University of Southampton

The Advisor Agent: a Model for the Dynamic Integration of Navigation Information within an Open Hypermedia System.

by

Robert James Wilkins

A thesis submitted for the degree of
Doctor of Philosophy

in the
Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

29 November, 1994
The disorienting effect caused by reading information that has been placed within a non-linear space such as that provided by an open hypermedia system is described. This is followed by a discussion on the roles of navigation tools as a means to reducing this disorientation by providing both temporal and spatial contexts for the user.

Two sets of navigation tools have been implemented, for the Microcosm open hypermedia system. The first is a set of tools based on those found within other hypermedia systems, but that have been adapted to take advantage of the facilities offered by the Microcosm open hypermedia system. The second set consists of novel tools that have been designed and implemented to handle some of the features specific to the Microcosm system. Both sets of tools are designed to exist within the architecture provided by the current partitioned communication model supported by the Microcosm system. However, this model poses certain difficulties for both the implementors and users of these tools. A new, direct communication model, for Microcosm, has been designed and implemented to overcome these problems.

The new communication model allowed for the introduction of new processes, based upon the agent programming paradigm, into the Microcosm system. Such
agents can be used for three major areas within hypermedia: navigation, maintenance and information monitoring. All three areas are discussed and an implementation for an agent, the advisor agent, in the context of navigation is described.

Finally, the implementation of such an agent within an open hypermedia system highlights the need for both agent and hypermedia functionality to be provided by the operating system as an environment wide service. The work described within this thesis provides a base for research into this area.
## Contents

1. Introduction .......................................................................................... 1
   1.1. Hypertext then and now ................................................................. 1
   1.2. Navigation in hyperspace .............................................................. 4
   1.3. The agent paradigm ..................................................................... 6
   1.4. Thesis overview ......................................................................... 7
   1.5. Declaration ................................................................................ 9

2. Hypermedia navigation tools ................................................................. 10
   2.1. Cognitive clues provided by the linearity of text ................. 10
       2.1.1. How the linear thread aids navigation .......................... 10
       2.1.2. The physical manifestation of cognitive clues within a book ........................................ 12
       2.1.3. Extending the cognitive clues to large information collections .............................................. 14
   2.2. What is a navigation tool? ........................................................... 15
   2.3. Taxonomy of navigation tools ..................................................... 16
       2.3.1. Links ................................................................................. 16
           2.3.1.1. Explicit links ............................................................... 18
           2.3.1.2. Implicit links ............................................................... 20
           2.3.1.3. Typed links ................................................................. 21
           2.3.1.4. Conditional links ...................................................... 22
       2.3.2. Overviews .......................................................................... 22
           2.3.2.1. Local scope ................................................................. 24
           2.3.2.2. Global scope ............................................................... 25
           2.3.2.3. Dynamic scope .......................................................... 27
           2.3.2.4. User customisation .................................................... 27
           2.3.2.5. Dimensionality ............................................................ 27
           2.3.2.6. Complexity reduction ............................................... 28
       2.3.3. Histories ............................................................................ 29
       2.3.4. Tours ................................................................................ 30
           2.3.4.1. Tour creation ............................................................... 31
           2.3.4.2. Tour progression ......................................................... 32
           2.3.4.3. Extensions to the linear tour concept ...................... 32
           2.3.4.4. Tours: a link or document? ....................................... 33
4.3.5. Different levels of maps ........................................ 81
4.4. Summary ........................................................................ 81
5. Implementing novel navigation tools using filters .......... 83
  5.1. Memory filter ............................................................. 83
    5.1.1. A model of short term memory ......................... 85
    5.1.2. Mechanisms for presenting extra information ..... 86
    5.1.3. Use of colour .................................................. 87
    5.1.4. The memory filter in use .................................. 90
  5.2. Show links filter ....................................................... 91
  5.3. Graph analysis tool .................................................. 94
    5.3.1. Explicit links .................................................. 95
    5.3.2. Implicit links .................................................. 97
    5.3.3. Locating structure types ................................... 99
    5.3.4. Scatter graph representation of hypermedia
           structures ....................................................... 99
    5.3.5. Ordering the documents along the axes ............ 101
    5.3.6. Consistency of the scatter graph ...................... 102
    5.3.7. Example scatter graphs .................................... 103
  5.4. Active path guide .................................................... 105
    5.4.1. Active path guide implementation .................... 106
    5.4.2. Instant versus delayed response ...................... 108
    5.4.3. Advice construction ....................................... 108
    5.4.4. Levels of advice ............................................ 110
    5.4.5. Advice combination ....................................... 111
    5.4.6. Forms of advice ............................................ 113
    5.4.7. Pre-authored advice ..................................... 114
  5.5. Summary .................................................................. 115
6. Agents in the interface .................................................. 117
  6.1. Why do we need agents? ......................................... 117
  6.2. Implementation strategies ........................................ 119
    6.2.1. Artificial intelligence ..................................... 119
      6.2.1.1. Learning ............................................... 119
      6.2.1.2. Intelligent knowledge based systems .......... 120
      6.2.1.3. Divide and conquer theory ...................... 121
    6.2.2. Monitoring agents .......................................... 121
    6.2.3. Application based approach ............................ 121
8.3. Advice within the Microcosm system ......................167
8.4. The advisor agent: a proposed solution .................168
8.5. Implementation of the advisor agent .....................170
  8.5.1. Advice generation ....................................170
  8.5.1.1. What advice sources are available to the agent? ..........170
  8.5.1.2. What types of advice are offered by each source? ..........172
  8.5.1.3. Transforming Microcosm processes into advice sources ..........173
  8.5.2. Advice collection .....................................174
  8.5.2.1. Generic protocols for advice sources ..........174
  8.5.2.2. Dynamic nature of system .......................175
  8.5.2.3. Maintaining advice source information ..........176
  8.5.3. Advice amalgamation ..................................177
  8.5.3.1. The agent control interface .........................177
  8.5.3.2. Advice combination mechanism ..................178
  8.5.4. Advice presentation ..................................181
  8.5.4.1. Modes of interaction ............................181
  8.5.4.2. Abstraction of advice presentation .............181
  8.5.4.3. Dynamic nature of advice presenter and agent relationship ..........182
  8.5.4.4. Presentation of the combined advice to the user ..........182
     8.5.4.4.1. Memory filter ................................182
     8.5.4.4.2. Bar graph list advice presenter ..........184
     8.5.4.4.3. Circular plot advice presenter ..........185
8.6. An overview of the relationships between the advisor agent and Microcosm ......................187
8.7. Case studies of the advisor agent .......................188
  8.7.1. Case study one ......................................189
     8.7.1.1. Scenario A ..................................189
     8.7.1.2. Scenario B ..................................190
  8.7.2. Case study two .....................................