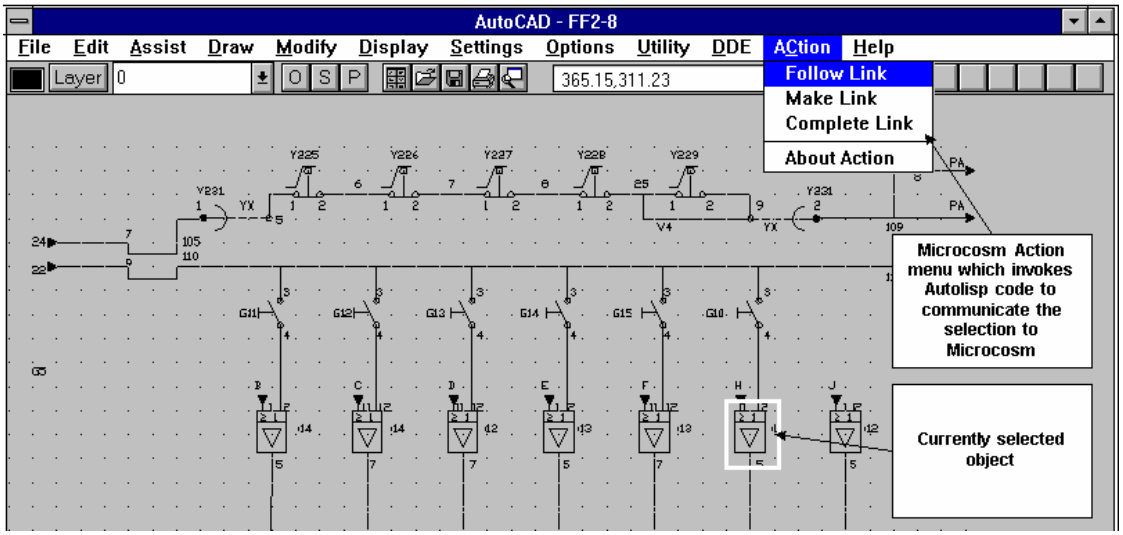

Figure 17: AutoCAD (adapted as a semi-aware viewer) following a link

The Chimera System (Anderson et al, 1994) takes a similar approach to Hypertext enabling third party applications. In the case of FrameMaker, which publishes a set of RPC calls, a proxy or "wrapper" program handles communication between Chimera and FrameMaker, and FrameMaker macros have been used to observe user actions and to communicate them to the wrapper. Multicard (Rizk & Sauter, 1992) has also used macros within editors such as EMACS for hypertext enabling.

5.5. Applications Which Cannot Be Enabled: The Universal Viewer

Perhaps the most interesting result of this research has been the realisation of the power of the hypermedia functionality that it is possible to achieve with applications that have no communication facilities whatsoever other than those that are standard within the host operating environment.

Earlier versions of Microcosm allowed the user to set an option within the Document Control System that told Microcosm what action to perform on any new data entering the clipboard. Thus the user could run *any* program that supported the clipboard, and every time some data was copied from the application, the appropriate action, such as *Follow Link* or *Compute Link* would be performed. This approach seemed very powerful, but the interface to this functionality was so obscure that it was rarely used (selecting *Copy* from the Edit menu is not an instinctive interface to link following!). However, others have emulated this approach, and Cawley et al. (1995) argue that such functionality represents an extra level of viewer awareness that belongs between *shim programs and launch only viewers* on my list at the start of this chapter.

In order to improve the interface and to further investigate this branch of the research we designed a "parasitic" program that we called the "Universal Viewer" or UV, which was implemented by Simon Knight. A list of applications with which we wish UV to co-operate are maintained, and UV monitors Microcosm events in order to identify when a program on this list is launched: when it identifies such an event it attaches a small Microcosm icon to the application's title bar (e.g. see figure 18). Clicking on this icon produces a menu, identical to the action menu on any other viewer. The UV is a "shim" or “proxy” which is responsible for all communication with Microcosm, and handling any communications that are possible with the application.

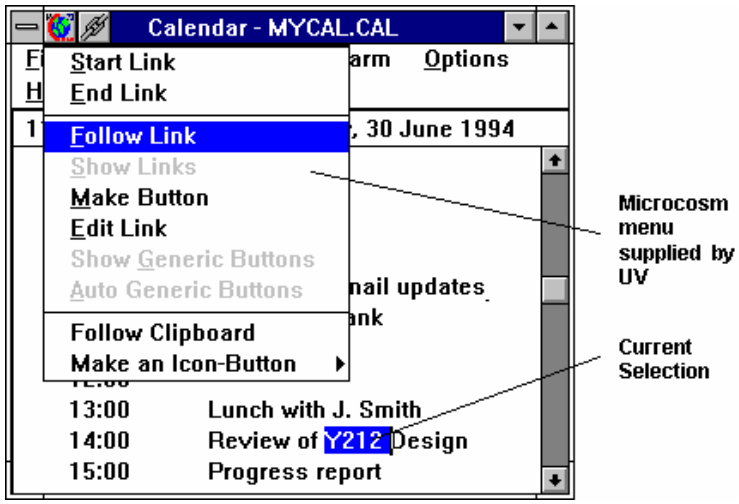

Figure 18: Microsoft's Calendar running under UV. The user is about to follow a link.

A similar approach has been taken by the designers of the ABC system (Shakelford et al, 1993)(Smith & Smith, 1991) who are working with the X-Windows system. They re-parent the Window of the required applications so that the new parent is the Generic Function Manager (GFM). To the user this appears as if the application is running with an extra menu bar inserted between the original application title bar and the application work space. The GFM allows the user to make node to node hyperlinks, but they have not attempted to handle anchors of any kind, unless the application is adapted at the source code level.

In order to co-operate with an application, UV needs to be pre-configured to know a few details.

- When Follow Link (or some other action) is selected from the UV menu, how will UV communicate to the application that the application should pick up the current selection?

  There are a number of options. The most common is to use a UVMACRO which is a list of menu selections, (conceptually similar to a set of keystrokes). Other options are to use DDE (where it is supported by the application), to use a Windows Recorder File (keystrokes again) or NONE, when this is not possible or meaningful as the application has no concept of selections.

- When the selection has been made, how will the application communicate that selection to UV?

  Options are via the clipboard, via DDE, via a named swap file or, again, NONE where this is not possible or meaningful.

- What options will be available for locating a destination anchor when a link is followed to a document in this application?

  The problem here is to change the unaware application into a state where it displays the destination anchor, e.g. by scrolling a text document until the anchor is within view, and then highlighting the text span. To achieve this links may be created which launch a named keystroke macro when the destination file is loaded. These named macros must be pre-defined for each application. A typical such macro is FINDSELECTION, which will take the text of the destination anchor (from the link information) and run the application's own search engine to find the first occurrence of that text. The list of available

macros are offered to the user at the time that a link is created, as shown in figure 19.

In the case where it is not practical or possible to drive the application into some appropriate state to display the destination anchor, then the only option will be to launch the application in its normal start up state. An option which slightly improves upon this worst case is to launch the application and at the same time display a USERNOTE which will be a message to users instructing them how to proceed so that the meaning of the destination of the link is clear.

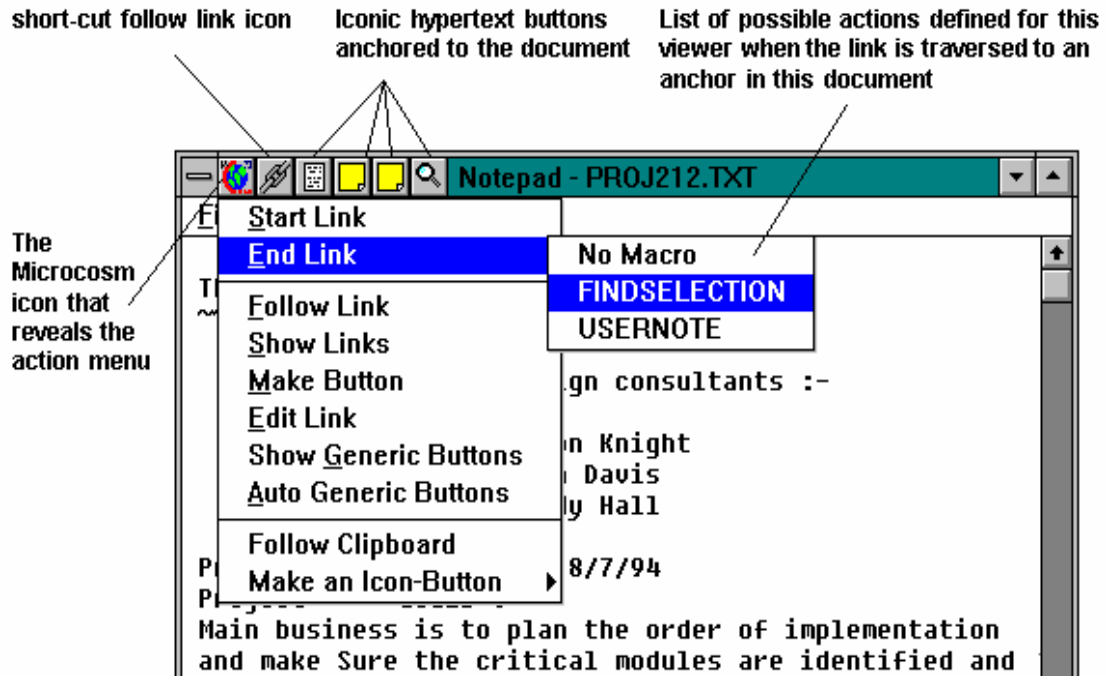

Figure 19: The project Notebook. The User is about to create a destination anchor in this document. In this case the anchor will a particular text string.

The resulting system gives us local and generic links out of many applications, and gives us methods of following links into specific points within application data: however buttons are still missing. In order to generate buttons we introduced the idea of adding small user definable graphic icons onto the title bar. These represent buttons that may be clicked at any time. When the cursor is moved over such a button, text appears describing the link, and putting the link into context, since its position on the title bar gives no clue as to the circumstances under which one might wish to follow this link. Although this facility is limited when compared to highlighting buttons in context, it is still surprisingly effective.

5.6. Further Research

At the time of writing, the Microcosm team are extending the Universal Viewer in three ways.

1. Wrappers and Views.

At present the Universal Viewer cannot be used very effectively with pictures, as it would be necessary for the user to make exactly the correct selection in order for the link to be matched in the linkbase. In order to allow link following from objects within pictures it is *necessary* to have some form of button or persistent selection. A method of achieving this is to introduce a concept of *views* and the idea of a *wrapper*.

These ideas can be best explained by the following analogy. An art gallery might consist of dozens of pictures of different sizes. In order to point out objects of interest to a visitor, the curator might choose to cut sheets of cellophane to the correct size for each picture, and by holding the cellophane over the picture itself, could draw round the interesting objects on the cellophane, and annotate them. The visitor to the gallery would then be supplied with the sheets of cellophane, and by choosing the right sheet to fit on the correct picture could then see the selected objects and read the curator's comments.

In this analogy, each picture is a separate *view* of the total data (the gallery), and the correct sheet of cellophane is a *wrapper* for that view. The equivalent problem for the Universal Viewer is to allow the user to declare that a particular state of an application (window size, data on view, display options and position) is a view to be remembered. Then when persistent selections are made in this view, they are stored in the linkbase. A user wishing to follow links from the application must first ask the Universal Viewer to return the application into one of the stored view states, at which stage any selections will appear over the top of the application.

The difficulty with this approach is knowing exactly the state of the application at the time that the author declares a view. In the simplest case, such as a simple bitmap viewer, it would probably suffice to store the current window size and

scroll bar positions. However an application such as a spreadsheet may allow a given data set to be viewed in a number of different ways (datasheet, bar graph, pie chart etc.) so it would be necessary to know something of the history of the user's interaction with the application prior to declaring the view.

2. Viewing the World Wide Web

The importance of having a Microcosm viewer for html and the solution that involves adapting the source code of NCSA Mosaic as a fully aware viewer are discussed in section 6.2. However, a simple and immediate solution has been achieved by using the Universal Viewer, running on top of the Netscape html viewer. (It was necessary to use Netscape rather than NCSA Mosaic, as Netscape enables the user to make selections within the client area.)

Using this package with no changes makes it possible to author and follow links out of html documents, and to follow links into html documents by launching Netscape with the appropriate URL.

However, a simple addition has been made to the Universal Viewer, which is an "Import Document" option. When UV is running over Netscape, and this option is chosen, it will create a DMS entry for the current URL, even getting the description of the document from the html. This makes it possible to use Microcosm as a "personalised viewport on a global information system", by browsing around the Web finding and keeping references to documents as and when they are discovered. In effect Microcosm is maintaining the hotlist (or bookmarks), and adding value by making it possible to *link out* of documents on remote servers. This subject is discussed in more detail in section 6.2.

3. Working from Outside Microcosm.

One of the problems we have identified with the Microcosm model is that it is always necessary to launch documents from within Microcosm. This is necessary because Microcosm expects to maintain a list of which documents are currently being viewed and in which viewer, so that it is able to send messages to these documents, such as the list of buttons to be displayed. The problem with this approach is that it is not possible for a user to be working in their normal information environment, and then to decide that they want to enable link services,

without first starting Microcosm, importing the document details, then launching the document from within Microcosm.

Simon Knight has produced two extensions to the Universal Viewer. The first is an additional menu on the Windows file manager, which allows the user to select a document and ask for a *Microcosm Launch*. If Microcosm is running, it will import the document details to Microcosm (if they are not already there) and instruct Microcosm to launch the document in it's choice of viewer. If Microcosm is not running, it will launch Microcosm first.

The second extension is a "grabber", which allows the user to view any document in their chosen application, then to point at the document and to ask to view it under Microcosm. If the default viewer for Microcosm is the Universal Viewer, this will immediately be launched over the application window, with any buttons etc. If the default viewer is one of Microcosm's viewers rather than the current application, then a *copy* of the document will be launched in the appropriate Microcosm viewer.

5.7. Summary

In this chapter I have discussed the design issues that must be considered in order to provide hypertext functionality to third party applications, either with a minimum of adaptation using application macro facilities, or with no adaptation.

The key to success in this endeavour is to separate both the link and the anchor information from the node content, so that the application is not required to provide the functionality to handle anchor identifiers within the data content. Instead, all that the application must be able to provide is the selected object (such as the selected text string), the name of the current data file and the action chosen by the user. In modern GUI operating systems, the operating system itself is usually able to provide of all these details except the action chosen by the user, and to this end we have provided a shim program known as the Universal Viewer, which can provide this information. Separating the anchors from the data has side effects with regard to the integrity of links when editing the data. This subject is discussed in detail in chapter 8.

The resulting hypertext system is compared with a fully aware system in table 3, and differs from its predecessors in two significant aspects. Firstly, navigation is primarily node-to-node or, local or generic anchor-to-node. Secondly, link anchors are generally not displayed as buttons or highlighted objects in the way that users have come to expect This issue is discussed in the paper "Ending the Tyranny of the Button" (Hall, 1994), which urges that users should be encouraged to expect to query the system for links in much the same way as they might query an encyclopaedia for information: in any system where links may be generated dynamically every object in the document potentially becomes the source of a link. Having the whole document highlighted would be intrinsically no more useful than having no buttons at all. In Microcosm the interface to hypertext functionality is achieved by taking a selection and identifying if the selection, or any part of the selection, represents a resolvable anchor. This makes it possible for users to ask what links are available within, for example, a whole paragraph of text.

It is clear to the author that in the future much of the functionality that we have described in this paper should be, and perhaps will be, implemented within the operating system. If the operating system allowed users to attach extra menus to any application (in much the same way as Microsoft use Visual Basic for Applications) and would also provide functions to return information about the current selection, such as the name of the selected object or the position of the selection, then link services could provide a higher degree of specificity for link anchors. The Apple Events suite for inter-application communication makes a start at providing these sort of facilities. The editing problem would be simplified if applications, such as the link service, could register in an "interest set" that would be notified when certain actions occurred, such as files moving or objects changing. In this case deamon processes could monitor such changes and immediately attempt to resolve any resulting integrity problems.

Finally, we hope that in the future operating systems will provide much improved facilities for attaching attributes to files, such as meaningful names, keywords and other information, in much the same way as applications such as MS-Word allow these attributes to be stored, and subsequently queried. The new generation of object oriented systems such as CAIRO, OLE 2.0, NextStep, OpenDoc (Apple Computer Inc., 1993) and CORBA (1991) show promise, and future research will be directed to discovering to what extent these systems can provide the facilities we require.

Fully Aware Viewers	Applications running with Universal Viewer
When loaded as result of a link following operation, the viewer loads the data so that the destination anchor is highlighted in some way.	When loaded as result of a link following operation, either the data is shown at the start, or a stored keyboard macro executes to display the destination anchor.
Viewer asks for buttons relevant to this data, which are then highlighted in context.	Universal Viewer asks for buttons relevant to this data, which are then located on the title bar of the application.
Author may pre-set viewer display parameters such as window size and position, fonts, colours etc.	Window size and position may be pre-set but other parameters are determined by operating system and application defaults.
Typical actions on the action menu will be, Follow Link, Compute Link, Show Links, Start Link and End Link. These menus are configured dynamically by querying which filters are currently available.	Actions on the action menu will be the same as the fully aware viewer. However it will not be possible to author *specific* source links, and destination anchors are resolved by content search rather than position.
If the document is edited the viewer will need to be able to communicate with the linkbases to resolve any links that have been moved.	Since there are no specific source or destination anchors, the document may be freely edited. The only problem that might occur is if a feature used, e.g. by a search engine, to locate the end of a link was altered, removed or duplicated.
Viewers have been written to produce links into and out of temporal media (digital sound and video).	No facilities for dealing with temporal data, other than launching the data at the start point, and following links from the whole document.
A *show links* action on the menu will find all links within a given selection and either highlight them within the current viewer or list them in the available links box.	A *show links* action on the menu will find all links within the current selection, and either display them as graphic icons on the title bar, or list them in the available links box.

Table 3: Contrasting the facilities provided by the fully aware viewers with those provided by applications running with the Universal Viewer.

Chapter 6. Extending the Microcosm Model

This chapter describes some of the extensions to the basic Microcosm model that we have made within the research laboratory.

6.1. Distributed Microcosm

As has been described in the previous parts of this chapter, a complete Microcosm environment consists of the following components:

- The documents
- The descriptions of the documents (held by the Document Management System or DMS)
- The Document Management System (the software which processes the descriptions)
- The Document Control System
- Viewers (software which allows users to view documents)
- The Filter Manager
- Link descriptions
- Linkbases and linkers (the software which creates and retrieves link descriptions)
- Other filters (e.g. the history filter and the computed linker)
- Data accessed by other filters (such as stored histories or information retrieval indexes)
- The Registry (which stores and retrieves the registry data)
- Registry data

When considering the distribution of Microcosm, it is necessary to decide which parts of the above are to be distributed, and how.

At one extreme is the *distributed data model,* where only data items will be distributed: documents, their descriptions, link descriptions, other filter data and registry data may be held on any machine: the processes (viewers, DMS, DCS, FM, filters, and the registry) all run locally but gather their data from systems on the network. All that is required to achieve such a system is a network infrastructure which allows one machine to access data from another fileserver machine, for example Windows for Workgroups, LAN Manager or PCNFS all allow one Windows PC to load data from other connected machines.

Where Microcosm is used in teaching laboratories such a set-up is commonly used. The application data is all mounted on a fileserver. The Microcosm software may be mounted on the fileserver or on the client, but in either case it is run on the client.

At the other extreme is the *distributed process model,* where any Microcosm component may be distributed: all process except the kernel processes (DCS and FM) and the viewers may run on any machine. A suitable inter-process communication model, such as Windows Sockets, is used to communicate between the client and the remote processes. Such configurations may conform to a strict *client-server model*, where some machines are set up as Microcosm servers for other client machines to connect to, or they may be *peer-to-peer*, in which case various machines may be running Microcosm locally, while publishing their process resources and data for other clients to share.

Clearly the distributed process model requires that the Microcosm message passing protocol is extended to include information about machine addresses, and the functionality of the filter manager must be enhanced to allow client machines to discover what server machines are available and what processes they offer. Although such functionality is not yet part of the standard Microcosm release, a research implementation is discussed in Hill & Hall (1994) and in more detail in Gary Hill's PhD thesis (1994).

Whether the distributed data or distributed process model is adopted depends greatly upon the purpose for which Microcosm is to be used. The distributed data model lends itself to a world in which Microcosm is used to personalise a view

upon a universe of documents, where users will browse this universe using whatever resource discovery tools are available, possibly third party, and then, having discovered interesting data or trails through the data, will use Microcosm to link to this data or to store the trails. On the other hand, the distributed process model assumes that Microcosm itself provides the mechanism for resource discovery. The universe of data will be known to assorted Microcosm servers, and the client will explore this universe by connecting to these servers in a similar way that World Wide Web servers and WAIS servers are currently used.

The development of a client-server model for "industrial strength Microcosm" is currently an active research interest within the Multimedia group.

6.2. Working with the Web

The World Wide Web (WWW) (Berners-Lee et al, 1992) was originally produced at CERN for the distribution of information about high energy physics, but it soon became apparent that it had a far broader application, and has now become the most commonly used distributed hypermedia system. The principle of the Web is that there are various servers located around the world which may be accessed by appropriate clients (such as Netscape and Mosaic) using a protocol known as Hypertext Transfer Protocol (http). Documents on the Web are held in a format called Hypertext Mark-up Language (html), which is an application of SGML. Links are marked up within html in the form of Universal Resource Locators (URL's) (Uniform Resource Locators, 1994). URL's contain the information about the access method (http, WAIS, ftp) and html allows various other formats such as GIF bitmapped files to be included in the text at run time.

In spite of its manifest popularity, the Web is not the final solution to all problems in the field of hypertext. It has the advantage of providing an excellent interface to file retrieval over a wide area network, it is easy to use, it is conceptually relatively easy to understand and fairly simple to author individual documents. Furthermore, it is possible to pass around URL's by other technologies such as email. However, it introduces a number of problems: the embedded nature of the links means that link editing is virtually impossible from outside the source document, dangling links are rife, and document organisations, once fixed cannot be easily changed. The reliance on html as the format means that all documents

must be published in this format, and it is not possible to provide hypertext functionality to other data formats.

6.2.1. Accessing the Web from Microcosm

In view of the popularity of the Web, the Microcosm team has taken the view that it is important that Microcosm is capable of inter-operating with the Web, which is fortunately made very simple by Microcosm's open architecture. We have experimented with two approaches to this problem. The first involved producing a fully aware Microcosm html viewer and the second involved running Netscape as an Unaware viewer.

An html viewer for Microcosm is in production that adds source code to Mosaic so that it provides a Microcosm action menu, with the normal *start link, end link, follow link*, and *compute link* options. Mosaic will display both html buttons and Microcosm buttons: when an html button is pressed, Mosaic will behave as normal, and replace the current document with the new html document specified by the associated URL. However when a Microcosm button is pressed, or when any selection is made and follow link selected from the action menu, then the usual message is sent to Microcosm which will attempt to resolve the link in the normal manner.

The problem with the above approach is that the base code we are using for the html viewer is somewhat suspect, so for real use of Microcosm we have preferred to use the Universal Viewer to run Netscape as an unaware viewer as shown in figure 20. In this case the action menu looks the same, but Netscape is unable to show Microcosm buttons: instead the user has the choice of clicking WWW buttons, or making a selection and asking Microcosm for links.

Microcosm link destinations (from any viewer) will initially resolve to a unique document identifier which will in turn be resolved by the document management system to either a path and filename, as understood by the host operating system, *or* to a URL. In the case where the DCS is asked to dispatch a URL it invokes Mosaic or Netscape as the viewer with the given URL as the data file.

One final problem exists. When users are browsing, using html buttons within Mosaic or Netscape, they might discover documents that they wish to make Microcosm aware of. Also, we might like the Microcosm history to be aware of

documents that were viewed, even if they were reached via html buttons rather than through the usual Microcosm dispatch process. For this reason we have added an "import document" option to the menu which allows the user to request that the URL for the current document is imported into the DMS, using the html title as the document description.

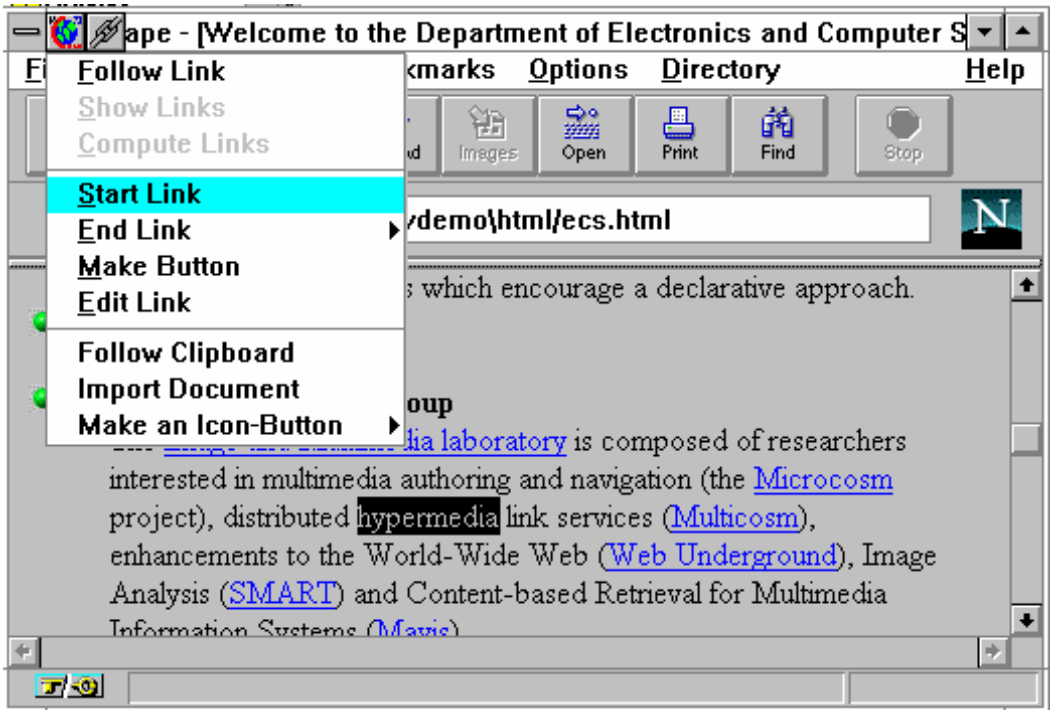

Figure 20: Creating a Microcosm Link in html using Netscape with the Universal Viewer

The advantage of using this approach to browsing the Web is that Microcosm acts as a kind of "super hotlist", allowing users to maintain a local viewport on the global universe of documents. Microcosm allows the user to keep all the document descriptions in the hierarchical DMS, while at the same time allowing the user to author and follow private, locally maintained links.

6.2.2. Converting Microcosm for the Web

Many authors find it easier and faster to author in Microcosm than in the Web. Also, if an application is built in Microcosm it is far simpler to maintain the structure and the links as the contents change and evolve. However, the popularity and accessibility of the Web often make it the preferred delivery platform. For this reason some of our users have adopted the strategy of using Microcosm to produce and maintain applications, then to convert the information for delivery on the Web.

The mapping from Microcosm to the Web is not complete: the Web cannot provide all the functionality that Microcosm provides, so it is important that users are aware of what will convert as they build their applications. RTF may be converted automatically to html and Windows bitmaps may be converted to GIF. Microcosm specific links and buttons may be converted into html buttons. Local links and generic links may also be converted to html buttons, by searching the text for every occurrence of the text string at the source of the link. The hierarchy of the DMS may be converted to a "front page" html document containing the top level logical descriptions, with links to separate pages for each branch of the hierarchy, containing further logical types or document descriptions themselves.

An interesting complication arises, when converting Microcosm applications for the Web, in that it is common to have more than one link from each source anchor (1 to many links), whereas the Web currently only supports 1 to 1 links. We have experimented with two methods of dealing with this problem. The first involves putting extra markers in line beside the button, thus:

" The main cause of hyperventilation (also)(also) is ....."

In the above example there are three links from the word "hyperventilation". The problem with this approach is that in a richly linked hypertext, the text tends to become saturated with extra link markers, rendering it unreadable.

A second solution is to generate a number of secondary documents, containing the descriptions of the links to be followed. Thus there would be just one link from "hyperventilation" to a secondary document containing the three link descriptions. Although this approach is more complicated it leaves the text more readable, and the interface looks similar to the available links box in Microcosm.

6.2.3. Putting Microcosm into the Web

The benefits of keeping links separate from the content are so substantial that we have started a project which aims to put Microcosm technology into the Web. This project, known as the Distributed Link Service or DLS, assumes that users will be working with a standard Web client such as Mosaic or Netscape and puts the Microcosm technology at the server end. A similar approach to separating structure from content is taken by the Hyper-G system (Flohr, 1995)(Andrews et al, 1995). However, this system prefers users to connect to the server using a special Hyper-G

client such as Harmony. Users connecting with standard Web clients are treated as a special case, and the service is inferior.

In the DLS, Web servers have access not only to documents, but also to linkbases, and are extended by CGI scripts to allow access to these linkbases. Documents are kept in html, but only those links which are always required to be present are maintained within the html as buttons. All other links are kept in the linkbases in standard Microcosm form. Now the client may identify which linkbases to connect to, so that whenever a document is dispatched by the server, all the appropriate links from the connected linkbases may be "compiled" into the html as URL's. This separation of links at the server has the advantages that:

- we may maintain separate views of the information for separate users and subject areas;
- tools may be produced to manage and maintain the links and the associated network;
- link authors may specify local and generic links, thus saving much authoring effort;
- programs may be written to generate links automatically, for example by making generic links to all the entries in a dictionary.

We have also produced an additional client end program called "Gumshoe", which uses the same technology as our Universal Viewer. This program "sticks" to the Netscape window and provides a drop down menu with Microcosm style options *Follow Link*, *Show Link*, *Start Link and End Link*, and any further options we wish to provide. The user may now make selections and choose actions within Netscape. These actions are then packaged as a query at the client end which is sent to the server which refers to the linkbases to handle the query. This approach has the further advantages that:

- users may now create and maintain their own links over the documents on the server
- there is no need to compile all the generic links into the html as buttons, since the user may query the text in the normal Microcosm mode

- third party applications may be used as viewers, and send queries to the server in the same way as Netscape does

However, this facility can only be provided if the user takes the trouble to download and install Gumshoe, or one of the alternatives we have produced for X-Windows and the Macintosh.

At the time of writing the DLS is in its early stages of development. However we can already demonstrate the extra power and flexibility this approach brings to the Web.

6.3. Working with Multimedia

The question of how to manage databases of multimedia data items is still an active research topic. There are those who argue that traditional databases should be extended to deal with multimedia data types (Grosky & Mehrota, 1989)(Rhiner & Stucki, 1991). The problem with this approach is how to manage queries on multimedia data fields, and most systems have dealt with this by associating a text description with such fields, and using the text for query resolution. We argue (Hall & Davis, 1994)(Davis et al, 1994a) that hypermedia link services working over the entire range of multimedia viewer applications can provide an acceptable alternative approach to locating multimedia data and integrating multimedia applications.

Some hypermedia link services take a "chunky" approach, allowing only node-to-node links. This approach has limitations, in that it does not allow the user to create or follow relationships between individual objects within the nodes. Most hypertext systems accept that any text span within a text document could be a potential anchor for a link source or destination, allowing for finer grained relationships than those only between nodes. The problem to be faced when extending this metaphor to dealing with non-textual data is how to define what constitutes an anchorable object, but before proceeding let us review the facilities that are available by default for *any* data type running in *any* (possibly unaware) viewer within Microcosm.

1. *The Microcosm file manager.*

   It is possible to locate any document by description using the Microcosm file manager. A file may appear in more than one place within the hierarchy, and will display its physical data type. This adds only a little more value than a standard file browser, but critics should not under-estimate this method of resource discovery.

2. *Document attributes.*

   Since all documents known to Microcosm have associated attributes held within the DMS (see section 3.6) and the DMS provides Boolean query facilities for such attributes, it is possible to argue that Microcosm anyway provides all the functionality of a traditional database management system extended to deal with external multimedia fields.

3. *Links to the documents node.*

   A link may be made from any source anchor to the *whole* of the node of *any* document type. Such links may be generic, local or specific.

4. *Iconic buttons from documents.*

   Even if Microcosm knows nothing about the content of a node, it is still possible to make iconic buttons (see section 5.5) from that node to any destination anchor.

The above points emphasise the fact that even with multimedia data types of which Microcosm knows nothing, Microcosm can provide all the functionality of a sophisticated browser and a multimedia database; it can add the value of node-to-node linking, or where one end of the link is a text node or Microcosm aware viewer, it can do object-to-node or node-to-object linking. This is a satisfactory position to start from, even *before* considering how to identify and use anchors on objects *within* multimedia data.

The remainder of this section discusses the progress we have made with various non-textual data types. Further research is described in section 6.4. Before progressing it is worth reminding ourselves that Microcosm makes no assumption about how an anchor will be expressed. The viewer determines the semantics of the

anchor description: the linkbase will store whatever it is told to store (so long as it is represented as ASCII text), and when requested to resolve a link from an anchor will compare whatever data is sent by the viewer with whatever is stored, so it is up to the programmer of the viewer to determine how to represent an anchor.

6.3.1. Bitmapped pictures

The easiest way to represent an anchor within a bitmapped picture is to use the co-ordinates of the rectangle or polygon which surrounds the object of interest. This allows the user to create specific link anchors and buttons. The Microcosm bitmap viewer allows the user to create such selections at either end of a link, and provides a number of ways of highlighting such anchors as buttons.

The more difficult (and interesting) problem is how to map the concept of generic links onto pictures. What one would like to be able to do is to "poke" at any object in a picture, and for the system to identify what the object represents, and to then follow any generic links on this type of object; for example if the object that was clicked upon was a horse, then we would like to follow generic links on the word (or concept of) "horse". Of course the horse in the picture might be piebald, and so we would also expect to follow links on "piebald", and it might be jumping in a show so we would want to follow links on "show jumping" and so on. The problems involved in understanding the semantics of a picture are far from solved, and although we are carrying out research in this area (see section 6.4) we needed to come up with a more immediate solution.

Our solution to this problem is to allow users to create selections tagged with keywords or phrases. These anchors are treated just like buttons and stored in the linkbase like any other button link. When the viewer starts running it will query the linkbase for any buttons belonging to this data, and these tagged anchors will be returned and highlighted in the same way as any "normal" buttons. The difference is that whenever a user moves the cursor over such a button it will pop-up a message displaying the text of the tag, and whenever this button is pressed, instead of attempting to follow a specific link based on the position of the selection, the viewer will send a message requesting to follow links on the *text* of the tag.

We can thus create a generic link on the word "horse", which takes us to any suitable resources we have on horses, and then wherever a horse appears in a

picture we can put a button over it, tagged with the text "horse". Clicking on any of these buttons will then take us to the same set of resources.

This solution is less general than the equivalent generic links in text, as there is a degree of effort identifying objects in pictures and manually tagging them with their associated concepts. However it is a more general solution than limiting oneself to specific links.

We found this approach particularly useful when creating a large number of bitmaps showing different views of the same block of a city. The bitmaps were created by rendering a 3-D model held in AutoCAD, which already had names associated with each object in the model, and generic links associated with each of these names. By knowing the viewpoint from which each bitmap had been produced it was possible automatically to create enclosing polygons for each object in each bitmap and to store them with their associated names. The result was that whichever bitmapped picture the user was looking at, they could click on an object and find out what information was known about it.

6.3.2. Object oriented drawings

Object oriented drawing packages are one of a small class of applications that can actually identify to themselves the individual objects within their control at any more appropriate resolution than the individual character or pixel. We have worked with AutoCAD which not only allows the user to create aggregate objects, but also allows the user to *name* them. Using AutoLisp it is possible to add an action menu to the application, and to arrange to send *follow link* (and other) messages to Microcosm with the *name* of the selected object as the source anchor. It is also possible automatically to extract all the names of the objects in a file and create local links based on these names.

Local links are appropriate in this context, since it is possible to move the viewpoint but, as long as you are still viewing the same file, AutoCAD will return the name of the object from whatever angle it is viewed.

Following links into a specific place in an AutoCAD model is also possible, since AutoCAD may be persuaded to highlight any named object.

6.3.3. Temporal links: sound and video.

The Windows Media Player acts as the application which can play digitised sound files (WAV), MIDI files, CD sound, and digitised video (AVI). Similar software exists for playing video disks. Using the techniques of linking to the entire file, and putting iconic buttons on the application title bar, as described at the beginning of this section, allows primitive links into and out of such media. However, we discovered that there was a need for finer grained linking into and out of such files, and for this reason decided to produce a Microcosm aware "sound viewer", described in detail in Goose & Hall, (1993) and a Microcosm aware video viewer.

Producing such viewers presents six issues:

1. *How will a source anchor be represented in such media?*

   Given that the link service takes no view on the form of an anchor, it is up to the viewer to decide how to express anchors. In the case of all these media the common unit of currency was the millisecond. Source anchors were expressed as start and end points in terms of milliseconds through the document. The viewer is responsible for understanding and interpreting offsets given in this format .

2. *How will a destination anchor be represented in such media?*

   Destination anchors were also expressed as single point offsets in milliseconds.

3. *How will the system present the presence of an anchor to the user?*

   We have explored two approaches. The first, which works in both sound and video, is to present a time line at the bottom of the viewer. A source anchor is presented as a bar which moves as time progresses, as shown in figure 21.

   When the bar crosses the "now point" it becomes active. If it is an automatic link, as soon as the start of the anchor reaches the "now point", the link will be dispatched. If it is an ordinary button, the user will have to double click on the bar to dispatch the link. The bars contain the text describing the link, and the viewers have normal CD player style controls so that the user may interact with the media, rather than only receiving a passive demonstration.

Automatic buttons were a new link type, introduced specifically to deal with this problem, but may be used with any other media type, so that whenever the button should be displayed it will automatically fire its associated link.

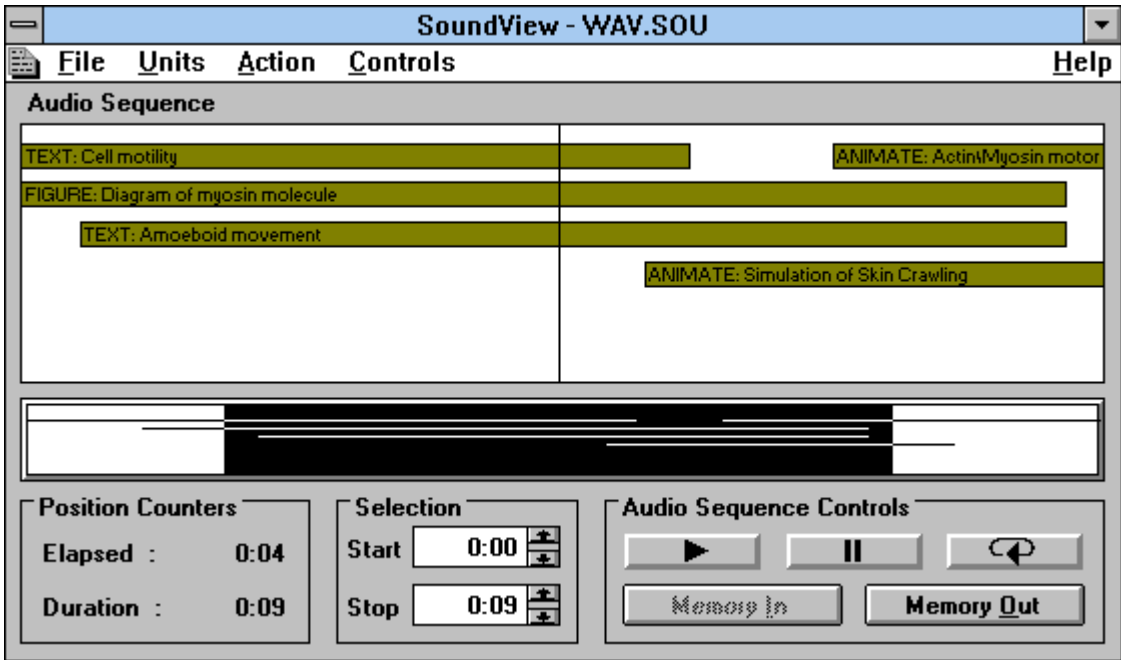

Figure 21: The Microcosm "sound viewer", showing button links on the time axis.

4. *Given a large resource (such as a video disk or a CD) how will the system know what part of the whole should be represented to the user?*

Some authors have taken the view that the part that should be played is the destination anchor. We have not done this, but rather have chosen to describe any large resource in terms of a number of smaller resources expressed in terms of the frames through the whole. Thus a CD consisting of say 12 recorded tracks might be divided into 12 "files" that will be individually registered with the DMS, and anchors within this file will be expressed in milliseconds through the individual file. Once such a file has started playing it will play to the end, unless specifically stopped by the user.

For video we have explored a second approach to displaying source anchors. Authors make source anchors by creating a polygonal selection over the object intended as the source anchor while the video is halted. They then play the video, dragging the selection so that it remains over the object as it moves.

The information containing the co-ordinates of the rectangle with time are stored in the linkbase in the same way that buttons are stored. When the video is played back the rectangle will re-appear, and clicking within the rectangle will dispatch the link with which it is associated.

The approaches described above are not mutually exclusive, although we have not, as yet, combined the two.

5. *How may discrete multimedia be synchronised?*

In the crudest sense, dispatching links at the moment that a button in one media crosses the "now point" is a form of synchronisation of events, and would allow an author to play, for example, a piece of video and then at a given point in the video to automatically play a sound track. However, the overhead of dispatching a link is an unknown and variable quantity, dependant on system configuration and load, so the synchronisation can only be approximate.

To effect accurate synchronisation of the types discussed would require that a single application controlled the replay of all the different temporal media, rather than the independent communicating process mode that Microcosm uses. Work on synchronisation of hypermedia components is being conducted at CWI (Hardman et al, 1994) and MIT (Buchanan & Zellweger, 1992).

The HyTime standard for multimedia (HyTime, 1992) (Carr et al, 1994a)(DeRose & Durand, 1994) was particularly designed with the intention of making it possible to specify the synchronisation of multimedia events, but at the time of writing very few systems conform with this standard: Microcosm will only partially map on to HyTime (Carr et al, 1993).

6. *What is the meaning of generic links in such media?*

The implementation described above only covers specific links implemented as buttons. The same problem applies to extending the metaphor to generic links as was described for bitmaps in subsection 6.3.1 above. It is possible to associate tags with the buttons, so that generic links may be followed on the subject on the text of the tag.

6.3.4. Spreadsheets

In the simplest case, the text of any cell of a spreadsheet may be used as the source of a link. These links may be generic, but since spreadsheets divide their workspace into cells, it is possible to use the cell address as an offset, so implementation of specific link sources, and of link destination anchors is possible.

However, spreadsheets are another example, like object oriented drawing packages, of applications that allow the user to associate a *name* with an object (a range of cells) in the package. An alternative approach to using the *text* of the cell as the link anchor is to use the *name* itself (i.e. a range of cells) as the anchor. This might be more meaningful to the user, as a name may be associated with the area of a spreadsheet that performs a particular calculation, for example the cost of building some machine from its components, and the user may wish to make links to and from this calculation, as a whole, to related drawings, manufacturers price lists etc.

6.4. Current Research Areas

There are still many research and implementation issues to be addressed in the area of link services and open hypermedia, and many new application areas to be explored. This chapter examines some of our current research issues in addition to those that have been discussed in chapter 3 and elsewhere.

Two important facilities that we are being urged to implement by our users are *multiple installable interfaces* and a *composite viewer.* Many users find the raw operating system interface difficult or intimidating to handle, and resource providers are keen to supply interfaces that are as simple to use as possible, Furthermore in the case of publishers, they are keen to provide an interface that is in keeping with their corporate image.

Such interfaces take over the screen, providing a single window within which the current document of interest is displayed. Access to all the filter configuration menus, the registry and the DMS is via a control panel. These interfaces are in principle quite simple to create, using, for example, programming systems like Visual Basic, and rely upon the idea that there is some API through which one can

access and control every aspect of Microcosm functionality. In practice this is not currently the case, so, for example, many filter settings may only be controlled by user interaction with the filter's interface. In order to proceed with the production of installable user interfaces we are now returning to the code to ensure that there is a function within the API to cover every action. An interesting decision to be made is whether all these functions should only be made available through a standard programmer's API, in which case the only way they may be accessed is by writing a program, or whether there is a class of functions that should be accessible by the Microcosm message protocol, so that the system could send messages to a particular component to change its settings in some way.

Composite viewers are viewers that can display sub-windows within the main window, possibly containing multiple data types. The author can control the size and position of the sub-windows and create links to the composite object. The advantages of such a system are that in any publishing application the publishers may wish to improve on the presentational features of raw Windows, and also it might be possible to create certain effects that are not easy to achieve when applications are all separate processes, such as parallel scrolling of text windows and synchronisation of multimedia events.

Another area of interest on the subject of interfaces is the use of virtual reality interfaces for information management. Virtual reality in this context does not involve helmets, gloves and suits, but refers to *desktop VR*, where the user is able to navigate about a 3D model presented on the normal computer screen, using standard computer input devices. We have used such systems as an interface to an urban model and as an interface to a machine's maintenance manual, but there are many applications to which such an interface might be applied including such ideas as extending the basic windows metaphor to a three dimensional "rooms" metaphor.

At present the majority of the hypermedia functionality of Microcosm is provided by the linkbases and associated filters, and the computed linker and its indexes. There is a much scope for the production of filters that apply intelligence to both the production of new links, using information retrieval techniques, expert systems and semantic techniques, and to the filtering of links according to context, using such techniques as relevance feedback (Durham, 1989).

Section 6.3 discussed the problems in extending the concept of link anchors to encompass all data formats. There are two strands to this research. One strand involves the idea of forming a concept database, which is effectively a hierarchical classification of all the topics within the subject domain. When a new resource is introduced to the system it is described in terms of subjects within the concept database, and these descriptions form links from that concept to that resource. Now if it were possible to form generic links from any object in a node to the associated concepts in the database, the link could indirect to the destinations linked to that concept as shown in figure 22.

The challenge then is to produce tools so that it is possible to point from any object in any media to the concept in the database. If this were possible then the action of importing a resource into the system would be all that would be necessary to have it fully integrated into the hypermedia system. At present it is still necessary to mark-up most media and tag the mark up with the required concept(s). This work is described in Beitner & Hall (1995).

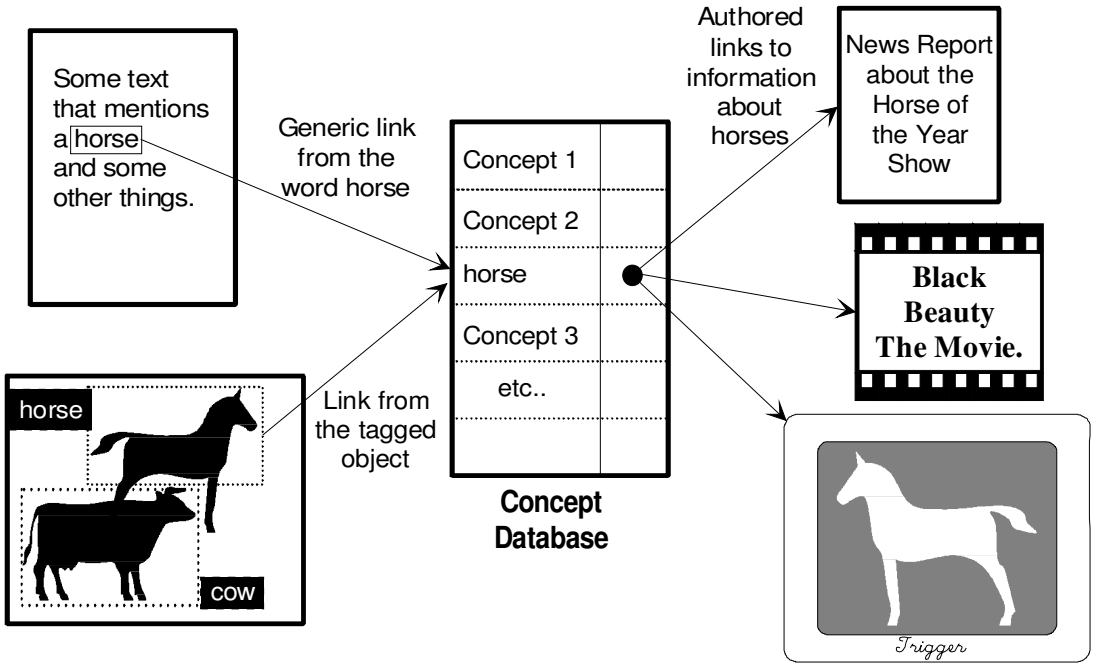

Figure 22: Indirect linking via the concept database.

The SERC funded MAVIS project (Hall et al, 1993b) within the Image and Multimedia laboratory at Southampton is investigating techniques for extending the concepts of generic links and computed links to work with images and video by using image analysis. The aim is to enable the user to make an arbitrary selection within an image and then to ask to *follow links* or *compute links* in the same way as is possible from text based documents. Object recognition is still a technology in a primitive stage, so the kind of techniques being used involve the production of signatures for features such as shape, colour and texture, then searching for similar signatures stored in the linkbases or indexes. This project has involved the production of a new linkbase architecture, since in this case the linkbase will rarely, if ever, find an exact match with the selection. Instead it must find "best matches", which may be formed by weighting the results of modules working on different features of the image. The work is described in Lewis et al, (1995), and Wilkins et al, (1996).

Finally, a fundamental change has been suggested to the basic link service. Originally the filter chain was designed so that every message passed through every filter. In order to optimise the system the model was changed so that messages were only directed to those filters which registered the ability to handle messages of any particular action (Hill et al, 1993). The revised model (Wilkins, 1994) suggests that filters and Microcosm aware viewers should be enabled to communicate directly with each other, either by broadcasting to all or by posting directly to another component that is known to be currently available.

This model has the advantages that the messaging is considerably faster, that it is possible to route messages around the filters in any order desired, that the possibilities for parallel processing are increased and that it is possible to notify components of the system of changes that affect them, even if they are not currently running. It has the disadvantages that if a message is simultaneously broadcast to a number of filters one loses the sequential aspect of the message passing, so it is no longer possible to insert filters that block messages. Furthermore the coding overhead to make viewers Microcosm aware is increased. Our understanding of the long term effects of this model is not complete, particularly in terms of backward compatibility with the current system. Further experimentation will be needed before we adopt this model but early prototypes are encouraging.

Chapter 7. Data Integrity Issues

The previous chapters introduced the concept of open hypermedia link services, and described the Microcosm system, which at the time of writing is one of the few systems that meets a large number of the criteria for openness. The problem with attempting to produce a system that is so open as Microcosm is that many opportunities for inconsistency of data arise.

My original prototype for Microcosm, which was known as LOOM, had a number of features that were not carried forward into the full implementations of Microcosm, not because we failed to specify these features, but simply because they were not seen as sufficiently important to ever reach the top of the to-do list. Most importantly, the LOOM text viewer had an edit mode which would allow link aware editing of a text file. This was important to me, as I envisaged using the text viewer for developing program code, and it was vital that the text could be edited without breaking all the links. However, the applications to which the early versions of Microcosm were put did not have such problems, being largely static published material such as historical archives. When such static material is being used, it is generally not necessary to edit files, so the opportunities for breaking links are many fewer.

In any case, providing a link editor does not preclude the possibility that files might be edited by some other programmer's editor, and we have subsequently discovered that maintaining consistency in a link service style hypertext is a complex problem. This chapter introduces these problems and the remaining chapters examine the problems in detail and explore the possible solutions.

7.1. Axiomatic Design Constraints for Microcosm.

Before investigating the limitations and problems with the current Microcosm model described in chapter 3 it is useful to remind ourselves of those policy priorities, along with the aspects of openness from subsection 2.4.1, which led to the design, and which result in a system with different features to many of the other current generation of systems.

Important axiomatic design constraints were that:

1. *The hypermedia layer would not prescribe the content of the storage layer;*

   -so that any application running on the host operating system could be hypertext enabled and new data formats could be seamlessly integrated.

   - open with respect to data format

2. *That applications could be enabled as viewers with little or no effort;*

   - open with respect to applications

3. *That users would be able to have their own view and extensions over a published set of resources with no effect upon those resources;*

   - open with respect to users

4. *That the functionality of the hypermedia layer could be extended by adding new processes dynamically;*

   - so that there would be no need to access the source code of any other components or to make any changes to the link service layer.

   - open with respect to data models

5. *The native file system would be used as the resource storage layer;*

   - So that the hypertext layer would be seen as a lightweight extension of the user's normal working environment

6. *The hypermedia layer of the model would contain no constraints to portability to other popular operating systems;*

   - open with respect to platforms

7. *That the model would allow the data resources and the processes to be distributed over a network;*

   - open with respect to platforms

8. That the model would scale to work in a large multi-user environment;

- open with respect to platforms and with respect to size.

It would be fair to say that the first four points have been fully and successfully addressed by the current version of Microcosm. The solutions to axioms 1 and 2 (any data type may be stored and any application hypertext enabled) has been more radical than any other system addressing the issue of application integration, but has knock on effects upon the consistency of the hypermedia model that are addressed later in chapter 11.

Adherence to the principle in axiom 5 (use of the file system as the storage layer) has not yet been as successful as we would have wished. We had hoped that users would be able to move seamlessly between using the Microcosm file manager/browser and that provided by the operating system. However we have found that most application builders name and model their data quite differently when they intend to use it under Microcosm than they had previously arranged it on the file store. The probable reason for this is that most of our users are attempting to present data to naïve users within an educational field, who will anyway be unlikely to attempt to browse the resources in ways other than those provided by Microcosm. In practice we have found that users will either work entirely within the Microcosm file manager, in which case they will ignore the Windows file names completely, or else they will work entirely from the Windows file manager, ignoring the Microcosm file manager and allowing Microcosm to assign default names to the files. Movement between the Microcosm hierarchy and the file system hierarchy does not seem to be practical.

One possible solution to this problem would be to abandon this principle and introduce a purpose built hyperbase storage layer which would replace the file system. This is the approach taken by many system implementors e.g. ABC (Smith & Smith, 1991) and Hyperform (Wiil & Leggett, 1992), and has the advantage that given such a layer it is possible to enforce hypermedia data model integrity constraints, but we are loathe to take this route as it would involve abandoning the "lightweight" approach and commit users to entrusting their entire working environment to Microcosm. The problem is caused by the fact that the current generation of operating systems do not provide suitable features for aliasing files, keeping symbolic links to files, do not provide long (English like) filenames and do not allow users to attach user defined attributes to files. Some operating systems

provide some of these features, but to use these features only where they are available would contravene axiom 6 (portability to other operating systems). One possibility that is currently under investigation is to use a fully functional third party document management system, such as Documentum which would entirely replace the file system. This idea is discussed in greater detail in chapter 9.

Axiom 6 (portability across operating systems) has not yet presented any problems: prototype implementations are now available for the Apple Macintosh and for X-Motif. Where Microcosm applications have not used data formats that are specific to a particular platform it is possible to move Microcosm applications from one platform to another, and to link to or from data on other platforms where the underlying network makes this possible: for example a PC running PCNFS may mount a Unix filestore and access linkbases and text files normally used by the Unix / X-Motif version.

Axioms 7 (distribution) and 8 (multiple users) have been fulfilled in the current Microcosm versions, but the rather naïve implementation has possibly exposed a number of problems which are discussed in chapter 11.

7.2. Link Integrity Problems

Integrity problems in hypermedia systems occur when an implicit constraint of the data model is violated. The most important implicit constraint in the standard hypermedia model is that a link should join two (or more) anchors. This constraint is violated if any of the anchors, or the documents containing those anchors, are missing. It might also be a violation to have unlinked anchors. The Dexter model (Halasz & Schwartz, 1990) ensures that integrity in these respects is maintained by

- *requiring that the within-component layer manages its own anchors.* When a document is edited the anchors will be moved to reflect the edits.

- *forbidding dangling links.* When a document containing an anchor is moved or deleted the corresponding link must be altered or deleted. This requires that the storage layer communicates to the hyperbase layer any structural changes.

Implementing these requirements generally depends upon embedding some kind of link anchor within the node content, and upon maintaining all documents

within a storage layer that is accessible only through the hypermedia system so that documents may not be moved or deleted without informing the hypermedia system, so that those links affected may also be deleted or moved. This solution is inappropriate for Microcosm: we do not wish to embed mark-up within documents and we do not wish to prescribe the storage layer: other solutions must be identified.

7.2.1. The Editing Problem

The editing problem occurs when the contents of a node are changed so that anchors (or, more correctly, persistent selections) which are held externally in linkbases no longer address the correct object. Such edits may occur during an active Microcosm session, or entirely outside the system if, for example, a user takes a standard text editor to some text document. In any system which uses embedded mark-up within the document to mark the position of anchors, then this will rarely be a problem since, in general, as the document is edited the mark-up will move around with the object which it is marking. However placing hypertext system specific mark-up within documents contradicts our axiomatic design constraints.

A number of other systems have also moved away from using internal mark-up and have used external references to point to anchor positions within documents. For example, HB3 (Leggett & Schnase, 1994) uses, and Intermedia (Haan et al, 1992) could optionally use, persistent selections which are maintained by the application, generally as a list of offsets into the file under consideration. Where persistent selections are *external* references to points or objects within documents, then there will be problems when the file is edited as it necessary to simultaneously update the external references into the document.

The editing problem is probably the major limitation of the Microcosm model when it comes to scaling Microcosm to deal with globally distributed documents.

7.2.2. The Dangling Link Problem

The dangling link problem occurs when a link anchor has been placed in some document, and that document has then been moved or deleted without informing the hyperbase system. This problem is not unique to Microcosm, and may occur in

any hypermedia system where the storage layer may be accessed form outside the hypermedia system. Many hypermedia systems use the filestore as the storage layer, and map nodes onto files. In these cases it is always possible that a user with appropriate access rights might rearrange the filestore hierarchy or delete files.

The World Wide Web (Berners-Lee et al, 1992) suffers particularly from this problem as the entire link is embedded (as a URL) in the source document. The destination of the link will often be a document on a remote server. If the system administrators of the remote server then re-organise their file structure it is possible that the link will no longer point to a valid document. It would be impossible for the remote site to inform all documents pointing to it of the changes, since there are no backward pointers: it would be necessary to interrogate every html document on every machine in the world to find out whether they pointed to this server.

The situation is much better in Microcosm. In Microcosm all documents are known by unique identifiers which are resolved to file names and paths by the Document Management System (DMS) only at the time that the file must actually be located. So long as the DMS is updated to point to the new file position, then all links will remain valid. This problem only occurs when users move or delete files without informing the DMS. The default behaviour on attempting to follow a link to a non-existent file is simply to report the problem to the user and to ask the user to attempt to point the system at the file so that the DMS can update itself.

7.2.3. Microcosm Link Structure

Before exploring the possible solutions to the editing problem, it is worth recapping the linking mechanism within Microcosm.

The following text represents a typical Microcosm link in formatted form.

```
\Description Basic Ciliary Structure and Function
\SourceSelection ciliary movement
\RealSourceSelection Ciliary movement
\SourceOffset 436
\SourceDocType TEXT
\DestSelection
\DestOffset 0
\DestDoctype TEXT
\ButtonAction FOLLOW.LINK
\SourceFile 600.04.02.92.16.39.18
\DestFile 600.04.02.92.17.12.53
```

This link is seen to be from a specific text string ("Ciliary movement") at a specific offset point (436 characters through the file) in a specific source file (known to the DMS as 600.04.02.92.16.39.18) of type TEXT, to the top (offset 0) of another TEXT file. This link is to be displayed by the TEXT viewer as a button, and when the button is clicked the action will be to FOLLOW.LINK. i.e. a typical link in Microcosm link is expressed in offsets through a file.

If a link is created as a *generic* link, then the tag *\LinkGeneric* will appear, indicating that the link may be followed from any point at which the *SourceSelection* appears.

The way that selections and offsets are expressed remains the responsibility of the viewer for a particular data type. For example, a simple viewer for bitmapped pictures expresses buttons as by giving the co-ordinates of a polygon within the picture as the source selection. This is stored within the linkbase, e.g. *\SourceSelection 174,63,320,63,320,75,174,75* and the linkbase will happily resolve this when the button is selected at another time.

7.2.4. Identifying the Problem

In the link examples above, if the data is changed it is possible that source and destination anchor positions will no longer point to the correct object. Typically, such changes occur when the owner of a document has edited the content of the document and:

1. All the owner's links become misplaced.

2. Links that other users have made into this document become misplaced.

Some small extensions to Microcosm enable us always to always know when an inconsistency *might* have occurred. At present the system linker does not place dates inside the links so I have produced a new filter called "Dater". This filter sits between the linker filter and the first linkbase, and intercepts CREATE.LINK messages and puts two new tags inside each link:

\SourceDocDate — This tag holds a representation of the operating system's date and time stamp for the file in which the source anchor exists, at the time that the link was made.

\DestDocDate This tag holds a representation of the operating system's date and time stamp for the file in which the destination anchor exists, at the time that the link was made.

It is now possible to apply the algorithm shown in figure 23 in order to ensure that there are no possible link inconsistencies, at any time that a document is accessed by Microcosm by whatever method (link following, from the DMS, from a History list etc.). This algorithm is applied by a filter called "Checker", which sits at the end of the filter chain, intercepting messages to dispatch documents. The algorithm is applied entirely within the link service layer, and does not depend in any way upon the format of the data in the file or the viewer being used. It is an entirely general algorithm for identifying the case when there *might be* some data inconsistency problem caused by file editing or filestore reorganisation.

The algorithm relies upon the principle that whenever specific links are made into or out of any file, then the filestore date of that file is recorded in the link. If, when the file is subsequently accessed, the filestore date of the file differs from the appropriate date recorded in the link, then it is evident that the file must have been changed since the link was created. This of course does not mean that the links are wrong, but points to the possibility that they might be wrong, and that it is therefore worth warning the user, and/or attempting to check and fix any links that are shown to be wrong.

Once a link, or set of links, has been shown to be suspect there is only one way to find out whether it is correct or not, and that is to check the context at the appropriate end of the anchor to see if the data that was supposed to be there is actually there. For example, if a link was made from the text span "*Ciliary Movement*" at some specified offset in a text file, and upon investigating the file we actually find the text "*ry Movement is ca*" at the given offset, then we *know* that the link is wrong, and in this case we might reasonably guess that the error has been caused by the net deletion of 5 characters from some point earlier in the text.

The task of identifying whether a link is *actually* correct is more complex than identifying *possible* inconsistencies. When offsets are held as simple byte offsets through a file, as they are in ASCII files, then this check may be made by the DCS within the link service layer. However, where offsets are held in some format that is specific to the particular application, then we will require that the application

itself attempts to match the links against the offsets. This topic is described in section 8.5.

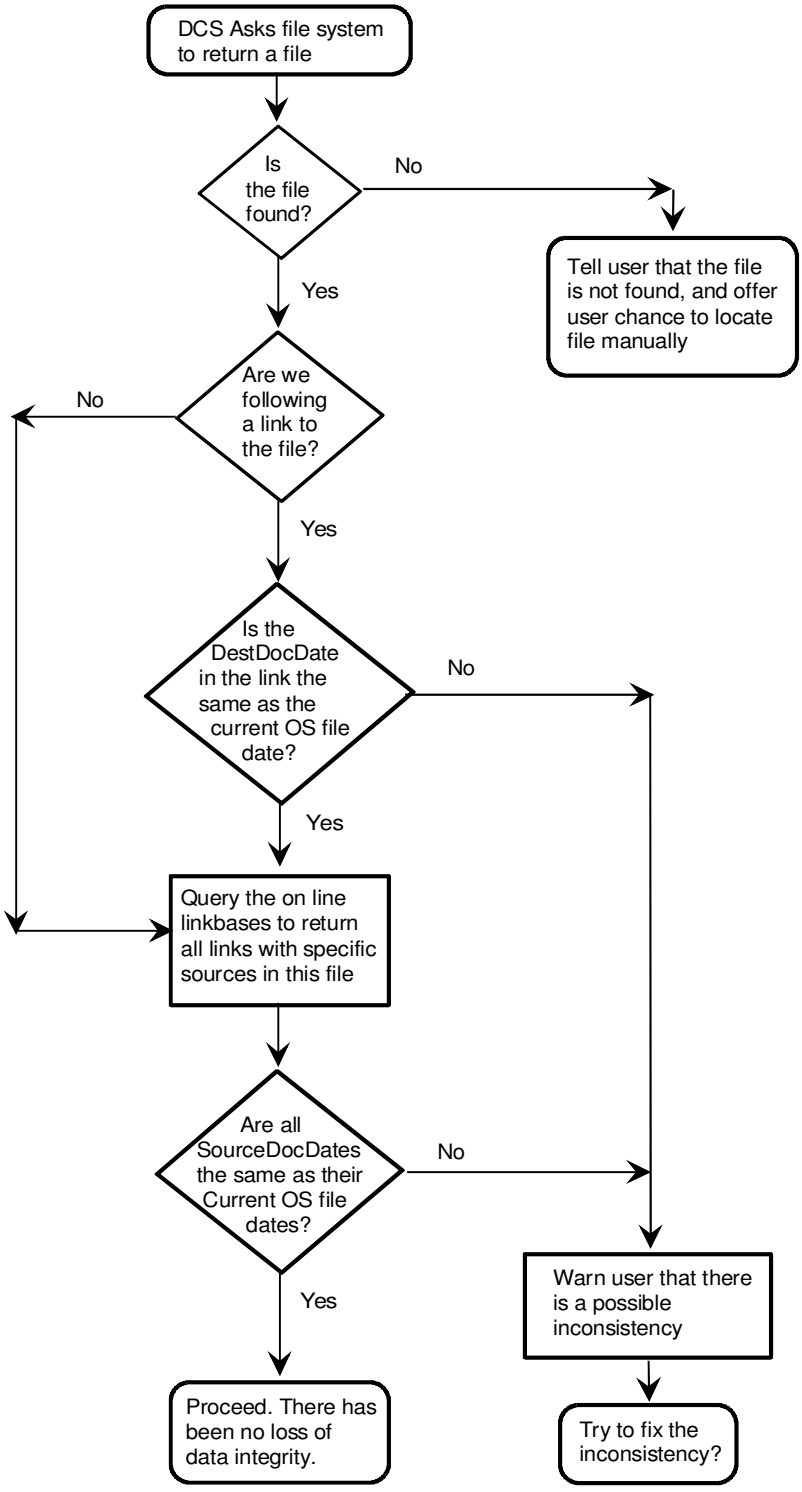

Figure 23: The algorithm to identify possible integrity problems.

The above approach guarantees to identify possible anchor/document inconsistencies *at the time* that the document containing the anchor is accessed. Another useful tool that I have built is an off-line link integrity checker. This will iterate over all the links in a user supplied linkbase checking that the source and destination files still exist and that the anchors within these files are date stamped with the same date stamp as the current version of the file. The result is a list of documents that are suspect with regard to the linkbase concerned, which then enables the user to check the document and fix any suspect links, using appropriate techniques from chapter 8. Steve Rake's LinkEdit, described more fully in section 8.2 also has a tool for identifying link anchors in missing files, but is unable to check whether existing files have been edited due to the lack of dates in the current default Microcosm link.

7.2.5. The Extent of the Problem

Before investigating solutions to the editing problem, it is useful to investigate the extent of the problem.

Broadly speaking, data formats may be divided into 4 categories:

- Text based information - such as ASCII, RTF, SGML, Word Processed Documents;
- Bit stream information - such as bitmapped pictures, digital sound and video;
- Object based information - such as AutoCAD DFX files and Windows Meta Files.
- Other third party proprietary formats.

Table 4 presents some results gained by examining a total of twelve Microcosm linkbases from demonstration applications used at Southampton, containing a total of 8042 links. I have divided the linkbases into three categories. The tutorial style linkbases contain those links that provide the usual hypermedia structure for navigating information resources. The technical documentation category contains many links between object oriented drawings and bitmaps. The glossary style linkbases contain exclusively generic links into text files.

	Tutorials	Technical Documentation	Glossary
Links with source anchor in non-text file	3%	57%	0%
Destination anchor is whole document	96%	39%	0%
Generic + Local Links	48%	57% (including generic picture links)	100%

Table 4: Analysis of Microcosm linkbases

Investigation of table 4 shows that the majority of links have their source anchors in text files or are generic links. Very few authors make links *out of* video and sound. Only a few make them out of bitmapped pictures, but those that do make such links tend to make a large number, as is shown by the technical documentation category above. However, it is very rare to edit such pictures in a way that actually moves the position of an object in the picture. Nearly all link *destinations* (even in text based files) are to the top of the document, which will of course remain the top of the document even after editing. The exceptions to this rule are the generic links into glossary and dictionary files, and into technical documents where many links are to objects within pictures and drawings.

Object based applications, such as AutoCAD, are not a problem, since rather than using offsets to express link anchors, we would tend to use the *name* of the object: the name will remain the same even if the object itself is moved during an edit. Other third party applications are potentially more of a problem. However, again, investigation shows this not to be the case, since most semi-aware viewers only support local and generic links out of the document, and only allow destination links to the top of the documents. Such links are unaffected by editing.

The above analysis points to the conclusion that the editing problem is only severe in the case of text based data: other applications either do not have the problem at all, or the problem is small, and can be resolved by informing the user that the problem has occurred and that links may need manually updating.

Chapter 8. Solutions to the Editing Problem

This chapter explores some of the solutions to the editing problem that have been tried. The first two options (section 8.1 and section 8.2) describe the current options available in the release versions of Microcosm, except that the RTF viewer implements a limited form of the "just-in-time" link repairs described in section 8.5.

8.1. The Publishing Model

A very simple solution to this problem is to adopt a *publishing* model. Once data has been added to the system it becomes read-only, so the editing problem disappears. Many Microcosm applications are produced to deliver what is essentially read-only material such as teaching materials or technical documentation, often using CD-ROM as the delivery media. The contents of the documents in such applications are not intended to be changed by the user: so long as the document contents do not change, then the user is free to make, delete and edit links knowing that the end points referenced within the link will remain unchanged.

The original authors, of course, still have the problem that if they wish to make a new version of some document for subsequent releases of the application, they must manually move all the link anchors from their current position to the new position.

Although this restriction might seem very severe, indeed, not a solution at all, it is worth noting that Microcosm had been in use for over a year before users began to complain about the editing problem. For the class of application for which Microcosm is generally used, the data content *is* fixed. Documents within teaching materials, technical documentation and historical archives, for example, once produced, do not change. New documents may be added to the collection, but the old are not generally edited. It was only when one of our major users started to produce a second edition of their teaching materials that they became aware of the significance of the editing problem.

8.2. A Manual Link Editor

Perhaps the most universal solution, but also the most difficult to use, is to put the onus on the user who edits a document to move the link anchors manually from the old position to the new position.

The current version of Microcosm has two editors that allow users to edit links. The most general in its application is a program called Editman, written by Mark Weal, which allows a user to select a specific linkbase (or DMS file) and to display any field as the key field, then view all the related fields. It is also possible to limit the search space by some query. For example one may ask for all links that have a source anchor in a specific file. Using this system it is a relatively quick job to locate a specific record and then it is possible to delete the record or to change selected fields and to update the record. In this case it would be necessary to update the offset field for the specified anchor. This of course pre-presumes that the user understands *how* to update the offset for the specified data type.

There are obvious limitations to this approach. Firstly, it is clear that the amount of manual effort required to move links around is undesirable. Secondly, the format of a link offset is somewhat obscure and requires the user to have an understanding of the internal representation of a link, which is undesirable when working with naïve users, and an unacceptable cognitive overhead for any user, however good their understanding of the system. For these reasons we have tended to reserve Editman as a tool for Microcosm developers rather than for users.

A second tool, LinkEdit has been developed by Steve Rake for the less experienced Microcosm user. This program only works with ASCII text files and RTF, and understands how offsets work within these formats. It allows the user to open a specific linkbase, and displays the links, by default indexed by description. It allows the user to select all links which have a specified document as source document or destination document (see figure 24), and it is then possible to edit the source selection, source offset, destination selection and destination offset (see figure 25). Some special features have also been added to this program in order to reduce the cognitive overhead on the user.

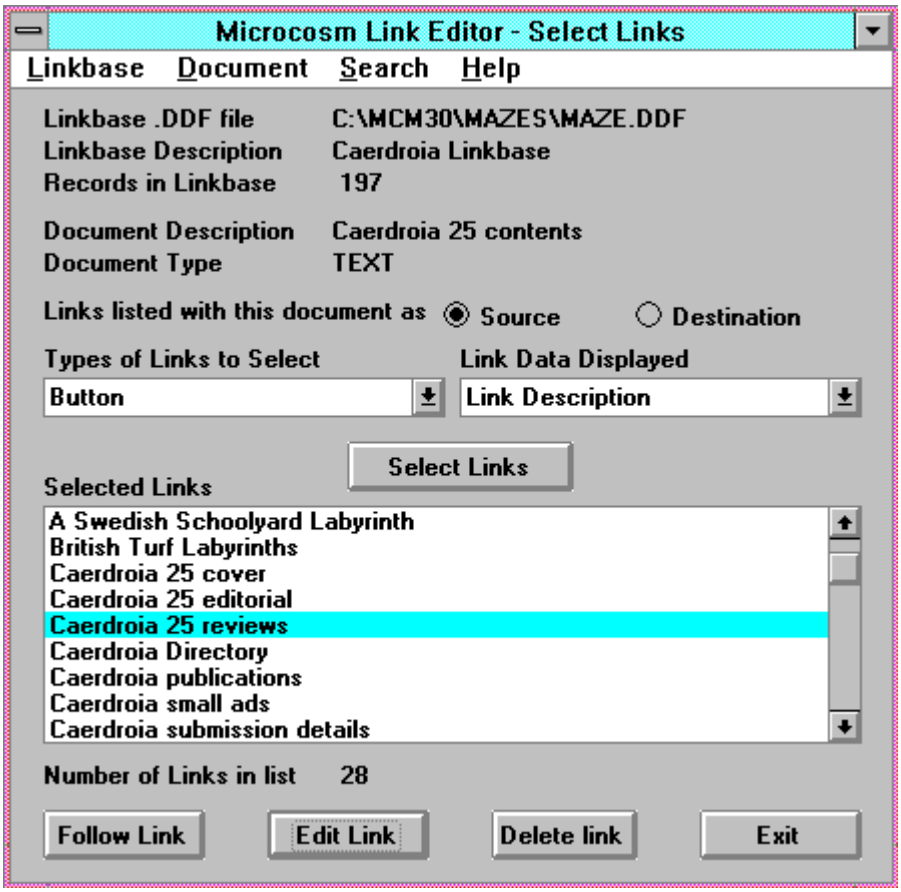

Figure 24: Selecting links in the Microcosm Link Editor: the user has asked to view all button links with their source in a given document, and is about to edit one.

Firstly, there is an option to view the source or destination document at the currently set offset, so it is possible to experiment with changes to the offset until the desired effect is achieved. Secondly there are two options "Fit Selection to Offset" and "Fit Offset to Selection". The first, "Fit Selection to Offset" will set the text of the selection to be the word at the currently set offset, whereas "Fit Offset to Selection" will search outwards from the currently set offset till it finds the text in the selection, and update the offset accordingly.

This system is most useful when you, as the editor of the document, are aware of the links that you are moving. It is quite possible that some linkbases will not be accessible at the time of the edit, so links in these linkbases will not be updated. Subsequently, when the user of this linkbase identifies that changes have occurred, it will not necessarily be simple to relocate the links without an original copy of the document to compare against.

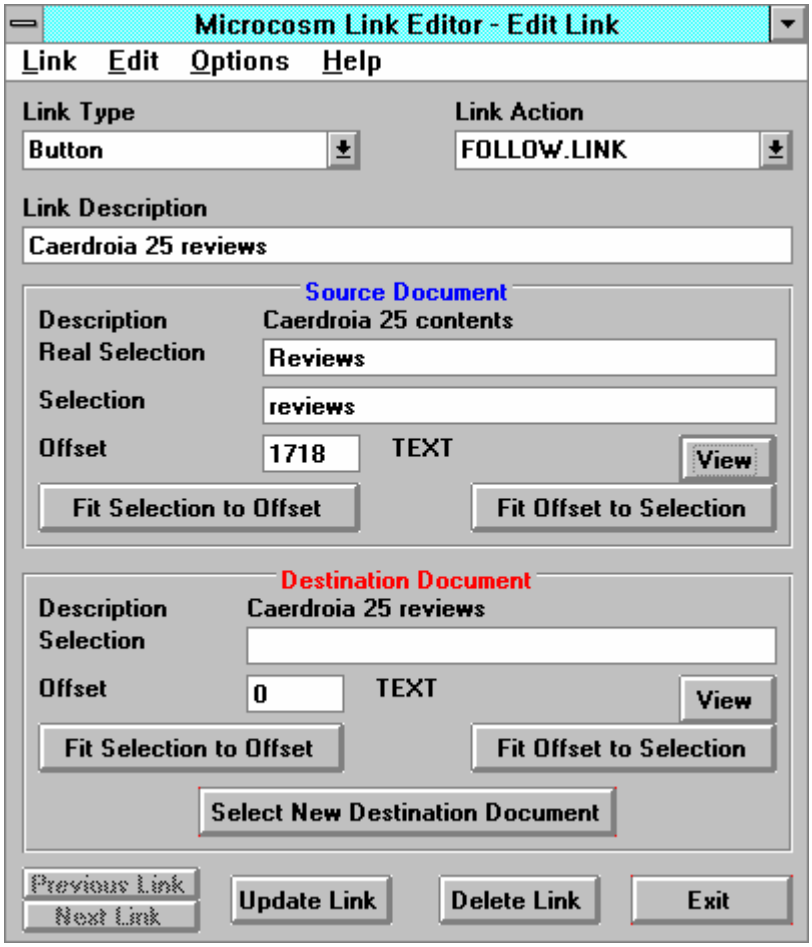

Figure 25: Editing a link using the Microcosm Link Editor.

There is therefore a need to provide tools that allow automatic reconstruction of links that have become broken due to edits, and such tools are discussed in section 8.5.

8.3. Link Service Aware Editing Tools

Perhaps the most obvious solution to our problem is to provide users with link-aware editing tools that understand the data formats in use by the system, and allow the user to edit a document while maintaining link integrity.

At first consideration this option might seem to be the solution to all our problems: after all, those systems which rely upon embedded mark-up provide

such editors. However a second consideration reveals potential problems. It is axiomatic in Microcosm that the system should be able to work with *any* data format. This approach, then, would require that, for every data format in use, we would have to implement a link aware editor. This problem is compounded by the difficulty of gaining access to the proprietary specifications of the data formats of third party applications, and the logical conclusion of adopting such a policy would then be to implement fully aware viewers and editors for all third party applications and their data formats, which would negate one of Microcosm's primary benefits - that of being able to quickly provide link services for new applications.

Another problem with this approach is that the system is unable to tell whether all linkbases containing anchors within the current file are currently available and on-line. It is possible that some linkbases may be held on removable media, and others may be users' private linkbases, held in private filestore to which the system or the editor of the file have no access.

Finally, no amount of link aware editors will be any good if users choose to use unaware editors to change a document.

However, there are clearly cases where such an approach would be the simplest and most useful. For example, when an author is in the process of producing new or revised teaching materials, and has ownership of, and access to, all relevant linkbases, and will subsequently publish the materials in a read-only mode: also when a user is using Microcosm within their private environment, and takes full responsibility for ensuring that files are not updated by any other method, and that there are no external links to the material. As discussed in subsection 7.2.5, by far and away the most important need for such an editor would be with text based files, which in practice usually means ASCII text, RTF and word processed files.

I have extended Word for Windows, using Word Basic, to produce such an editor, which supports link-aware editing for the three major formats of textual data that Microcosm supports, namely ASCII text, rich text (RTF) and Word for Windows files.

The algorithm is as follows:

1. Selecting Microcosm edit mode within Word causes a message to be sent to Microcosm requesting that Microcosm returns a list of all links within the currently installed linkbases that contain specific source or destination anchors within the current document.

2. The links are then stored in a table, each link being assigned a unique identifier.

3. The persistent selection represented by each anchor is turned into a *bookmark*, a supported feature of Word for Windows. Button links, specific links and destination links are displayed in different colours.

4. The user may now edit the file. The bookmark moves around as the text moves. The user may also edit within a selection.

5. When the user elects to save the file, the file is first saved back to disk, in whichever format it was imported from, and the system queries the OS to obtain the file date of the new version. Then, one by one, the system iterates over the links, deleting the old link from the linkbase and creates the new link, with the offset and date fields updated as appropriate for the format.

8.4. Embed the link anchor in the document.

This is the solution implemented by many Dexter model systems, and it has already been stated that if this solution is adopted, then there should be no problem with link integrity. It would violate our first axiom if we required that all link anchors were embedded within the document. However, there are some applications that allow user-defined mark-up to be embedded within the data in such a way that the application can continue to operate on its data in its normal manner when outside Microcosm, while maintaining the position of the mark-up. Instead of sending and storing offsets in the link we operate with unique anchor identifiers (ID's). When an anchor is made in such a document we mark up the selection and assign an anchor ID which is stored in the mark-up. This anchor ID is then stored in the link as the source or destination selection. When a link is followed from this selection, the anchor ID is sent as the source selection and will uniquely match the link in the linkbase. In effect this solution is storing the anchor

identifiers within the links but requiring the application to maintain a table of corresponding persistent selections.

Two examples of data formats that may use this system are RTF and many word processor data formats. RTF allows user defined mark-up which is simply ignored by applications that cannot understand it. The solution therefore is to embed such mark-up within the data at the time that the link is made. The data may then be edited (we use Word-4-Windows as our RTF editor), and saved back to disk after use while maintaining the mark-up. Word Processors often allow the insertion of invisible fields of some kind, and it is possible to use the same technique as with the RTF viewer. I have produced a version of Word for Windows which uses bookmarks in order to store the persistent selections, and maintains a table of anchor ids which map onto the bookmarks. Microcosm needed no changes in order to work with the system.

Axiom 1 (The hypermedia layer should make no assumptions about the storage layer) is often interpreted within the Microcosm community as meaning that no mark-up should ever be placed within documents. It should be stressed that adopting this option is not a breach of this policy. The importance of this axiom is that it is policy not to affect data in any way that makes it impossible to continue to use the data with external applications that expect to handle such data. Thus it would be wrong to embed mark-up in, say, a windows bitmap, as this would make it impossible for the windows bitmap viewer to understand the data. However, where the data format *allows* user defined mark-up it may well make sense to use such a facility.

The main advantage of this technique is that all link anchors (whether within currently installed linkbases or not) will be updated whenever the file is edited, even when Microcosm is not running. However, there are problems with such a technique:

1. It might be assumed that once mark-up is placed within a document this mark-up could be used to allow the host application to indicate button positions. However this is not the case as a button should only be displayed *if* the matching linkbase is currently installed. It is still necessary to enable the application to communicate with Microcosm about *which* buttons should be displayed in a particular session.

2. In an environment in which many links are made in and out of a document by multiple users the data could become very cluttered by mark-up.

3. Since the process of linking actually changes the file, it is necessary to save the file after links have been made. This is different from the usual Microcosm practice, and requires that the user has write access to the file.

For reasons 2 and 3 above we have not implemented the Microcosm aware RTF viewer in this way, but have continued to use external link persistent selections. The RTF viewer is generally assumed to be used as a method of "publishing" documents that were originally produced on a word processor, and it would be undesirable to allow the end user to have write access to such materials.

However, this method does lend itself to use by word processors in a personalised environment where only a single user is making links, and this user has write access to the file and is making frequent updates.

8.5. Apply just-in-time link repairs.

Subsection 7.2.4 introduced an algorithm that allows the Microcosm system itself to identify whether any anchors in a document that is just about to be viewed are *possibly* inconsistent with the current state of the document. Having identified such a situation it would be desirable to identify which, if any, links were inconsistent and attempt to repair them.

Since it is the responsibility of each individual viewer to decide what data to store in a link as the source selection and the source offset, it is not possible for Microcosm itself to make such checks, so the task must be handled by the viewer which understands the meaning of the particular data in the link.

The general principle of this technique is to examine the data at each given anchor's offset and determine whether this data is the same as the data that is stored in the link. If the data found at the offset was incorrect one could then apply some algorithm which attempts to locate the data that was stored in the link and to re-write the link back to the linkbase. For example in subsection 7.2.4 an example was introduced where a link had originally been made from the text span "*Ciliary Movement*" at some specified offset in a text file, but after the file had been edited

the text located at the stored offset was "*ry Movement is ca*". We now know that this link is wrong, and on inspection we might find that the correct link anchor was located at a point 5 characters earlier in the text. We could now write the correct link back to the linkbase.

However, the theory is simpler than the practice. In practice many *destination* anchors are stored as offsets with *no selection* so there is no stored information regarding the data expected at this offset. Also, many source anchors may be very short, for example the digits from a numbered list, which are highly unlikely to be unique within a file. In these cases it is necessary to have access to more information than simply the content of the anchor if we are to be able to have a reasonable chance of repairing the link.

Two methods of improving this situation suggest themselves.

1. When a link is stored a certain amount of *context* must be stored with the link. This is the surrounding information which should be unique within the file, and would enable the system to attempt to relocate a link by searching for this whole context within the file rather than relying upon the data within the anchor which may not be unique, or in the extreme case may not even exist.

2. At the time that the link is stored we should not only store the forward offset (the offset from the front of the file) but also the reverse offset (the offset from the end of the file). This idea makes use of the fact that frequently any edit to a file will have occurred *either* before the anchor *or* after the anchor, but less frequently will edits have occurred both sides of the anchor. With this information to hand it may be possible in many cases to immediately relocate the link without having to resort to searching for the given context. This is the approach suggested by HyTime (1992), which identifies the possibility of the editing problem, and offers this as a partial solution.

The problem, then, is how much context to store in the link. The less context that is stored, the less chance there is of this data being unique within the file. The more context that is stored, the higher the chance that any edit might actually alter that part of the file that was used as the context. The solution that I have adopted is adapted from the HyperTED (Vanzyl, 1993)(Vanzyl, 1994) system. A minimum size for a context is arbitrarily fixed as 10 characters, which are taken from the forward offset position. If these ten characters are unique within the file, then they are used

as the context: if not the string is increased by the number of characters necessary to make the context unique. Note that if the selected anchor text itself is unique and longer than 10 characters, then there is no need to store the context, as the selection will suffice.

Now, when it comes to repairing a link, the algorithm in figure 26 applies. When all links have been checked and repaired it is necessary to give the user the statistics of what has happened and to ask if the user wishes to save the new links back to the linkbases before proceeding.

This solution has the advantage that it makes no assumptions about integrity until a particular file is used, so it allows for the fact that applications outside Microcosm may edit the document in the meantime. However it requires the viewer to perform a significant task, and therefore violates our second axiom (that new applications may be integrated with a minimum of effort). It would be very difficult to adapt a third party application to perform this task, using, for example, a macro language, and is therefore probably only useful in heavyweight fully aware viewers. A further possible disadvantage of this approach is that it only ever mends links that are in linkbases that are actually installed at the time that the document is loaded, and can only mend links in linkbases to which the user has write access.

The release version of the Microcosm RTF viewer uses a variant on this method. It does not store dates, or extended context within the link, but does store both the forward and the reverse offset, and uses the two offsets and the context provided within the stored selection in order to attempt to mend any offsets of *button links* which have sources within the document . The speed of the system is sufficiently fast that we had been delivering this functionality as an undocumented feature of the release version for many months without any user noticing any decrease in speed.

A number of hypertext researchers, on hearing about this method, have commented that it can never guarantee 100% integrity of all links. This is true. However, if we postulate that the method can fix 99% of broken links (a pessimistic figure) then it seems a highly worthwhile addition to the system, so long as the user is warned of changes that have occurred and is signalled that there were links that could not be repaired.

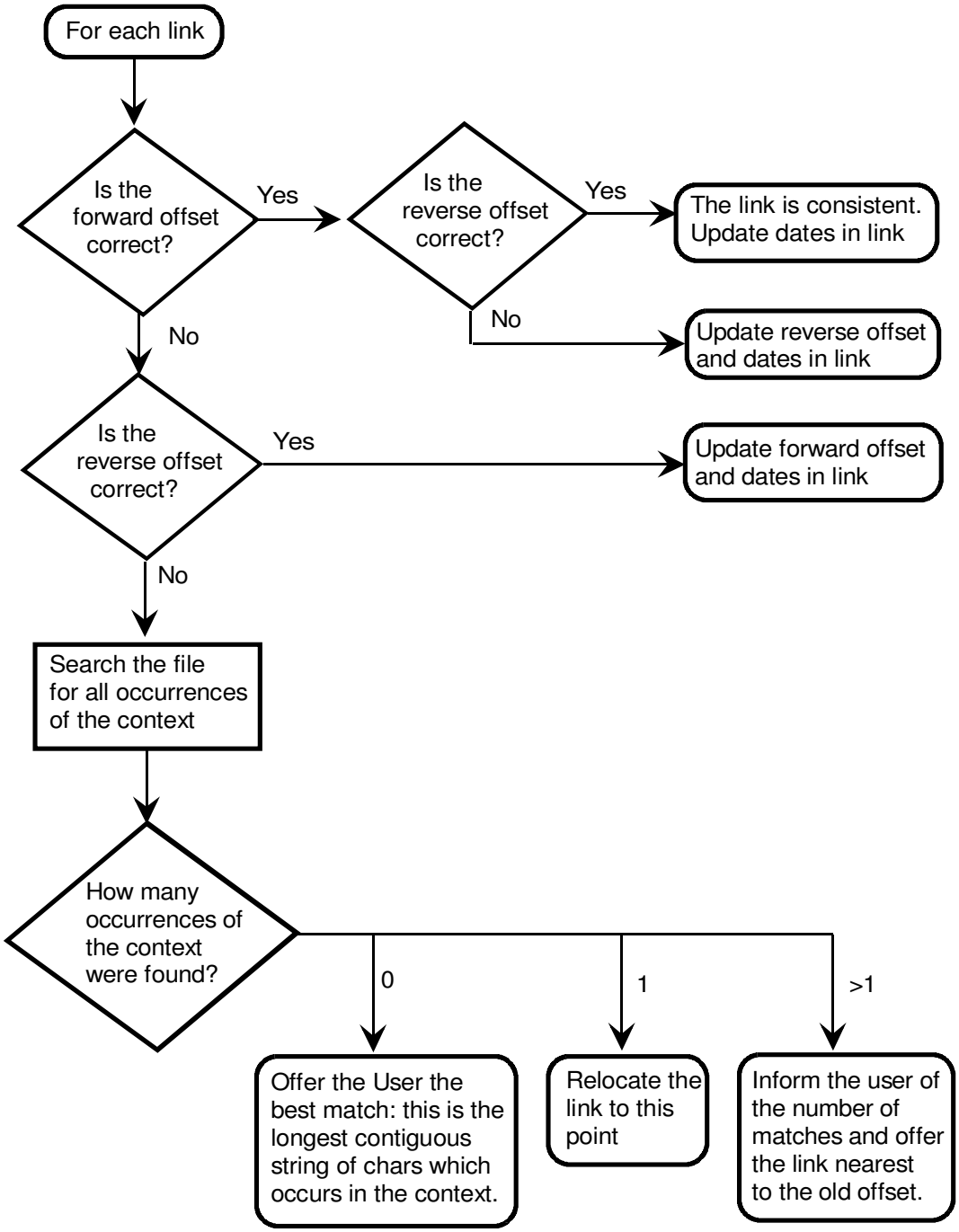

Figure 26: Link repairs using context and offsets from both ends of a file.

It is worth noting at this point that this method is only possible since Microcosm stores, for its link anchors, both the offset *and* the data expected at this offset (the context). Systems which try to keep the link service layer free of any data supplied by the application layer will not be able to use such a method.

8.6. Maintain a Shadow File

A variation that combines the advantages of the above two techniques is to maintain a "shadow file". This is a file which contains the important details about anchors within any given document, and relates these anchors to a link identifier in the same way that the embedded link contained a link identifier, as shown in figure 27.

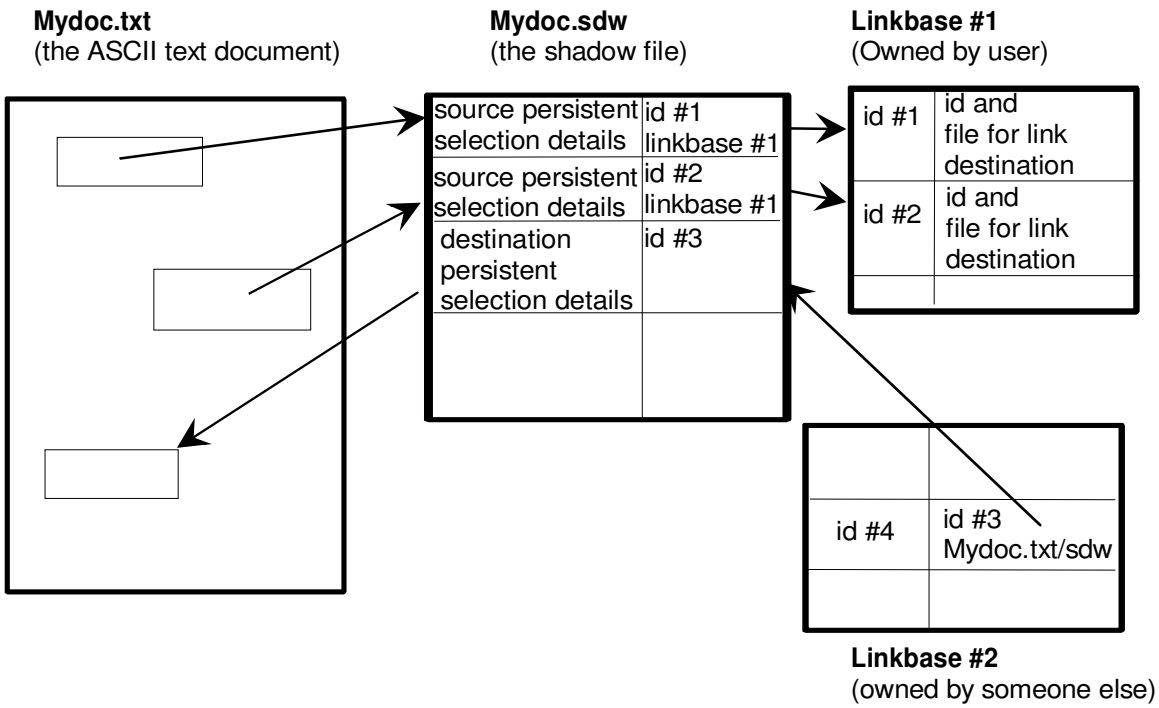

Figure 27: Using a Shadow File to relate selections to anchor id's.

In this system, Microcosm always queries the appropriate shadow file for *specific* link anchors, and will then use this information to find the appropriate anchor from the appropriate linkbase (for source anchors) or to find the appropriate persistent selection (for destination anchors). The shadow file will hold all the information that a standard Microcosm linkbase might hold to describe a source or destination persistent selection, i.e. the selection, the offset, the file and any context. It will also keep a single copy of the related file's system date at the time that it last checked the integrity of the links.

Now, a link-aware editor may be implemented, which maintains the information in the shadow file. However, it would still be possible to edit the data using some unaware editor such as a standard text editor. In this case the shadow file would not be maintained, but when the file was next loaded by the Microcosm viewer it would be possible to apply just-in-time link repairs, as specified in section 8.5.

This combination of link aware editing and just-in-time link repairs has some advantages.

1. Whenever link repairs or link-aware editing are applied, *all* link offsets will be repaired - even those in linkbases which are not currently connected.

2. There is no requirement to have write access to the data file, as was the case where links were embedded in the data itself.

3. The check to see whether link repairs must be applied will be considerably faster, as all that is necessary is to check whether the file system date held by the shadow file is the same as the current file system date of the data file: there is no need to query the on line linkbases for relevant links, then check each one.

4. The speed at which link repairs may be carried out, or a link aware editor may be started, will be faster, as there is no need to send a query to all the on-line linkbases asking for the links which have specific source or destination anchors in this document, as this information is exactly what is held in the shadow file.

5. The requirement to carry out link repairs is likely to be less frequent, as whenever link repairs are applied, all anchors will be updated. The case where a new linkbase is introduced and links must be repaired again will not occur.

6. This method does not enforce the use of one particular editor.

There are also some of disadvantages:

1. The data content contained in link anchors will no longer be inside the linkbase. One of the advantages of keeping all such information inside the

linkbase is that it is possible to make queries of linkbases of the kind "Show me all links that contain the word *disintermediation* in their anchor". The required information is now inside the shadow files, and the query will have to be applied to all the shadow files, sacrificing the advantage of having this information centralised.

2. There is potential for introducing a new consistency problem if the shadow file became "detached" from its data file, for example during re-organisation of the file store.

3. Generic links would still have to be implemented from the central linkbase. This introduces an ugly asymmetry into the system, where all FOLLOW.LINK type queries would have to be applied to the shadow file (to get the specific links) and then to the linkbase itself (to get any generic links). Producing tools, such as the manual link editor, would become increasingly more complex.

This method is, in effect re-introducing the distinction between links and anchors, and its major advantage is that it makes it possible to update links, even when the relevant linkbase is not connected. If Microcosm had been implemented so that all links were held in one central linkbase, but attributed with the name of the virtual linkbase to which they belong, then this problem would not exist, and the main remaining advantage of this method would be the speed increases when querying for relevant links.

This method was the method that I used to implement links in my prototype Microcosm. It has never subsequently been implemented due to the problems of implementing generic links in this system, but it is possible that it would provide a much faster, and possibly more robust, system for dealing with editing problems in text than is provided by the simple link-aware editing method or the just-in-time link repair method on their own.

A variation on this approach is taken by the Hyper-G system (Flohr, 1995) (Andrews et al, 1995), which maintains the shadow file as one of the attributes of the document within its object oriented database. Hyper-G documents may only be accessed by Hyper-G clients, which then embed the anchors into the data at the client end before any editing occurs, so this system has the advantage that it ensures complete integrity of its links, while at the same time providing many of the advantages of separating links from content

All of the methods outlined so far in this chapter (with the exception of the publishing model) attempt to provide tools to maintain or fix link anchors that are expressed as specific offsets into a data file. The next two methods attempt to solve the problem by avoiding it altogether.

8.7. Avoid specific links anchors.

Specific anchors are those which are situated over a specific object at a specific position within a specific file: in Microcosm terminology these are specific links, buttons, and destination anchors. Generic links, local links, and destination links which are made to the top of a file, rather than to an offset within the file, do not suffer form the editing problem: they are edit-proof.

To do without buttons, specific links and specific destinations might seem rather a drastic loss of hypermedia functionality. However, the quality of hypermedia link service that may be achieved using such a system deserves closer inspection.

In the paper *Ending the Tyranny of the Button* Professor Wendy Hall (1994) draws attention to the fact that clicking on buttons as the primary or sole method of browsing multimedia archives leads to a passive form of interaction. Query based navigation of the sort where the user selects an object of interest and asks the system, "have you any more information about this?" leads to a very much more active and involved navigation.

In the paper *Light Hypermedia Link Services: A Study in Third Party Application Integration*, Davis et al (1994b) examine the quality of hypermedia system that can be produced using the Universal Viewer (see section 5.5) which cannot handle specific link anchors. Generic links and local links are provided: buttons may be attached to the top of a document (rather than to a specific object within the document) and link destinations must either be to the top of a document or to a macro which locates the correct object, for example by using the application's built-in search engine (see the next section). The paper *A Process Approach for Providing Hypermedia Services to Existing, Non-Hypermedia Applications* (Kacmar, 1995) also introduces the idea of re-parenting an unaware application's window, and

providing an area at the top of the new parent window in which descriptions of links generally applicable to the document may be placed.

It appears that there is an emerging body of opinion that says that specific links and buttons may be generally unnecessary, and could be considered to be over-kill. A light hypermedia link service providing links between document nodes (rather than between objects within nodes) may be all that users require. Further evidence for this viewpoint may be gained by examining a number of linkbases, as shown in table 4 in subsection 7.2.5. Even in the tutorial applications nearly 50% of the links are edit proof.

8.8. Express specific link context in an edit-proof manner.

If it were possible to link to objects that are separately and uniquely identified by the application concerned, then moving objects around within the node would not change their name, so it would be possible to make anchors with the "offset" expressed as the *name* of that object.

Naming of objects is the method of linking used by SGML, and one of the methods supported by HyTime. It is also the approach taken by a number of open systems, such as PROXHY (Kacmar & Leggett, 1991) and the Sun Link Service (Pearl, 1989). However, we have only so far found this approach useful in object oriented drawing programs such as AutoCAD (see subsection 6.3.2) and spreadsheets (see subsection 6.3.4). The Sun Link Service attempted such an approach with text by treating each individual line as a pseudo-object, which then begs the question, if an object was known as "line 104" before an edit, what text will be referenced after an edit that adds an extra half line using a word wrapping editor?

The modern trend of operating systems and applications towards object orientation such as OLE 2.0 OpenDoc and CORBA (1991) may in the long term provide increasingly sophisticated support for the identification of objects within the workspace, but the extent to which this will be useful to hypermedia developers would depend on how finely grained the objects are, within the file as a whole.

Another approach to specifying link anchors in HyTime is to specify anchors in terms of queries that will identify the object. An advantage of this approach, is that as well as being edit-proof, it enables links to be specified dynamically, so that when the contents of a document changes, the link will automatically reference all correct objects as specified by the destination query at the time that the query is made.

A variation of this approach, for specifying anchors within a text document is adopted by HyperTED (Vanzyl, 1993) and is described in Vanzyl et al. (1994). This involves using a text search engine to find the anchor. When an anchor is created it is stored, along with sufficient context to ensure that the anchor is unique within the text. So long as this fragment of context itself is not edited, the links will be safe from other reorganisations, additions and deletions within the file. We have made use of this approach in our Universal Viewer (subsection 5.5) to provide link destinations - which are effectively links to the first occurrence of a specific object within a file - by storing keystroke macros which activate the host application's search engine. I have also added "Search links" as an option within word for windows, as an alternative to offset based links.

8.9. Editing Bit Stream Data

The vast majority of the discussion in this chapter has been about editing text based applications. This approach may be justified by pointing out that bit stream data such as bitmapped pictures, digital video and sound files are rarely edited in such a way that any specific offsets would change. Cases where this might happen are where an editor has been used to reduce a picture to a smaller size (although there should be little reason to do this since the bitmap viewer allows dynamic scaling of pictures) or some frames have been cut from a video, or an introduction added to the front of a sound track. It would probably not be economically sensible to produce link aware editors for such media. Just-in-time link repair relies upon having some context stored in the link. Although this is possible, representing such context in ASCII within a linkbase would be storage intensive, and the search time to re-locate the context for a number of links in large multimedia files might be too large to be practical.

However, the feature of link editing in such media, which does not generally apply to text, is that all link offsets, at least within some given area, will be affected by the same amount: if a picture is clipped then the x and y co-ordinates will all shift by the same amount: if it is re-sized, then the same multiplier would apply to all links: if frames are removed from a video, then the time at which all links should apply will be reduced by the same amount. This motivates the idea that it might be possible to implement a manual link editor, which allows the user to identify all links which will have been affected by a given edit (e.g. all links after the point at which frames have been removed from the video) and to modify them all by the same multiplier or adder.

In practice I have only come across one case where a user needed to make wholesale edits to such media on a large enough scale that it was impractical to delete old links and to re-make the links into the new version of the document. In this case a large number of electrical circuit diagrams had been scanned from their paper versions, and these diagrams had been intensively linked with generic picture links. At a later stage the application authors had obtained much better copies of the original diagrams, and these had been scanned, sometimes at different scales and with different origins. The task then was to locate the same object within the new version of the picture. A project student is currently investigating using feature extraction image processing techniques to identify the best possible matches in the new version, and is attempting to produce a link mending tool for bitmaps.

8.10. Versioning

A further approach to the editing problem is to keep separate copies for each version of a file, so that link integrity is maintained by ensuring that each link is linked to the version of the file in which it was originally anchored. This topic is discussed in detail in chapter 10.

8.11. Use of Diff Files

A "Diff" file is the output of the *diff* program which compares two versions of a file and produces an output file which describes the difference between the two

versions, in terms of deletions, insertions and changes that would need to be made to the first file in order to arrive at the second. Given the existence of such a file it would be possible to calculate exactly the movement of any particular offset from the first file to the second.

Unfortunately the standard *diff* algorithm works in terms of changes to whole *lines*, which is not the ideal solution for reconstructing offsets in terms of bytes through a file. For example, a particular anchor might actually be within a line that has been edited: all we would know would be the text of the old line and the text of the new line, and we would then have to resort to text searching in order to recover the exact offset within this line. This situation is likely to occur frequently as it is often the case that whole paragraphs are represented as a single line, devolving the word wrapping to the viewer at run time, in which case any edit within this paragraph will be represented by *diff* as a change from one line to another.

Fortunately, there is another version of *diff*, known as *diffb* which produces similar output to *diff*, but works in terms of bytes through the file. I have produced a program which uses the output of *diffb* to produce the position of any byte offset in a file in a new version of that file.

This method depends upon the existence of two versions of a particular file being available to repair. There are two ways that this might be used:

1. In a system which supports versions of documents the links will be tagged with the version of the document into which they are anchored. When a user wishes to move links from an old version into a new version they may run a program which will generate the *diff* file between the two versions, and automatically update all the links. This would suggest that *RCS* should be used as the file storage layer, as RCS is a system which anyway maintains the original file and that set of *diff*'s necessary to reach each version of the file.

2. In a system that only supports one current version of a document, authoring users might be persuaded to produce the *diff* file at the time that the update edit was made, and leave dated tags in the docuverse entry for the file, pointing at the *diff* file representing the diff between the latest version and the one they edited. When the system discovers out of date links into the file it is necessary to back track through all the *diff* files building up a *diff* between the

current version and the version into which the original links were made, before applying the updates.

The algorithm works as follows. The *diffb* program produces three different types of messages, e.g.:

110,114d118	The 5 bytes at 110 to 114 through the old file have been deleted. The byte that was at 115 is now at 118 (due to the cumulative affect of previous edits);
38a40,42	Three new bytes have been added, starting at the 38th character through the old file. They are at bytes 40-42 in the new file.
110,114c112,120	The bytes 110 to 114 through the old file have been changed, and the replacement spans the bytes 112 to 120 inclusive in the new file. This represents a net addition of 4 bytes to the file.

By scanning through the *diff* output file and identifying the last edit before the old offset in the old file, it is possible to calculate by how many bytes the offset has changed. For example, if we were looking to fix an offset that was 132 bytes through a file, and the last edit had been 110,114c112,120, then we would know that all offsets, in the region after this edit until the next edit, had been increased by 6 bytes. Of course, decisions must still be taken about what to do in the case where it turns out that the offset being fixed is actually within an edit that deletes or changes the old offset, so this algorithm, although very fast and generally reliable, can still not guarantee to fix 100% of offsets.

This method may be used as a form of "just-in-time link repairs", as described in section 8.5, and provides a high degree of reliability in fixing links and does not need the context to be stored in the link, but depends upon the user who edits the file arranging to either save a version or to save the *diff* file. Just-in-time link repairs based upon context make no such requirement and may even be useful in the case where the document has been edited without any reference to that fact that it is being used within the hypertext system.

Keeping diff files between different versions of documents, and using these to affect “just-in-time link repairs” when a document is next viewed through an active Microcosm session is perhaps analogous to the situation that may occur in distributed databases when a link to a remote server goes down. In this case the

system maintains an audit trail, and this audit is used to update remote tables when the connection is recovered.

8.12. Summary

Chapter 8 has examined a number of approaches to the solution of the editing problem. None of these solutions is complete, in the sense that it can be used to solve the problem for all data types and work with all applications, so the solution that is adopted must depend upon the particular circumstances, and this may well depend upon the type of application being used. Generally speaking Microcosm applications fall into two categories.

1. Published Information Systems. These are sets of teaching materials, electronic journals, technical manuals etc. In these cases the end user is not expected to edit the documents in the system. Any editing that is done will be done by the owner of the application, who will, in most cases, have access to all linkbases used by the system. The author is most likely to have used fully aware Microcosm viewers for all data types. In these cases a fully aware text and RTF editor is a perfectly acceptable solution to the problem. Where other users are making links into and out of the owner's files, then the fully aware text editor with a shadow file could improve upon the situation.

2. Personalised Desktop Information Systems. These are the applications that individual users create in order to enable them to find and navigate through the large bodies of information that they collect in their computing environment, and that others put on file servers for them to access. These applications tend to be much more dynamic. It is quite possible that links will be made to other users' documents and those other users may update those documents at any time. The viewer applications used will tend to be third party applications such as word processors and spreadsheets rather than fully aware viewers. In these cases the editing problem is much more severe.

   The best solution here is to use linking techniques that do not require specific link anchors. Generic links, local links, and iconic buttons on the application title bar provide a perfectly acceptable linking mechanism and, when

combined with other navigation methods such as computed links and attribute searches, provide an excellent navigation environment.

It is interesting at this stage to speculate about how the editing problem in Microcosm might be improved if the system had been implemented as a hyperbase, with as single database server being responsible for the maintenance of all the nodes and links, perhaps as some virtual disk system (see section 9.3) which would still allow third party applications to access their data from this hyperbase. In this situation the hyperbase would always know when a document had been edited, and the hyperbase would always have access to all the links. (Presumably the hyperbase would implement permissions so that users could only edit documents and links to which they had access rights). Under these circumstances:

1. Link aware editing is a more practical solution, since it will be a simple matter to make a query to the hyperbase asking for all link anchors in this document, rather than just the set in the connected linkbases.

2. Perhaps more usefully, it will be possible to apply "Fix-on-Save". In this scheme, at the time that a document is submitted to the hyperbase for saving, the hyperbase will have both the old version and the new version of the document. It will now be possible to apply both *diff* based techniques (section 8.11) and content based techniques, such as those used for "just-in-time" link repairs and for the bitmap link repairer discussed in section 8.9. This pre-supposes that there will exist a link repair algorithm for each data type in use, which is impractical, but where such algorithms have been produced it is reasonable to expect that they will be more accurate than is possible when only the latest version of the data is available.

Chapter 9. Solutions to the Dangling Link Problem

The dangling link problem was introduced in subsection 7.2.2, and occurs when a document that contains a link anchor is deleted, moved or unreadable. In practice this problem is most severe when a document containing a destination anchor is deleted, since users may select the source anchor and then be informed that the destination anchor is not available. This is frustrating as the users know that there was intended to be a relationship, and they were interested in discovering that relationship. A missing source document will never be noticed by the user unless the system implements bi-directional links, in which case, in this context, it is still a destination anchor.

The dangling link problem is easier to deal with than the editing problem, since there is no need to know anything about the content of the document - only it's whereabouts. The following analysis considers methods of dealing with the dangling link problem in any link service hypermedia system.

9.1. User Onus

The simplest solution is to put the onus on users not to move or delete documents that are accessed by a hypermedia application, or to put the onus on the user to update the system to reflect any changes. This approach seems so simplistic that it might hardly deserve mentioning - except that it is the method currently used by many systems, including the World Wide Web and Microcosm.

The problem with this approach is that a user may not know that the document is part of a hypermedia application, and even if they do, they might not know whether there are currently any links to the document, since the link is held elsewhere (in another document on another server in the case of WWW, or in a linkbase in someone else's filestore in the case of Microcosm, for example).

In the case of the World Wide Web the problem is not solvable, since it will never be possible to examine the universe of all documents on all servers in the world to see if they contain a pointer to a document that is about to be moved or deleted. All documents must remain in the same place forever, once they have been

made available on a server. In an attempt to solve this problem an Internet Draft of July 1994 proposes the adoption of Uniform Resource Names (URN's) to replace URL's. These would be globally unique names for each resource published on the Internet, which would be allocated in much the same way as IP addresses are currently allocated, in order to guarantee uniqueness. Once resources were described using URN's it would be necessary to provide some directory service which would map a given URN on to a URL which would actually locate the file. It would be the responsibility of the owner of the resource to ensure that the directory service provided the correct mapping at any time that their filestore was re-organised. It is exactly this functionality that is provided by Microcosm's Document Management System (DMS).

In the case of a link service that keeps all its links in one linkbase or in a set of linkbases that are always available to the system, it would be possible to provide a tool to search all links for references to the document, and change the name of the document to point to the new location, or remove the link altogether when the document is deleted. However, this would not be possible in Microcosm, because linkbases may be held on removable media or private filestore. Instead we have implemented Microcosm to allocate unique identifiers to all documents, which are held in the DMS along with the actual location of the file. Links do not refer to the filename but to the unique identifier, which indirects through the DMS in order to locate the actual file. This simplifies the problem from being one of changing every link containing the document to being one of changing a single entry in the DMS. However, the responsibility is still with the user to make this change or dangling links will occur.

An interesting approach to ensuring that owners of documents take responsibility for keeping links up to date with document movements in a wide area link service system has been proposed by Antoine Rizk (Rizk et al, 1994) who suggests that all link requests on the wide area network pass through a "Kiosk" which will *charge* users for access to the link. Value added service providers might pay to have their links in the linkbase, and will receive payment every time the link is traversed to their document. Under these circumstances it seems likely that document owners will ensure that their links do not dangle!

9.2. The Closed Document File Structure.

One solution to this problem is to organise the file structure in such a way that it is not possible to move or delete documents except through link service aware tools. This was the approach taken by Intermedia (Haan et al, 1992) which maintained all its documents under a specific directory in the filestore which was owned by Intermedia: only Intermedia itself could change this filestore. Thus applications which move or delete files must be written in such a way that they pass a message to a tool which updates the linkbase or DMS as appropriate.

We have not adopted this solution in Microcosm, as axiom 5 of our design requirements stated that the native file system would be used as the resource storage layer so that the hypertext layer would be seen as a lightweight extension of the user's normal working environment. Putting all documents into an area owned by the hypermedia system prevents them from being accessed and used by tools outside the system.

9.3. The Virtual File System

An extension of the concept of using a closed file structure is to introduce a virtual file structure which is in effect a layer above the real file system to which files may be written. Many hyperbase implementors have adopted this approach, and refer to this layer as the hyperbase layer (see subsection 2.4.2). For example GMD-IPSI's SEPIA makes use of a hyperbase layer called Hyperbase (Schütt & Streitz, 1990) which was extended to CHS (Streitz et al, 1992) both of which are implemented on the Sysbase RDBMS: the Hyperform hyperbase (Wiil & Leggett, 1992) is currently being re-implemented on top of the Exodus OODBMS. Others have implemented the hyperbase layer form scratch e.g. ABC's Distributed Graph Server (Shakelford et al, 1993).

The advantage of using a database system to implement a virtual file system is that one may make use of the built in facilities for maintaining integrity. For example, in this case, the database schema to represent the hypermedia structure model may state that a link connects two nodes. The DBMS may then be relied upon to carry out integrity checking and to enforce consistency.

The problem with using a virtual file system is that files that are committed to the hypertext system will become inaccessible to applications which cannot interface to the hyperbase layer, and for this reason we adopted the principle that the host file system would be used as the hypermedia layer (axiom 5).

However, we have recently begun to reconsider this axiom. Mylene Melly has implemented a version of Microcosm on X-Windows using the Exodus OODBMS as the storage layer in order to inherit the concurrency control necessary for Computer Supported Co-operative Work (Melly, 1994). At the same time we have become increasingly aware that in real commercial organisations there is often a need for greater control of the filestore than is provided by common operating systems.

A solution to this problem is to implement a virtual filestore that appears as a mountable server disk from the user's perspective, but allows the owner to specify the behaviour of the filestore in a more detailed manner than the normal operating system provides. For example, the Double-space system, now integral in MS-DOS 6, compresses an entire file system into one large file and then presents this file to the user as if it were a disk in its own right. Users may read and write to this disk in a transparent manner, but in reality all disk accesses are interrupted and processed by the Double-space software.

A number of commercial document management systems are now starting to emerge which adopt a similar approach. The Documentum system (a spin-off from Xerox Parc) is one such system: it allows users to access files stored on a Unix server from multiple clients (including Windows, Macintosh and Unix) in an almost transparent manner. The system allows owners to associate attributes with documents, including long English descriptions, provides a versioning system and provides a uniform access control system, which is considerably more sophisticated than any of these operating systems provide. There is a sizeable programmer's API which enables tailoring of the system. Documentum is implemented as an SQL database which handles the document attributes, access, concurrency control etc., and a filestore which is owned by Documentum, and can therefore only be accessed through Documentum.

In cases where such a system is already in use, we do not see that making use of it as the hyperbase storage layer offends axiom 5: in effect it *is* the host file system. We are currently planning to adapt a version of Microcosm to use Documentum to

replace the Microcosm Document Management System, and to provide the storage layer.

The advantage of using such a system, in the context of this chapter, is that the system will ensure that dangling links never occur. Whenever a document is moved the DMS entry will be updated automatically. The system may be programmed to invoke certain actions whenever certain events occur: it would therefore be possible to invoke link editing tools when a document is deleted or when a document is edited.

9.4. Operating System Extensions

In the previous section we identified two pre-requisites in order to avoid the dangling link problem:

1. The DMS needs to be able to track the movement of files;

2. The Hypermedia system needs to know when files are deleted so that it may fix any links that are affected.

In the ideal world the operating system itself would provide facilities to enable us to achieve this. For example, the Macintosh system provides aliases for filenames, which enable the user to move files while they retain the same aliases, but since such facilities are not available across all platforms, we were not able to rely upon this facility (axiom 6).

A facility that we would very much like to see built into operating systems would be the concept of an "interest set". Users could register interest in certain events in the operating system, and nominate the process that would be invoked when such an event occurred: in the case of Microcosm we might invoke a link integrity checking algorithm whenever a document was deleted or edited, and a DMS editor whenever a document was moved. We have investigated implementing such a facility for Windows, making use of the *WindowsHook* calls. However such a system would be limited by the fact that it could only trace filestore alterations made by the Windows File Manager, and would not be aware of alterations to the file system made by other utilities or applications.

9.5. Link Integrity Checking Tools

All the approaches discussed above attempt to prevent the dangling link problem from occurring by ensuring consistency. The final two methods rely upon fixing the problem when it occurs.

A simple solution is to provide off-line link integrity tools which examine all link anchors in a given linkbase, and after querying the DMS for the actual document position, check that the document is actually present. In the case where a document is not present there are three options:

1. Offer to delete the link containing the anchor.

2. Offer the user the opportunity to manually locate the document. This is the approach that is taken in the current version of Microcosm.

3. Use an automatic search engine to attempt to locate the document. (see section 9.6)

An advantage of this approach is that it will identify links that dangle at their source end, which will never be discovered in the course of normal link navigation. In this sense it may be used as a link garbage collection tool.

The LinkEdit program discussed in section 8.2, contains an option to identify dangling links, but at present all it does on discovering a dangling link, is to notify the user and offer to remove the offending link. This tool will check all links in a given linkbase to ensure that all documents are anchored in a document listed in the DMS, and will then check that the DMS entry actually points to a real file within the file system.

9.6. The Search Engine

This approach relies upon storing enough information about the file to locate it again after it has been renamed. The most universal solution is to store the file length, date stamp and time stamp in the DMS. These features may then be used to find the file again with a high degree of confidence. The problem is how far to extend the search if the file is not found on the disk on which it was previously

located. A suggested answer is to search all disks on which there are files mounted in the current application, but user control will be necessary.

A search engine might be invoked whenever a link was found to be dangling, either at the time a link was followed or when a link integrity check was being conducted.

9.7. Summary

Dangling links do not present the same difficulties for Microcosm that are presented by the editing problem. Given the right environment, such as a closed or virtual file system, it is possible to provide tools which prevent the problem from occurring in the first place. Even when they do occur, any system that implements links externally from the content data will be able to batch process those links to identify dangling links. Mending dangling links may be possible, but depends upon there being some bounds to the size of the search area within which one might reasonably expect to locate the file.

Perhaps dangling links are a social problem rather than a technical problem? In a library people are expected to check their books out when they remove them. Failing to do this will leave the library systems pointing to a source that is not present. The person who fails to check-out a book has behaved anti-socially to a greater extent than just stealing a book: they have also broken the system. Similarly, when we make files available on a server, for others to link into, we have contributed to a system. If at some later stage we move the server or the file, then we have broken that system, unless we adopt some method of pointing users on to the new version.

From another point of view, one could argue that dangling links are not really a problem. If a link dangles it does no more harm than to disappoint the user: the editing problem is far more severe, as it can lead to the situation where link anchors are positioned over objects which bear no relation to the link.

Chapter 10. Versioning

10.1. Why Use Versioning in Hypermedia?

In his paper "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems", Halasz (1988) identified versioning of hypermedia systems as an issue that the systems of the day had failed to address. He saw versioning as important for two reasons:

- To allow exploration of the history of the evolution of a particular hyper-document.
- To enable the exploration of alternative configurations.

Three years later in his keynote speech to Hypertext '91 (Halasz, 1991) he commented that very little activity had taken place in this area and asked whether versioning was, after all, not an issue for hypermedia systems. He concluded that perhaps users did not require versioning, either because the nature of their applications did not require such functionality, or because users have been so used to living in the DOS, Unix and MacOS world, that they didn't know what they were missing. Another possible explanation is that the cognitive overhead of arranging versions is too high for the average user who comes to hypertext as a simple lightweight method for arranging information.

However, within the past three or four years, considerable new work has been conducted in the area, and it is clear that industrial strength hypermedia systems will be required to provide some form of version control.

Malcolm et al. (1991) believe that all versions of objects within a hypertext project must be under configuration control in order to enable auditability. Ted Nelson in his book Literary Machines (Nelson, 1981) identifies the need for versioning to provide safety (undoing unwanted changes) and intellectual backtracking, in order to understand the process by which a particular hyper-document was derived. Hypertext projects that involve software development or engineering design require versioning for safety and to enable exploration of

alternative designs. The HAM system (Campbell & Goodman, 1988) is such an example, having been used as the hyperbase layer for CASE applications and CAD applications.

However, versioning is not only useful to provide safety, backtracking and exploration of alternative designs: recent trends towards the use of external link anchoring models, such as Microcosm and RHYTHM (Maiolo et al, 1993), have led researchers to realise that versioning may be employed as a mechanism for ensuring data model integrity. It is this aspect of versioning that is of particular relevance to this thesis.

10.2. Versioning in Software Development Environments

Versioning and configuration management have traditionally been the province of software development environments. This section identifies the techniques and vocabulary that are in use within this community, and the next section identifies the problems with transferring these techniques to hypermedia systems.

Each module in a software engineering project may be saved as a different *version* as edits are made. All the versions of a module are grouped together in a *version group*. Typically a version management system, such as RCS (Tichy, 1985), will store one base version of the module, along with a set of changes, or *deltas*, which describe the edits that must be made to the previous version in order to arrive at the new version (or vice-versa). The version management system will normally also store attributes with the delta, that describe the reasons for the change, the author, and the date of the change. In order to avoid concurrency problems due to lost updates, such systems usually provide a locking mechanism, which will only allow the first user who checks code out to check it back in again. Other users may check code out, but will be unable to check any changes in, unless they begin a new branch to the version tree. RCS does provide a *merge* program which attempts to merge two branches of the version tree back together again, but our experience of using this system for the development of Microcosm has shown that such a merge rarely has the desired effect.

A complete software system will consist of code from a number of modules. A *configuration* describes, usually in terms of *selection criteria* based on dates, authors

and other attributes, which modules are to be used in a *release* of the system. A *baseline* describes the exact versions of each module that has been delivered in each release of the system.

10.3. Versioning Problems Particular to Hypermedia

On the face of it, it would seem reasonable to simply adopt the methods described in the previous section to version components of hypermedia systems. Indeed many Hyperbases, e.g. HAM (Campbell & Goodman, 1988), CHS used with Sepia (Haake, 1992) and HB3 (Hicks et al, 1991), incorporate a versioning engine which provides such functionality. However, Davis et al. (1992) point out that it is not always this simple. When a new version of a node is created, which node should the links now point to? When a new link is created in the most recent node, should it also be retrofitted to the old versions of the node? Østerbye (1992) divides the problem of applying simple versioning of components to hypertext systems into two categories: *structural* issues which are those related to the hypertext data model, and *cognitive* issues which are those related to the user interface. He identifies five particular problems under these two headings.

1. *Immutability*

   The standard model of versioning assumes that once a version has been created it is immutable: that is, it is no longer possible to change this node in any way: it is frozen. Østerbye argues that it would be unreasonable to assume that just because a node is frozen, that users would not wish to continue to link to and annotate that node. In systems which embed link anchor information within the node this would not be possible, since each linking action would physically change the node itself. On the other hand, where link anchors are stored externally, it might be the case that making certain types of links might actually change the meaning of the node. Would this be desirable? What sorts of link making should be frozen?

2. *Versioning of Links*

   In a hypermedia system we would possibly need to address the versioning of links as well as nodes. Three cases are identifiable.

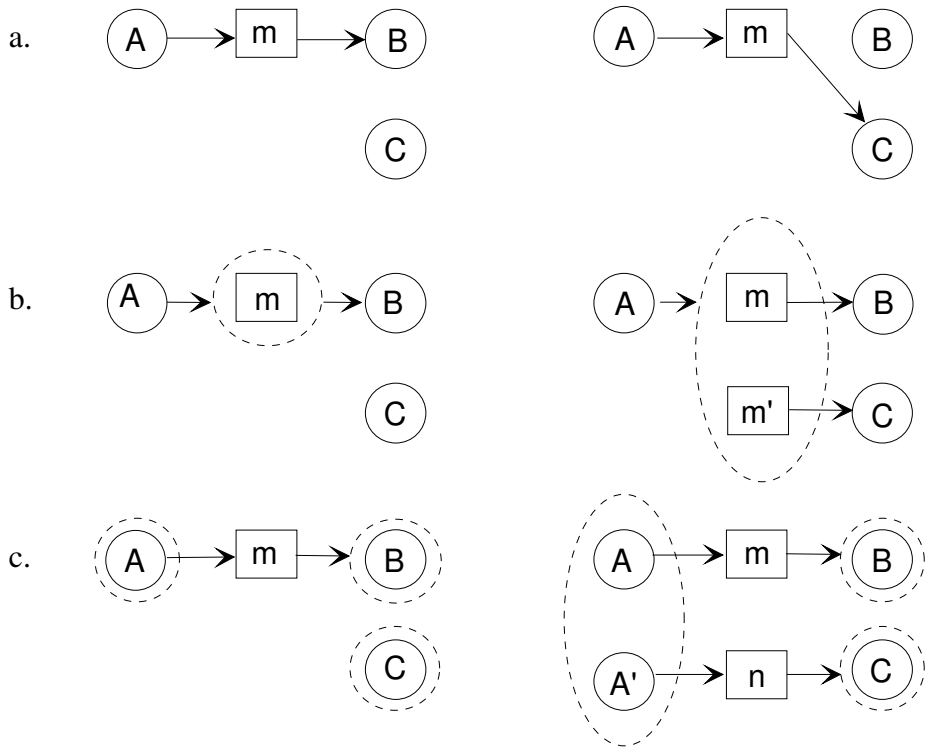

Figure 28: Problems with Versioning Links.

Figure 28.a shows the case where there is no versioning of links. When a link is moved to a new node, the link to the old node is lost.

Figure 28.b shows the case where links are versioned. The new version of the link points to the new node, and the old version to the old node. It is now ambiguous which link should be offered or followed.

Figure 28.c shows the case where we have versioning on nodes only, and new link making is frozen. Thus if, after node A has been frozen, we wish to point the link to C, we must make a new version of the node, and create a new link, n, to do the job.

Of course, it would also be possible to version both *nodes* and *links*, but this introduces further problems of maintaining consistency between the two.

3. *Versions of Structure*

We would like to be able to version the entire network at various times. This is almost analogous to software configurations, but in some hypertext data

models this may be more complex due to the necessity of capturing a consistent set of nodes and links.

4. *Version Creation*

   In his "Introduction and Survey" (1987), Conklin refers to the cognitive overhead that is forced upon users in naming and describing new nodes and links. Any system that employs versioning must have a minimum default behaviour which, from the user's perspective, requires no greater effort than using the system without versioning. Furthermore, when versioning is being explicitly used, the model must be sufficiently simple to encourage users rather than discourage them.

5. *Element Selection*

   In a system that versions nodes, which version of the node should a link point to? HyperPro (Østerbye, 1992) introduces the concept of a *specific version link*, which points to a specific version of the node, and a *generic version link* which computes the version of the node from some selection criteria, which might be as simple as "the most recent version".

   Element selection based on a query language would also be possible, but would force the user to specify the query at the time of each link creation, again increasing cognitive overhead.

10.4. Versioning in Current Hypermedia Systems

Ted Nelson's original vision of Xanadu (Nelson, 1981) always included the need for a versioning mechanism. Xanadu documents are constructed by a list of pointers to streams of bytes which are held on servers, denoting both content and links. Nelson suggests the idea of a "pounce", which is the list of all parts that make up a given document at a given time, and the idea of "alts" which would be alternative versions of a given document.

However, possibly the first realisation of versioning in real hypertext systems was in the concept of "contexts" (Desisle & Schwartz, 1987) used within the Neptune system (Desisle & Schwartz, 1986) which was built upon the HAM

machine (Campbell & Goodman, 1988) and used primarily for CAD applications. The designers of Neptune recognised the need to allow users to experiment with alternative configurations of a hypertext without altering the currently stored version, followed by the need to merge the experimental version back into the original hyperspace. A context is, in effect, a configuration: a set of nodes and links that form a private workspace. Within the private workspace new nodes and links may be created and old ones edited. Links may be made to new versions or old versions of nodes. When contexts are merged conflicts may arise, for example if the node in the original context has been edited at some time since a user copied the node out into a private workspace. The merging algorithm can spot such conflicts, but requires manual intervention to sort them out. The authors were possibly the first to observe the conflict that arises between providing a sufficiently rich versioning model, to cater for all situations, and the cognitive overhead placed upon the user.

Intermedia (Haan et al, 1992) allowed a user controlled form of contexts called "webs". A web was the set of all links and anchors that a user was currently concerned with, and allowed a private view across a set of documents in the same way that is achieved by a Microcosm linkbase. However, Intermedia differed from Microcosm, in that access to all hypermedia links, nodes and anchors could only be achieved through the Intermedia layer, and this layer enforced consistency by ensuring that whenever any document was edited, moved or deleted, that all anchors referring to this document were updated or removed, and all links updated appropriately.

The PIE system (Goldstein & Bobrow, 1987) carried out versioning at two levels in a similar way to the standard software development environment. All hypertext elements were individually versioned (using a linear versioning mechanism) and on top of this, configurations known as "Layers" were introduced. A layer was a co-ordinated set of changes made from some base model. It was thus possible to navigate through the layers to trace the development of the system. It was possible to mix layers from separate projects to produce new projects, but this could lead to possible inconsistencies, for example when trying to merge two projects which both contained the same original base project, but which had been changed in different ways in each of the two projects being merged.

The current generation of Hyperbase systems have tended to build some degree of versioning into their systems. Within this area there appears to be some degree of difference on the question of whether it is necessary to version links as well as nodes, and this difference is often a reflection of the method used for anchor value representation. For example, HyperPro, which uses embedded anchors, does not version links (Østerbye, 1992). Nodes (and composites) may be versioned, but the links do not need versioning as they continue to point to the correct position in all versions of the document. On the other hand HB3, which uses external representations of anchors, does version links and anchors as well as nodes (Hicks et al, 1991). The authors do not address the user interface problems involved, and admit that this approach introduces much scope for inconsistency.

The Hyperbase workshop at Hypertext '93 discussed the importance of versioning in so far as it relates to the Hyperbase data model but, in spite of strong representations from both sides, were unable to agree whether versioning is part of the hypermedia data model or simply a feature of the underlying file storage layer.

One of the most significant pieces of recent work in the area of versioning was the production of CoVer (Haake, 1992), a contextual version server sitting between the SEPIA hypermedia system and the CHS Hypermedia server (Streitz et al, 1992). This system versions both nodes and links, and addresses the issue of the which version of a node a link should point to, and the cognitive overhead, by automatically assigning attributes to objects which designate the task that the user was undertaking at the time that the versions were created. Tasks are arranged in a task hierarchy, and a user who is browsing the network may follow links relevant to a particular task.

The RHYTHM system (Maiolo et al, 1993) is perhaps the only system which has explicitly used versioning as a method to assist in the maintenance of link consistency where using an external anchoring model. A document in this system is a composite made up of inclusions of a number of smaller "atomic" documents. A link joins one inclusion to another. A version of a document is a new list of the inclusions in that document. If the appropriate inclusion is in both versions, then the external reference to this inclusion will remain valid. Also, if the text of the inclusion is edited, links to it will remain valid.

10.5. Versioning in Microcosm

As we have seen from the above discussion, it is possible to devise systems that allow the versioning of only nodes, or the versioning of links only, or both. Furthermore versioning may be at two levels: the level of the individual entity or the level of the entire configuration. This section examines the techniques necessary in Microcosm to version links and nodes, and then presents a versioning scheme for Microcosm which versions both together.

10.5.1. Versioning Links

Currently when a link in a linkbase is updated by any of the link editing tools, what actually happens is that the old link is marked as deleted and the revised link is inserted as an entirely new link: a batch tool is provided which allows the user to garbage collect the deleted links if their number becomes significantly large. However, it would be a simple matter to leave the old link undeleted and only add the new link. Since Microcosm is quite happy to find multiple links at one source and to offer the user a set of destinations, this scheme will immediately work but would introduce the problem for the user of selecting which particular version of a link to follow.

In order to help resolve which link might be followed it is necessary to adopt a simple concept of a context, which in this respect is nothing more than a sub-name for the application. An authoring user is able to select an old context or create a new context. Whenever new links, documents or versions of links are created they are marked with the time of creation and the name of the context in which they were created. A browsing user is then be able to subscribe to zero or many of these contexts, and any links found will be offered in ascending date order, along with the name of the context in which they were created, providing sufficient cognitive hints for the user to make a sensible selection.

The fact that Microcosm works with multiple linkbases, already allows users to create webs in much the same way as is used by Intermedia for creating configurations of links.

The problem with versioning links alone, is that this limits the changes that may be made to those which are represented by structure. It is possible to append

to a node, but not to change the content of a node, without having to edit all relevant link anchors. If one did change the content of a node, for example, using a link aware editor, then the original version of the node would be lost which defeats the intention of keeping versions of the links.

10.5.2. Versioning Documents

The situation with versioning documents (nodes) alone, is much the same as that of versioning links only. New versions of a document can be stored in the DMS with an extra attribute indicating the context in which they were created, but in all other respects containing the same set of attributes. The document dispatcher can be adapted to offer all versions, along with the information stating the context in which each version was created.

The problem with this scheme is that as new versions of a document are created the positions of the anchors, stored in the linkbases, become outdated. Each time links are followed to a different version of the document it will be necessary to apply just-in-time link repairs, and, in the case where a user views an old version of a document, this will quite possibly lead to the loss of links made into more recent parts of a document. It is clearly necessary to keep versions of links that are synchronised with the versions of the documents.

10.5.3. The Microcosm Versioning Scheme

This subsection describes a scheme, and the necessary adaptations to the release version of the system, for versioning in Microcosm, that involves versioning both links and nodes. This scheme has not been implemented, nor is it likely to be implemented in the current version of Microcosm, as it would require significant changes to base code, and there is little user push for such a facility. However, subsequent client-server based versions of Microcosm intended for industrial strength applications may well adopt such a scheme. The purpose of presenting this scheme here is not so much to propose a realistic versioning system for the release version of Microcosm, but rather to demonstrate how such a scheme could assist in maintaining the integrity of the data model.

Currently the Microcosm registry stores a list of applications (projects) of which it is aware. At logon time a user is required to select a particular project: this

information is then used to decide which set of linkbases and other filters to load, and is also used by the DMS to decide which set of documents it will offer to the user at any time. New documents added during this session will be tagged as belonging to the current application. In order to adopt my Microcosm versioning scheme, the registry list of applications must be extended to allow contexts as sub-names for each application, and these contexts must be arranged as a hierarchical tree structure hanging off the base application. When a user logs on they may choose to create a new context at any point in the tree, or select an existing context. Once a new context has been formed below an existing context it will no longer be possible to make document edits to the original context, as shown in figure 29 (although link making at this context level will still be possible).

During the session, the user may edit and delete documents, and add new ones (assuming that they have authoring rights on this application in this context). All edits will only be effective at this context level. If a document is edited it will lead to the production of a new version of this document, marked as valid at this context only, and if the edit results in changes to links, then new links will be created, again marked as valid only in this context.

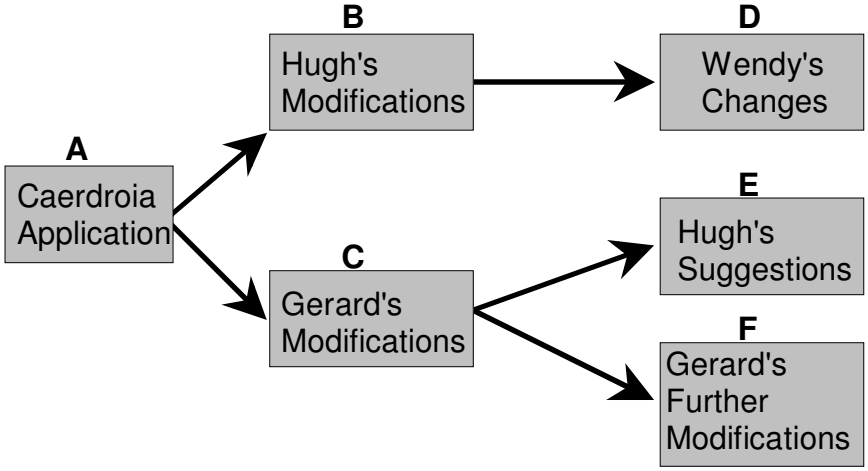

Figure 29: The Tree Structured Context Hierarchy.

The strength of this scheme is that no version of any document or link is ever thrown away. It will still be possible to log into the system and see only the original application, or to log into any other context.

When a user is browsing within some context, the versions of the documents and links offered will always be the most recent up to the level of that context. For example, using figure 29, if a user was logged in to context C, they would be able to see documents and links that had been created in the original application and in C, up to the time that C was frozen, but not those that were made in any other context.

The View Document Dialogue (see figure 8) will need to be altered in order to show only the version of a particular document relevant to the current context, but where a document is a version, this will be indicated, and it will be possible for the user to scan through and visit other versions of the document as well. Similarly, the link dispatcher must be modified to only show the appropriate version of a link, but again where a link has been versioned it will be possible to scan through other versions of the link, and visit the documents that they point to. For simplicity's sake when old versions of a document are visited by such techniques, further link creation within the document will be disabled. Link following will be possible, but only through that set of links that had been defined for that document at the time that the version was created.

The release version of Microcosm contains a tool that allows the export and import of applications from other machines. This needs to be adapted so that it is possible to export a complete context, and then install it as a new base application as desired. This involves filtering out all the old version entries in the DMS and linkbases, and removing the versioning tags before exporting.

A final tool that might be useful would be a merge tool. This could be used to create a new application that was the result of merging two contexts. (Note: it would not be used to create a new context since the context tree is a strict hierarchy: multiple inheritance is not allowed.) This merge algorithm would be non-trivial, since it would be necessary to spot cases where both contexts had made different versions of the same source document. In this case it would be necessary to ask for user intervention to decide what the final version of the document was to consist of, and having done that it would be necessary to resolve all the links from both contexts.

10.5.4. The Advantages of the Microcosm Versioning Scheme

Section 10.1 identified the following motives for versioning.

- Safety. One can undo changes;
- Ability to investigate alternative configurations;
- Facility for intellectual backtracking;
- Exploration of the history of development;
- Maintenance of link integrity.

All of the above advantages are supported by the Microcosm versioning scheme, while putting little extra conceptual overhead on the user other than selecting the context within which they wish to work or browse. However, it is the aspect of maintaining link integrity that is the subject of this thesis and which requires further examination.

First, it is possible to lock the system so that when a user has authoring possession of a particular context no other user may be using this context. This guarantees that no concurrency problems can occur, such as a user making links into a document which is currently being edited, while still allowing users to work in all other contexts as normal. Thus a versioning scheme may help to prevent concurrency problems. This concept is discussed further in section 11.3.

Secondly, it ensures that updates made to the system do not affect other versions of the system, corrupting views with which other users might have been satisfied. For example, a user might have made links into the Caerdroia application in figure 29, and have taken the linkbase away on a floppy disk. These links will remain valid after documents have been edited, since they will point to the original versions of the document rather than the new versions, (which the user will still be able to access by browsing through the versions).

However, there is one problem remaining. When a new version of a document is made, how do the old link anchors get moved into the new version? There are four possibilities.

1. Links don't get moved.

   If the person who edits a document fails to employ any method for copying link anchors from one version of a document into a new version, then all the old links will still point to the version in the previous context. The new version would only service links that had been made specifically within this version. A user would need to browse through earlier versions in order to discover other links made to these versions. This would provide the highest degree of link integrity, in that it would guarantee that all links offered would be to or form the correct place and shown in the correct context that the original author intended, and that none would be lost. However it would impose a high degree of effort upon the browsing user, and it is probable that opportunities to make connections would be missed.

2. Links don't need to be moved.

   Some links do not need to be moved. For example, if the link anchors were embedded within the document, as discussed in section 8.4, then the new version would still contain the necessary anchors. Similarly, edit proof links such as generic links, local links and links destinations that point to the top of the document do not require moving. Nor do links based on search engines (section 8.8). All that is required to deal with such links is to adapt the link following mechanism so that where a document is named as the source or destination of a link, the engine will automatically follow links to the version furthest down the context tree to the current context, and will follow the links from *any* version of the same document.

   This solution makes life very easy for the authoring user, but does open some opportunities for inconsistency, such as the case where an embedded link anchor has been removed as part of an edit. In this case the user will not be aware of the anchor, and will probably not look in earlier versions of the document, even though it will still be present in these versions.

3. It is the editor's responsibility to see that link anchors are moved.

   In chapter 8, a number of techniques were discussed for editing link anchors with specific offsets. It is the authoring user's responsibility to ensure that one of these methods is employed. The most likely method is that a link aware

editor will be used at the time that a document is edited, and this link aware editor will save the new versions of the links rather than replacing the old versions.

4. Modified "just-in-time link repairs".

   The current implementation of "just in time link repairs" updates out of date links pointing into the current document, at the time that the document is loaded. This algorithm may be altered to spot when there is a more recent version of the document within the scope of the current context and to make new versions of the appropriate links. This could be done using context as described in section 8.5, but given that, in this scenario, we have both versions of the document available to us it would be more appropriate to update the links using the *diff* of the two files, as described in section 8.11.

This entire discussion on maintaining link integrity by versioning assumes that editing of documents is only carried out in a Microcosm friendly way. If a version of a document is edited off line, rather than correctly versioned, then the scheme will fail. In this scenario this does not seem an unreasonable restriction, in view of the fact that in any application where one was employing a versioning system, such as RCS or SCCS, one would only be able to access the documents through the official interface, and in this environment, creating a new version of a document without moving the links into the new document would be an act of deliberate vandalism.

Chapter 11. Concurrency Problems

Most Microcosm users currently either run the system as a single user, or else by mounting a set of read-only shared network resources which they supplement by adding private links and resources on their local machine. In these cases the concurrency problem does not occur. However, if two or more users are attempting to update the same document, DMS file or linkbase file at the same time, then problems may occur. The standard problem is the *lost update* which occurs when two users both load the same database record and alter it, then user *A* saves, then user *B*. User *A*'s update will be overwritten by user *B*. The problem with the Microcosm DMS and linkbase files is currently even worse than this as, at present, each user caches all the files locally, to increase speed of response. Changes are written back to the central database at the time that the user quits, but there is no notification mechanism to inform other users that there is a need to update their local copies, which again can lead to lost updates.

Clearly the current design of Microcosm is flawed with respect to concurrency control, and this problem can open many opportunities for loss of data (when new links are overwritten by another user's updates) or for loss of data integrity (when updated links are lost again by another user's updates). This chapter examines the science of concurrency control in hypermedia systems and examines ways that Microcosm could be improved to solve this problem.

The standard method of dealing with concurrency control is to introduce some form of *locking*, whereby a transaction locks the records that it is about to alter, so that other transactions are unable to use it during this interval. For a treatment of locking mechanisms the reader is referred to any standard text on database systems e.g. Elmasri & Navarthe (1994). The fundamental difference with hypermedia systems is that transactions are typically long (Grønbæk et al, 1993), so if a user is working on a document it will prevent other users accessing that document for some considerable period. It is tempting to suggest that the document might be viewed by other users in a read-only mode, but in that case link making may also need to be disabled, otherwise link anchors might become invalid when the user editing the document saves.

In order to introduce locking into hyperbase systems it is necessary to identify the smallest possible component that can be locked. In practice this means making composite documents, where each subsection of the document is a separately addressable (and therefore lockable) object, and each attribute of an object is separately lockable.

Hypermedia systems intended specifically for collaborative computing, such as EHTS (Wiil & Leggett, 1993) which is based on Aalborg University's "Hyperbase" (Wiil & Østerbye, 1990) and DHM (Grønbæk et al, 1993) provide specific support for fine grained locking mechanisms and event notification. Wiil and Leggett (1993) identify the following concurrency control requirements for hypermedia systems:

1. *Event notification*, so users may be made aware of changes that other users have made to objects in the system.

2. *Fine grained notification*, so that it is possible to distinguish, for example between a write operation on a whole object and some attribute of that object.

3. *User controlled locking*, in order to provide support for long transactions, such as editing a large object.

4. *Shared locking*, so that for example, one user may at least read an object that is currently being edited by another.

5. *Fine grained locking*, so that every attribute of an object, as well as the object itself, may be separately locked.

6. *Persistent collaboration information*, so that in the event of a server crash it will be possible to restore the server process to the state at the time of the crash.

The current implementation of Microcosm does not lend itself to such solutions as Microcosm is *not* a client-server architecture, and was not intended for such fine-grained collaborative working. When users are working on a shared resource base they download the DMS database and the communal linkbase data from the server at the time that they begin a particular application session, and it is only at the end of the entire session that these database files are written back to the file server, if they have changed. This introduces enormous opportunities for lost updates to occur, since a session may be long, and many other users may have updated the

communal database during the session, all of which will be lost when the original user writes back.

Before investigating solutions to the problem in Microcosm it is worth understanding why Microcosm has been designed as it has. Microcosm was never intended to be a client-server architecture. The current generation of PC's on which Microcosm runs will not operate as process servers to external clients. Instead we designed Microcosm as a "personal viewport on an external universe of information": Microcosm would be able to access any resource on any machine (of any architecture) so long as it could arrange to copy whatever files were needed from that machine. Therefore Microcosm must be able to operate where the only access to shared resources is via a file server. This requirement does not preclude Microcosm from working in a client-server mode, but precludes this from being the only scenario. Work on accessing remote linkbases and DMS's using peer to peer networking is currently a research interest within the laboratory (Hill & Hall, 1994) (Hill, 1994) (Melly, 1994).

A second reason why Microcosm was not written as a client-server architecture is remote database access speed. In the early stages of the design of ZOG, Akscyn observed (McCracken & Akscyn, 1984) that speed of access to nodes in a hypertext environment was an essential requirement if users were not to become discouraged from enquiring about links. In order to ensure that the fairly complex queries that Microcosm makes of its linkbases and DMS were carried out at sufficient speed, especially when one considers the number of messages that would need to pass to the server and back, we felt that it was necessary to cache all the linkbase and DMS indexes and data locally, even if they were originally fetched from a fileserver. (In recent years systems have become much faster: also thanks to the WWW, users' expectations of the speed of response have actually dropped, so it might now be feasible to implement a client-server architecture).

Records in a Microcosm linkbase and DMS are not fixed length, neither are fields. In the prototype versions of Microcosm we used fixed width records and fields, and we used a commercial database to implement the linkbases. However it soon became apparent that users would wish to add their own fields (e.g. user defined document attributes) and that it was not possible to fix the width of a field, since, for example, the size of a selection might be anything from a couple of characters, to a large paragraph. For these reasons we re-implemented the

linkbases and DMS, initially using a commercially available C-Tree package, but this introduced further limitations, such as the number of fields that could be indexed, so we re-implemented again using our own in house B-Tree written by Ian Heath.

This version of the database keeps each ASCII record as a line in a text file known as the "RAW" file, and maintains indexes, based on every field (tag) that has been introduced, into the records of the RAW file. A substantial API allows programmers to write code which queries databases based on this code. At the time that the code is loaded it will check to see that the indexes still correspond to the RAW file, and if not will recreate the indexes. Building new indexes takes considerably longer than copying indexes from a fileserver, so generally the full set of indexes are maintained on disk, with the RAW file, and copied to the user machine at start-up. The current system has no concept of individual record locking.

Given the above information, it is now possible to investigate solutions to the concurrency problems that the current version of Microcosm introduces when using shared documents, linkbases and DMS on a fileserver architecture.

11.1. Read-Only Shared Resources.

The simplest solution to the concurrency problem is to prevent it from ever happening. All shared resources (documents, linkbases and DMS) should be made read-only. This appears to be a highly restrictive solution, but in many applications, such as the delivery of teaching materials, this solution is perfectly adequate. Link making will still be possible, so long as any links created are only saved in a private linkbase. This is anyway the usual Microcosm configuration: changes to the shared resources are carried out at a time when no other user is concurrently accessing the resources.

Unfortunately, the current version of Microcosm does not support the idea of a private DMS, so it is not possible for users in such an environment to add private documents to the system. The modification of the Microcosm system to allow separate chained DMS's (in much the same way that linkbases are chained) would

solve this problem, but is currently not implemented: it is on the agenda for the next version.

11.2. Crude Locking and Update Notification.

As explained in the previous section, in many situations, users will not wish to modify the shared resources. However, in a system in which a *small number* of changes to a shared resource are envisaged at a time when other users are accessing the system, then it would be possible to adopt a crude user controlled locking and notification scheme as follows.

Whenever a user wishes to make changes to a shared linkbase or the DMS they must first inform the system of their intention, which will then:

a. Check that the file date on the shared version of the database is the same as the file date on the locally cached version. If not, they must first re-load the database.

b. Check the lock on the relevant database. If it is locked, then someone else is editing it, so we must wait till they have finished.

c. If it is not locked, then the system must now lock it.

d. Make the required edits.

e. Write the required databases back to the server.

f. Unlock the database.

In order to allow editing of a shared document the problem is more complex, since it is possible that links will also be changed so the system must:

a. Check that the file date on all shared linkbases is the same as the file date on the locally cached versions. Reload any out of date linkbases.

b. Check that all locks on the linkbases and on the document to be edited are available.

c. Lock all linkbases and the document..

d. Edit the document using a link aware editor, or mend the links after the edit.

e. Write the document and linkbases back to the server.

f. Unlock the document and linkbases.

g. Set a flag stating the date and time at which *any* shared document was last edited.

Locks must be kept in a shared file (which itself needs setting as read only by any application while changing it!) which contains a list of linkbases that are locked, the DMS lock, and the *date of last document edit* flag.

From the point of view of any user simply browsing the system, the locally cached DMS and linkbases will remain consistent (if not entirely up to date) so long as no document has been edited, in which case it is possible that the locally cached links will no longer be valid. If each shared file accessed is preceded by a check on the *date of last document edit* flag, then it is possible to see whether edits have occurred since the local linkbases were loaded. In this case it will be necessary to reload the relevant shared linkbases. Of course, it is possible to check and reload linkbases and DMS at any time, but constant checking and reloading could become a nuisance.

The above system enables users to make edits to a Microcosm application at the same time as other users are accessing it, but has two shortcomings.

1. It does not ensure that private linkbases are updated at the time that shared linkbases are updated. This can only be solved by using one of the link repair methods detailed in chapter 8.

2. If many users are making many changes, particularly document edits, the number of occasions on which a user needs to reload the shared linkbases and DMS might become so frequent as to become a serious hindrance to use of the system. In an application area where such frequent changes are envisaged this approach would be inappropriate.

11.3. Microcosm Version Contexts.

Subsection 10.5.3 introduced a scheme for applying versioning contexts within Microcosm, and pointed to the fact that such a scheme is also useful in helping to prevent concurrency problems, since links and documents are never edited, but rather, new ones are created. In this scenario, only one user is allowed editing rights on a particular context at one time: other users are allowed to browse within the context, but may not change anything that belongs to this context. Link making is only be allowed into a private local linkbase, and these links cannot be guaranteed to remain consistent, as items within the context might be edited after they were made. Workers within other contexts are able to continue to work as normal.

Under these conditions the only concurrency problem that remains is that, when the DMS and shared linkbases are written back to the server at the end of a session, it is possible that they could overwrite changes made during the editing period by other users. However, in this case, it is possible to produce a simple database update algorithm. This is possible because, when using versioned contexts, nothing in any other context is changed by the current context, and only one user may be editing the current context. The update algorithm then becomes a case of removing all records marked as belonging to the current context from the shared database, and adding all records from the new version of the current context.

This method causes minimum disruption for other users, so long as the restriction of only one authoring user per context is acceptable.

11.4. Client Server Architecture for Databases.

If none of the above solutions to the concurrency problems are acceptable in a particular application, then presumably the application is one that requires a true co-operative working environment. The simplest way to achieve this would be to rewrite those parts of the architecture that handle database records and documents as a client-server hyperbase.

If this were done, then individual nodes, links and DMS records could be write-locked. Furthermore, within such an architecture, notification control and link integrity maintenance would be far simpler to implement. However, the advantages of the simple customisation of the single user environment and the possibilities opened by a truly distributed architecture would be lost .

Mylene Melly has been working on extending Microcosm for co-operative working (Melly, 1994). In order to achieve this she has re-implemented the linkbases using the Exodus object oriented database, and has done away with the DMS altogether. She has retained the peer to peer communications of the system. The result is that a filter manager can arrange to route link operations through a linkbase running on a remote machine, and individual records within each linkbase may be locked. The resulting system solves the problems of concurrency, but the introduction of distributed linkbases and the absence of a DMS further aggravates the problems of maintaining link integrity when editing documents.

In order to produce a truly "industrial strength" version of Microcosm for large corporate teams working in a co-operative manner, the team are currently evaluating the possibility of implementing the linkbases and DMS using a commercial database with a client server architecture. It may well turn out that rather than allowing separate linkbases and DMS's the best solution is to use only one database to hold all links and DMS entries. Records within these databases could be separated into logical linkbases and DMS's by some tag, and the database might be distributed in order to optimise for speed at any particular site. If the team opt to take this route, we will have to weigh up very carefully the advantages of increased consistency and the ease of concurrency control, when balanced against the loss of openness and simplicity that is characterised by the current model.

Chapter 12. Conclusions

12.1. The State of Hypermedia

Ever since the introduction of commercial hypermedia systems in the mid 1980's, conference keynote speakers have urged forward the research community, predicting that the following year would be the year in which hypermedia would finally become a commercial success and would be accepted by the user community. However, by the early 1990's the euphoria began to diminish and a change in tone became noticeable. Frank Halasz (1991) revisited his "seven issues for the next generation of hypertext" and attempted to re-focus the research community onto what many believed to be the important issues, but perhaps the best explanation of the problems facing the community was provided by Ian Ritchie (1992) in an attempt to instil some commercial realities.

Ritchie explained that when a new technology is introduced, there is an initial phase when it is enthusiastically embraced by the "early adopters". These are the technophiles who take a technology and pro-actively seek a problem which can use this technology as a solution. During this phase, the uptake of the technology will be fast, but there is a limit to the growth defined by the number of such early adopters. However, the much larger group of users will be unimpressed at this stage: these are not people looking for new technologies, but people who only look for a solution to a problem when they *have* the problem, but are unable to solve it with any technology with which they are currently familiar. Only at this stage will these users begin to evaluate new technologies.

In the late 1980's we saw the effect of the early adoption of hypermedia. By the early 1990's the community was becoming dispirited by the failure of the technology to continue to grow at the predicted rate. Now in the mid 1990's we see the real users making considerable use of the technology in the form of the World Wide Web. The importance of this system cannot be under-estimated, and its success has dumbfounded the pundits who forecasted that the way forward was ever more sophisticated systems. It turned out that what the majority of users really wanted more than anything else was seamless access to information on the

Internet! Magazines now report that Tim Berners-Lee invented hypertext, entirely ignoring the past history of research and development.

However, although the Web is a very open system in some respects, there are application areas for which it is unsuitable in its current form: co-operative working environments, corporate information systems, technical documentation, CD-ROM publishing, and personal information environments to name but a few.

Perhaps the problem for hypermedia has been the breadth of application areas in which it has attempted to provide solutions. Currently we see the acceptance of hypermedia in areas such as technical documentation (using systems like Guide, Hypercard and Microcosm) and the delivery of educational and other information resources (often using the Web). However, hypermedia's promise to provide new environments for the organisation of personal information, co-operative working or corporate information systems has not yet been realised.

There are two issues to consider here. First, the user community is reluctant to commit to a technology in which there are no accepted standards. Of course, standards do exist, such as HyTime and HyperODA, but commercial realisations of these standards have yet to make any dent in the market place, and as yet no large software house has produced a system which might become a *de facto* standard. Secondly, the commercial systems that do exist do not yet provide either the security or the functionality that is required of industrial strength hypermedia systems (Malcolm et al, 1991).

Probably the key technological issue for hypermedia now is integration. Users will only fully accept the technology when it is *part* of their working environment. They do not wish to be compelled to leave their current working environment and enter a new environment in order to be granted hypermedia functionality: this way, all they can do is provide static information environments, which are of course those application areas which have been most successful so far, for exactly this reason. The next generation of hypermedia systems must be integrated with the users' personal working environments so that they can provide and access multiple methods for browsing and locating information with the minimum disruption to their working patterns. This thesis has examined the background to the current state of hypermedia, and reported on those systems which are on the leading edge.

12.2. Microcosm

The Microcosm project has been one of the first to attempt to address the issues of industrial strength hypermedia, by providing a link service that is entirely separated from the applications that access that link service, and which allows the applications to communicate with the link service at a number of different levels. The system, being modular in design, is highly flexible and can be tailored and extended in a number of ways in order to represent a number of different hypermedia models and to provide a range of functionality, particularly a rich array of methods for browsing and locating information.

As a research tool, Microcosm has been used in a range of application areas, some of which have been described in this report, and it maybe that it is this flexibility that is both the strength and weakness of the system. It is very difficult to explain exactly what Microcosm is, just because it is so flexible that most projects use quite different configurations of the system, and use it for such different applications. This thesis has presented the design and functionality of the base system, the commercially released product known as Version 3, while at the same time introducing a number of extensions, such as the Universal Viewer, which are available within the research laboratory.

12.3. Integrity Issues

Any system which entirely separates structural information from content provides the potential for inconsistencies to occur, such as the editing problem (subsection 7.2.1) and the dangling link problem (subsection 7.2.2). These problems have been fully explained, and a number of solutions to each problem have been proposed and examined, with particular reference to their application within Microcosm.

The conclusion of these studies is that there is no one solution which guarantees to ensure integrity of the hypermedia structure under all conditions when working with a system that is as open as Microcosm; it is necessary to impose some conditions on the use of the system if integrity is to be assured. Conditions may range from simple restrictions as to the use of the system, such as forbidding the editing of documents except by the use of a link aware editor, through to providing variants of the system which are configured in such a way

that these problems do not occur. For example, the system might require that all edits cause new versions of the document, or might be implemented such that all DMS and link requests are serviced by a single database management system. Such a DBMS could control the integrity and handle the concurrency problems, while providing all the security features required by a commercial organisation.

However, all of the conditions have the effect of restricting the full functionality of Microcosm: they are in effect *closing* the system in some respects. As the number of organisations that approach us to discuss using Microcosm for their information needs increases, we have begun to learn that, to some extent, this will be necessary, but need not have the appearance of restricting the user. To illustrate this point I will consider the major application areas.

Information and educational resource delivery, and technical documentation have thus far been our major application areas. In these areas the opportunity for failures in link integrity are small: the resources are generally published in a read-only format, and will not themselves be changed by the user. Link creation and editing are possible in the knowledge that the content will not be changing. The authors of the information may wish to produce new versions of the information, but they can arrange to supply the data in some form, such as RTF, for which link aware editing is provided.

Where Microcosm is being used to provide a personalised information environment, the situation is quite different: the information will be frequently changing and the file store will be occasionally reorganised. However, in this case the quality of information environment required is generally quite different. Link source anchors may be generic or local, so are not subject to the editing problem, and destination anchors are generally whole files, or can be expressed in terms of queries rather than byte offsets, again avoiding the editing problem. Also, in such an environment one can reasonably expect the user to take responsibility for ensuring that the DMS entries are maintained. The problem here is more one of providing a light hypermedia link service with the minimum disruption to the working environment, rather than one of ensuring link integrity.

It is in the area of co-operative commercial information environments that the link integrity problems are most severe. In such an environment users will wish to share data, co-operatively edit data and publish data, while still allowing other users to link into and out of the information. In a large scale environment, systems

which rely upon techniques such as just-in-time link repairs (section 8.5) may well cause such a heavy load on the system, as it is currently implemented, that they would not be practical. For such an environment it will be necessary to resort to a client server technology to serve the documents and the links, and to hold these links in such a way that the system can control the integrity, either by only allowing link aware editing, or by using a versioning system of the kind described in subsection 10.5.4.

12.4. The Future

In section 12.1 I examined the current state of hypermedia. There are application areas in which hypermedia has been successful, notably all aspects of information publishing, such as electronic journals, technical documentation and the delivery of educational material. There are those that advocate that, with the advent of the Web, hypermedia has passed through the stage of being an active research topic, and is now no more than a development problem. However, I disagree with this view, and in this final section will summarise some of the important issues that must still be addressed.

There are many application areas in which hypermedia has still to make its mark. People use computers, mainly, to store and retrieve information. Hypermedia has the potential to provide the richest set of tools for providing the functions necessary to store and retrieve heterogeneous information. One might envisage the hyperbase as the "killer application" of the future, in much the same way that relational database has been for the previous decade.

Hyperbases for corporate information management, or even personal desktop management have not yet arrived. The designers of Microcosm always intended that the system would be used in such areas. The problem now is to build the industrial strength version of the product, that will provide all the qualities associated with commercial database systems, such as persistence, versioning, wide area network accessibility, multiple users and concurrency control, along with a range of tools for maintaining, manipulating, tailoring and extending the system.

Perhaps one of the most exciting recent developments in hypermedia has been the realisation that hypermedia is about more than simply clicking on buttons.

Hypermedia can provide an interface through which a user may ask the system to find out what it knows about any object within view, and with the advent of agent technology the system can reach out to the whole of the Internet in search of this information. Microcosm was designed from the beginning as a system that allowed the user to select objects and ask questions about the object, and the filter chain makes it possible to dynamically install any process, local or remote, that might help to service a request. The subject of the use of agents to support the information environment will, alone, be a topic for research for the foreseeable future.

The future of the Microcosm system, or indeed any other link service, depends upon making the link service available to all the tools and applications that users expect in their working environment, with a minimal disruption to their working practices. If users are to be encouraged to invest the effort to make trails and links available to other users, the integrity of the systems of the future must not become corrupted every time that a document is moved or edited. This thesis has examined in this issue in detail, and suggested a number of approaches to ensuring that such integrity can be maintained.

Bibliography

Akscyn, R., McCracken, D.L. & Yoder, E. *KMS: A Distributed Hypertext for Sharing Knowledge in Organizations*. Communications of the ACM. Vol 31(7). pp 820-835. July 1988.

Anderson, K.M., Whitehead, E.J. Jr. & Taylor, R.N. *Chimera: Hypertext for Heterogeneous Software Environments*. In: The Proceedings of the ACM European Conference on Hypermedia Technology, ECHT '94. Edinburgh. pp 94-107. ACM Press. 1994

Andrews K., Kappe, F. & Maurer, H. *Hyper-G: Towards the Next Generation of Network Information Technology.* Journal of Universal Computer Science, April 1995.

Apple Computer Inc. *Inside Macintosh: Inter-application Communication.* Addison Wesley. 1994

Apple Computer Inc. *OpenDoc Technical Summary. 1993*

Aßfalg, R. (ed.). *The Proceedings of the Workshop on Open Hypertext Systems, Konstanz*, May 1994.

Beitner, N.D.. & Hall, W. *Putting the media into hypermedia.* In: The Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking 1995, Vol. 2417. 1995.

Berners-Lee, T.J., Cailliau, R. & Groff, J-F. *The World Wide Web.* Computer Networks and ISDN Systems 24(4-5), pp 454-459. 1992

Berners-Lee, T. *Hypertext Transfer Protocol (HTTP)*, available by ftp as http-spec.ps from info.cern.ch in directory /pub/www/doc. CERN, 1993

Bernstein, M. *An Apprentice that Discovers Hypertext Links.* In: A. Rizk, N. Streitz and J. Andre eds. *Hypertext: Concepts, Systems and Applications. The Proceedings of The European Conference on Hypertext*, INRIA, France. pp 212-223, Cambridge University Press. 1990

Bernstein, M. *Enactment in Information Farming*. Technical Briefing in: The Fifth ACM Conference on Hypertext. ACM. 1993

Bigelow, J. *Hypertext and CASE*. IEEE Software 5(2). Mar. 1988

Brown, P.J. *Turning Ideas into Products: the Guide System*. In: Proceedings of the ACM Hypertext '87 Workshop. The University of North Carolina at Chapel Hill. pp 33-40. Nov. 1987.

Buchanan, M.C & Zellweger, P.T. *Specifying Temporal Behaviour in Hypermedia Documents.* In: D. Lucarella, J. Nanard, M. Nanard, P. Paolini. eds. *The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 262-269. ACM Press, 1992.

Bush, V. *As We May Think*. Atlantic Monthly, pp 101 - 108, July 1945.

Campbell, M & Goodman, J. *HAM: A General-Purpose Hypertext Abstract Machine*. Comm. ACM 31(7). pp 856-861. July 1988.

Carr, L.A., Barron, D.W., Davis, H.C. & Hall, W. *Why Use HyTime?*. EP-ODD, Vol. 7 No 1. 1994a

Carr, L.A., Davis, H.C. & Hall, W. *Using HyTime Architectural Forms for Hypertext Interchange.* Information Services & Use. 13(2). 1993

Carr, L.A, Hollom, R.J., Hall, W. & Davis, H.C. *The Microcosm Link Service and its Application to the World Wide Web.* In: Cailliau, R., Nierstrasz, O. & Ruggier, M. (eds.). *The Proceedings of the First World-Wide Web Conference.* pp 25-34. CERN. May 1994b.

Cawley, Ashman, Chase, Dalwood, Davis & Verbyla. *A Link Server For Integrating the Web with Third Party Applications* In: The Proceedings of AusWeb95, http://www.scu.edu.au/ausweb95/papers/integrating/cawley/

Conklin, J. *Hypertext: An Introduction and Survey.* IEEE Computer 20, 9, pp 17 - 41. 1987.

Conklin, J & Begeman, M.L. *gIBIS: A Hypertext Tool for Exploatory Policy Discussion.* ACM TOIS. pp 140-152, Oct 1988.

Davis, H.C., Hall, W., Heath, I., Hill, G. & Wilkins, R. *Towards an Integrated Information Environment with Open Hypermedia Systems. In:* D. Lucarella, J. Nanard, M. Nanard, P. Paolini. *eds. The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 181-190. ACM Press, 1992.

Davis, H.C., Hall, W. & and Heath, I. *Media Integration Issues within Open Hypermedia Systems.* In: Damper, R.I., Hall, W. & Richards, J.W. (eds.) *Multimedia Technologies and Future Applications.* Pentech Press, 1994a.

Davis, H.C., Knight, S.K. & Hall, W. *Light Hypermedia Link Services: A Study in Third Party Application Integration.* In: The ACM Conference on Hypermedia Technology, ECHT '94 Proceedings. pp 41-50. ACM. Sept. 1994b.

Davis, H.C. & Heath, I. *An Evaluation of Actor for Developing Windows Applications.* Technical Report, CSTR 89-5. The University of Southampton. 1989.

Davis, H.C. & Hey, J.M.R. *Automatic Extraction of Hypermedia Bundles from the Digital Library*. In: *Shipman, F.M. III, Furuta, R.,& Levy, D.M.* The Proceedings of Digital Libraries '95. Texas A&M University, June 1995

Davis, H.C., Hutchings, G.A. & Hall. W. *A Framework for Delivering Large-Scale Hypermedia Learning Material.* In: Hermann Maurer. ed. *Educational Multimedia and Hypermedia Annual 1993, Proceedings of ED-MEDIA '93,* Orlando, Florida, USA, AACE. 1993.

Davis, H.C. *Using Microcosm to access Digital Libraries*. In: Schnase, J.L., Leggett, J.J., Furuta, R.K. & Metcalfe, T. (eds.). *The Proceedings of Digital Libraries '94.* Texas A&M University, June 1994.

Davis, H.C. *To Embed or Not to Embed...,* Communications of the ACM, Vol 38(8), pp 108-109. August 1995.

DeRose, S.J & Durand, D.G. *Making Hypermedia Work: A User's Guide to HyTime.* Kluwer Academic Press. 1994

Desisle, N & Schwartz, M. *Neptune: A Hypertext System for CAD Applications.* In: The Proceedings of ACM SIGMOD '86. Washington D.C. pp 132-142. ACM Press. May 1986.

Desisle, N.M. & Schwartz, M.D. *Contexts - A Partitioning Concept for Hypertext.* ACM Transactions on Office Information Systems, pp 168-186, Vol. 5(2) April 1987

Duck, S. Hall, W., Pickering, C. & Riley, M. *A Project Management Based Decision Support System for Use in the Construction Industry*. In: Powell, J.A. (ed.), *Proceedings of the Conference on Informing Technologies to Support Engineering Decision Making*. pp 143-154. EPSRC. Nov. 1994.

Durham, T. *The Gathering of an Information Harvest*. Computing. 27th April 1989.

Elmasri, R. & Navathe, S.B. *Fundamentals of Database Systems (Second Edition)*. Benjamin Cummings. 1994

Engelbart, D.C. *A Conceptual Framework for the Augmentation of Man's Intellect* In Vistas of Information Handling, Vol. 1, Spartan Books, London. 1963.

Flohr, U. *Hyper-G Organizes the Web*. Byte 20(11). pp 59-64. November 1995.

Fountain, A.M., Hall, W., Heath, I. & Davis, H.C.. *MICROCOSM: An Open Model for Hypermedia With Dynamic Linking*, In: A. Rizk, N. Streitz and J. Andre eds. *Hypertext: Concepts, Systems and Applications. The Proceedings of The European Conference on Hypertext, INRIA, France.* Cambridge University Press. 1990

Fox, E.A., Akscyn, R.M., Furuta, R.K. & Leggett, J.L. *Digital Libraries: Introduction.* Communications of the ACM, Vol. 38(4), pp 22-29, April 1995.

Gladney, H.M., Ahmed, Z., Ashany, R., Belkin, N.J., Fox, E.A. & Zemankova, M. *Digital Library: Gross Structure and Requirements.* IBM Research Report RJ 9840. IBM, May 1994.

Goldstein, I. & Bobrow, D. *A Layered Approach to Software Design.* In: Barstow, D., Shrobe, H. & Sandwell, E. (Eds.). *Interactive Programming Environments* , pp 387-413, McGraw Hill, 1987.

Goodman, D. *The Complete Hypercard Handbook*. Bantam Books, New York. 1987.

Goose, S. & Hall, W. *The Development of a Sound Viewer for an Open Hypermedia System*. CSTR 94-03.. The University of Southampton, U.K. 1994.

Greif, I. *Hypertext and Group-enabling: Lessons from the Desktop*. Key-note address given at Hypertext '93. Seattle, Washington. November 1993.

Grønbæk, K. & Trigg, R.H. *Design Issues for a Dexter-Based Hypermedia System. In:* D. Lucarella, J. Nanard, M. Nanard, P. Paolini. *eds. The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 191-200. ACM Press. Nov. 1992

Grønbæk, K. & Trigg, R.H. *From Embedded References to Link Objects: Toward a New Data Model for Open Hypermedia Systems*. To be published in The Proceedings of Hypertext '96. ACM 1996

Grønbæk, K., Hem, J.A., Madsen, O.L. & Sloth, L. *Designing Dexter-Based Co-operative Hypermedia Systems.* In: Proceedings of Hypertext '93, Seattle, Washington, November 1993. ACM. 1993

Grosky, W. & Mehrota, R. (eds.). *IEEE Computer, Special Issue on Image Database Management*. Dec. 1989

Haake, A. *CoVer: A Contextual Version Server for Hypertext Applications*. In: D. Lucarella, J. Nanard, M. Nanard, P. Paolini. eds. *The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 43-52. ACM Press, 1992.

Haan, B.J., Kahn, P., Riley, V.A., Coombs, J.H. and Meyrowitz, N.K. *IRIS Hypermedia Services.* The Communications of the ACM. 35(1) Jan. 1992.

Heath, I. *An Open Model for Hypermedia: Abstracting Links from Documents*. PhD Thesis. The University of Southampton. Oct. 1991.

Halasz, Frank G.", *Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia systems*, Communications of the ACM, 31(7), pp 836-855, 1988

Halasz, F. & Schwartz, M. *The Dexter Hypertext Reference Model*. In: The Proceedings of the Hypertext Standardization Workshop. pp 95-133, Gaithersburg. US Government Printing Office. Jan. 1990.

Halasz, F. & Mayer, S. (edited by Grønbæk, K. & Trigg, R.H). *The Dexter Hypertext Reference Model.* Communications of the ACM. pp 30-39. 37(2). Feb. 1994.

Halasz, F. *Seven Issues Revisited.* Keynote speech to Hypertext '91, San Antonio, 1991

Hall, W. *Ending the Tyranny of the Button*. IEEE Multimedia 1(1). 1994.

Hall, W & Davis, H.C. *Hypermedia Link Services and Their Application to Multimedia Information Management.* J. of Information and Software Technology, pp 197-202, 36(4). 1994.

Hall, W., Hill, G.J. & Davis, H.C. *The Microcosm Link Service*. In The Proceedings of Hypertext '93: the Fifth ACM Conference on Hypertext. Seattle, Washington, November 1993. pp 256-259. ACM Press. 1993a

Hall, W, Lewis, P.H. & Davis, H.C. *Enhanced Handling of Images and Digital Video Sequences in Multimedia Information Systems.* SERC Project GR/J53614, 1993b

Hammwöhner, R & Rittberger, M. *KHS - Ein offenes Hypertext-System.* Technical Report 28-93 (WITH-3/93), Department of Information Science, University of Constance. 1993.

Hardman, L., Bulterman, D.C.A. & van Rossum, G. *Adding Time and Context to the Dexter Model*. Communications of the ACM. pp 50-63. 37(2). Feb. 1994.

Hicks, D.L, Leggett, J.J, & Schnase, J.L. *Version Control in Hypermedia Databases*. Technical Report TAMU-HRL 91-004. Hypertext Research Laboratory, Texas A&M University. July 1991.

Hill, G.J, Wilkins, R.J & Hall, W. *Open and Reconfigurable Hypermedia Systems: A Filter Based Model.* Hypermedia 5(2). 1993

Hill, G.J. & Hall, W. *Extending the Microcosm Model to a Distributed Environment.* In: The ACM Conference on Hypermedia Technology, ECHT '94 Proceedings. ACM. Sept. 1994.

Hill, G.J. *Extending an Open Hypermedia System to a Distributed Environment.* PhD Thesis. The University of Southampton. April 1994

HTML. *HTML overview*, http://www.ucc.ie/info/net/htmldoc.html, 1994

Hutchings, G.A. *Patterns of Interaction with a Hypermedia System: A Study of Authors and Users*. PhD Thesis. The University of Southampton. June 1993.

Hutchings, G.A., Carr, L. & Hall, W. *StackMaker: an Environment for Creating Hypermedia Learning Material.* Hypermedia, 4(3). pp 197 - 212. 1993a.

Hutchings, G.A., Hall, W & Colbourn, C.J. *Patterns of Students' Interactions with a Hypermedia System*. Interacting With Computers. 5(2). 1993b

HyperODA, *HyperODA - a Working Draft for Extending ODA Standards to Support Hypermedia Applications*. ISO/IEC JTC1/SC18/WG3 N1898, 1992

HyTime, *Hypermedia/Time-based Structuring Language*, ISO/IEC 10744:1992, *1992*

Julienne, A. & Russell, L. *Why You Need ToolTalk*. SunExpert Magazine pp 50-59. Mar. 1993.

Kacmar, C.J. & Leggett, J.J. *PROXHY: A Process-Oriented Extensible Hypertext Architecture.* ACM Trans. on Information Systems, 9(4) pp 299-419. Oct. 1991.

Kacmar, C. J. *A Process Approach for Providing Hypermedia Services to Existing, Non-Hypermedia Applications.* To be published in EPODD, 1995.

Lawton, D.T. & Smith, I.E. T*he Knowledge Weasel Hypermedia Annotation System.* In The Proceedings of Hypertext '93: the Fifth ACM Conference on Hypertext. Seattle, Washington, November 1993. pp 106-117. ACM Press. 1993

Leggett, J. & Schnase, J. *Dexter with Open Eyes.* Communications of the ACM 37(2) pp 77-86. Feb. 1994

Leggett, J.L., Schnase, J.L., Smith, J.B. & Fox, E.A. *Final Report of the NSF Workshop on Hyperbase Systems.* TAMU-HRL 93-002, Texas A&M University. July 1993.

Lewis, P.H., Wilkins, R.J., Griffiths, S.R., Davis, H.C. & Hall, W. *Content Based Navigation in an Open Hypermedia Environment.* In: The Proceedings of the IEE Colloqium on Document Image Processing and Multimedia Environments. IEE, November 1995.

Li, Z., Davis, H.C. & Hall, W. *Hypermedia Links and Information Retrieval.* In: *The Proceedings of the 14th British Computer Society Research Colloquium on Information Retrieval*, Lancaster University, 1992

Li, Z. *Information Retrieval for Automatic Link Creation in Hypertext Systems*. PhD Thesis, The University of Southampton, U.K. October 1993.

Lovins, J.L. *Development of a stemming algorithm.* Mechanical Translation and Computational Linguistics, 4, pp 22-31, 1968.

Maioli, C., Penzo, W., Sola, S. & Vitali, F. *Implementing External Anchors in the Dexter Reference Model*. Technical Report. Laboratory of Computer Science, University of Bologna, Italy. 1993a

Maioli, C. Sola, F. & Vitali, F. *Wide Area Distribution Issues in Hypertext Systems*. In: The Proceedings of ACM SIGDOC '93. ACM Press. 1993b

Malcolm, K.C and Poltrock, S.E & Schuler, D. *Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise,* In The Proceedings of Hypertext '91 San Antonio, Texas. ACM Press, 1991

McCracken, D & Akscyn, *Experience with the ZOG Human-Computer Interface System*. International Journal of Man-Machine Studies, pp 293-310, Vol. 21, 1984

Melly, M. *Co-operative Working in X-Cosm*. Poster at *ECHT '94 Edinburgh, Sept. 1994*.

NCSA *Mosaic.* http://www.ncsa.uiuc.edu/SDG/Software/WinMosaic/HomePage.html, 1994

Nelson, T. *Literary Machines*. Published by the author. 1981.

Nielsen, J. *Multimedia and Hypermedia: the Internet and Beyond.* Academic Press. 1995.

Nielsen, J. *The Art of Navigating Through Hypertext*. Communications of the ACM. 33(3). pp 296-310. 1990.

Noll, J. & Scacchi, W. *Integrating Diverse Information Repositories: A Distributed Hypertext Approach*. IEEE Computer 24(12) pp 38-45, Dec. 1991

Corba, OMG Technical Document 91-12-1, Revision 1.1. *The Common Object Request Broker: Architecture and Specification*, December 1991.

Østerbye, K. *Structural and Cognitive Problems in Providing Version Control for Hypertext*. In: D. Lucarella, J. Nanard, M. Nanard, P. Paolini. eds. *The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 11-22. ACM Press, 1992.

Pearl, A. *Sun's Link Service: A Protocol for Open Linking*. In: Proceedings of Hypertext '89. Pittsburgh, Pennsylvania, November 1989. pp 137-146. ACM. 1989

Poltrock, S. & Schuler, D. *Conference Report: Hypertext '93.* ACM SIGLINK 2(3), Dec. 1993

Price, R. *MHEG: an Introduction to the Future International Standard for Hypermedia Object Interchange.* The Proceedings of the First International Conference on Multimedia. pp 121-128. ACM Press, 1993

Rahtz, S.P.Q., Carr, L.A & Hall, W., *Creating Multimedia Documents: hypertext processing.* In: McAleese, R & Green, C., (eds.) *Hypertext: state of the art.* intellect, 1990.

Rhiner, M. & Stucki, P. *Database Requirements for Multimedia Applications*. In: Kjelldahl, L (ed.) *Multimedia: Systems, Interaction and Applications*, Springer Verlag, pp 269 - 282. 1991.

Ritchie, I. *The Future of Electronic Literacy: Will Hypertext Ever Find Acceptance?*. Keynote Speech at: *The ACM Conference on Hypertext, ECHT '92 Milano*, p 1. ACM Press, 1992.

Rizk, A., Maezieux, F. & Legger, A. *A Distributed Hypermedia Link Service on a WAN: An Experiment with MHEG on the ATM network*. Eurographics'94 symposium on multimedia/hypermedia in open distributed environments (EG-MM'94), Graz, Austria, June 6-9, 1994

Rizk, A. & Sauter, L. *Multicard: An Open Hypermedia System*. In: D. Lucarella, J. Nanard, M. Nanard, P. Paolini. eds. *The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, Italy, December 1992, pp 181-190. ACM. 1992

Robert M. Akscyn, Donald L. McCracken, and Elise A. Yoder. *KMS: A distributed hypermedia system for managing knowledge in organisations.* Communications of the ACM, 31. pp 820-835, July 1988.

Salton, G., Yang, C.S. & Wong, A. *A Vector Space Model for Automatic Indexing.* Comm. ACM 18(11), pp 613-620, Nov. 1975.

Schnase, J.L., Leggett, J.J., Hicks, D.L., Nürnberg, P.J. & Sánchez, J.A. *HB1: Design and Implementation of a Hyperbase Management System.* EPODD 6(1) pp 35-63. March 1993.

Schnase, J.L., Leggett, J.J., Furuta, R.K. & Metcalfe, T. (eds.). *The Proceedings of Digital Libraries '94.* Texas A&M University, June 1994.

Schütt, H.A. & Streitz, N.A. *Hyperbase: A Hypermedia Engine Based on a Relational Database Management System. in* A. Rizk, N. Streitz and J. Andre *eds. Hypertext: Concepts, Systems and Applications. The Proceedings of The European Conference on Hypertext,* INRIA, France. pp 95-108, Cambridge University Press. 1990

Shakelford, D.E., Smith, J.B. & Smith, F.D. *The Architecture and Implementation of a Distributed Hypermedia Storage System*. In: The Proceedings of Hypertext '93, Seattle, Washington, November 1993. ACM. 1993

Shipman, F.M. III, Furuta, R.,& Levy, D.M. *The Proceedings of Digital Libraries '95*. Texas A&M University, June 1995

Shneiderman, B. *User Interface Design for the Hyperties Electronic Encyclopaedia.* In: Proceedings of the ACM Hypertext '87 Workshop. The University of North Carolina at Chapel Hill. pp 189-194. Nov. 1987.

Smith, J.B. & Smith, F.D. *ABC: A Hypermedia System for Artefact-Based Collaboration*. In The Proceedings of the Third ACM Conference on Hypertext, Hypertext '91. San Antonio, Texas. pp 179-192. Dec. 1991

Streitz, N, Haake, J., Hannemann, J., Lemke, A., Sculer, W., Schutt, H. & Thuring, M. SEPIA: *A Co-operative Hypermedia Authoring Environment.* In: D. Lucarella, J. Nanard, M. Nanard, P. Paolini. eds. *The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 11-22. ACM Press, 1992.

Tichy, W.F. *RCS - A System for Version Control*. Software Engineering and Practice, 1985

Tompa, F.W., Blake, G.E. & Raymond, D.R. *Hypertext by Link-Resolving Components.* In: The Proceedings of Hypertext '93: the Fifth ACM Conference on Hypertext. Seattle, Washington, November 1993. pp 118-130. ACM Press. 1993

Uniform Resource Locators. http://info.cern.ch/hypertext/WWW/Addressing/URL/Overview.html, 1994

Vanzyl, A.J. *HyperTED Technical Description.* http://adrain.med.monash.edu.au/HyperTEDTechnical.html, 1993

Vanzyl, A., Branco, Heath. I & Davis. H.C. *Open Hypertext Systems An Examination of Requirements, and Analysis of Implementation Strategies, comparing Microcosm, HyperTED, and the World Wide Web.* Available form the authors. 1994

Vanzyl, A.J. *Open Hypermedia Systems: Comparisons and Suggestions for Implementation Strategies.* In: Wiil, U.K & Østerbye, K. (eds.). The Proceedings of the ECHT '94 Workshop on Open Hypermedia Systems, Edinburgh, Sept. 1994. Technical Report R-94-2038. Aalborg University. October 1994.

Wiil, U.K & Leggett, J.J. *Hyperform: Using Extensibility to develop dynamic, open and distributed hypertext systems. In:* D. Lucarella, J. Nanard, M. Nanard, P. Paolini. *eds. The Proceedings of the ACM Conference on Hypertext, ECHT '92 Milano*, pp 251-261. ACM Press, 1992.

Wiil, U.K. & Leggett, J.J. *Concurrency Control in Collaborative Hypertext Systems.* In: The Proceedings of Hypertext '93: the Fifth ACM Conference on Hypertext. Seattle, Washington, November 1993. pp 14-24. ACM Press. 1993

Wiil, U.K & Østerbye, K. (eds.). *The Proceedings of the ECHT '94 Workshop on Open Hypermedia Systems, Edinburgh, Sept. 1994.* Technical Report R-94-2038. Aalborg University. Oct. 1994.

Wiil, U.K. & Østerbye, K. *Experiences with HyperBase - A Multi-user Back-end for Hypertext Applications with Emphasis on Collaborative Support.* Department of Computer Science Technical Report R 90-38. Aalborg University, October 1990.

Wilkins, R.J. *The Advisor Agent: a Model for the Dynamic Integration of Navigation Information within an Open Hypermedia System.* PhD Thesis. The University of Southampton. U.K. Sept. 1994.

Wilkins, R.J., Griffiths, S.R., Lewis, P.H., Davis, H.C. & Hall, W. *Media-based Navigation with Generic Links.* To be published in the Proceedings of Hypertext '96. ACM, April 1996.

Wright, P & Lickorish, A. *An Empirical Comparison of Two Navigation Systems for Two Hypertexts.* In: McAleese, R & Green, C., (eds.) *Hypertext: state of the art.* intellect, 1990.

Yankelovich, N., Haan, B., Meyrowitz, N., & Drucker, S. *Intermedia: The Concept and the Construction of a Seamless Information Environment.* IEEE Computer, 21(1) pp 81-96. 1988

Glossary of Microcosm Terms

Anchor. Strictly speaking, an anchor is the object which is the end point of a link (source or destination), and which contains the information to enable the system to locate a persistent selection within a node. Microcosm does not have any identifiable separate anchor objects, but they are implicit within the definition of the link. The terms anchor, persistent selection and *link end-point* are used interchangeably within the Microcosm community. Anchors in Microcosm are not limited to representing persistent selections, and, for example, a source anchor could be an event in a temporal media viewer and a destination anchor might be some process, or a macro to be invoked by some viewer.

Application. In Microcosm the term "application" is used to refer to all the resources that comprise a specific project: the list of documents, the linkbases, the installed filter set and all default settings within the registry.

Button. A button is the binding of some specific selection and an action. The button will be coloured or highlighted in some way devolved to the viewer. When the button is double-clicked, the action will be performed. Typically the action will be "follow-link".

Computed Links. Computed links are achieved by information retrieval, using an inverted index of all the text known to the indexing program. The computed linker will offer the user a ranked list of the documents that contain, statistically, the best match of vocabulary to that in the query.

Document. The term “document” is used synonymously with “file” or “node”.

Document Control System (DCS). The DCS is the kernel of Microcosm which handles messages from viewers, and organises to dispatch viewers with given data, and to send messages to viewers.

Document Management System (DMS). The Document management system is a database which holds attributes for each document known to Microcosm. It is keyed by a unique identifier for each document, and attributes will include the description of the file, the location of the file on the file system, the logical type(s) of the file (where it will appear in the Microcosm hierarchy) and the physical type

of the file which determines which viewer will be used to display it. User defined attributes such as keywords and authors may also be stored.

Filter. A Filter is a process which receives Microcosm messages from the Filter Manager. On receiving a message it analyses the message for any tags it understands. If it does not understand any of the tags it will simply pass the message back to the Filter Manager. If it understands a tag it will take some action, and may then block the message, change the message, or create new messages, which are passed back on to the Filter Manager. Filters are used to implement the hypermedia functionality. For example the linker, the linkbases and the computed linker are all filters.

Filter Manager (FM). The filter manager is responsible for handling the ordered list of currently installed filters, and dispatching messages to each of these filters.

Generic Link. A Generic Link is a link that will be available from any place where the selection recorded in the link is made. For example, the text string "Microcosm" may be made into a link that points to the top of a file describing Microcosm. Thereafter it will be possible to follow this link from *any* occurrence of this text string in any file.

History. Microcosm maintains a history of all documents that have been seen by the user in the current session, along with the information about the method by which this document was reached, such as the name of the link that was followed, or the fact that the document was launched from the "select a document" dialogue.

Link. A link is a binding of a source (what must be done to make this happen) and a destination (what will happen if you do this). In reality most links are connections between some source selection, e.g. a string at some specific point in some file, to a destination selection, e.g. a selected area of a bitmap. However, links may also be anchored on events and processes.

Linkbase. A linkbase is a database (and the software to handle that database) which can store links, and can be queried to find links.

Linker. The linker is the program which intercepts messages to start and complete links, and then sends a create-link message to the linkbases. It contains the interface which allows the user to specify the type of the link.

Local Link. A local link is one which may be followed from any occurrence of an object within a specific file. For example, in a file describing the game of chess, a local link may be made from the string "king" to a file containing a picture of a the chess piece and to a text file containing the description of the moves that a king may make. A generic link would not have been appropriate in this example since the meaning of the word "king" may have different connotations outside the file about chess.

Message. A Microcosm message consists of a number of tagged fields. The action field defines the action that the sender intended to be carried out, and the remaining fields contain as much data as is needed (and possibly more) for any filter or viewer to handle the message.

Micon. A Micon is a moving icon which is an abstract of a piece of video.

Mimic. A mimic is a pre-defined guided tour through some set of resources.

Node. In Microcosm a node is a file or document.

Persistent Selection. In general hypertext terminology, a persistent selection is an active area in some data when viewed by a hypertext aware viewer. Generally these persistent selections are expected to be coloured or highlighted in some way, in which case the Microcosm equivalent of persistent selections are buttons. However, Microcosm also supports selections which are not highlighted in any way, but which, when selected by the user, will allow some action to be taken. In Microcosm a selection, whether persistent or not, is just one case of a link end-point or anchor.

Registry. The registry is a database which holds all the settings needed by components of Microcosm. It is conceptually the same as a Windows INI file.

Viewer. A viewer is any program which is able to display data and allow the user to browse that data. A viewer may be fully aware, in which case it handles all the Microcosm protocols, semi-aware , in which case it handles some of the Microcosm protocols, or unaware, in which case it handles none of the Microcosm protocols, but may still be able to provide hypertext functionality via the Universal Viewer. A viewer may also be an editor.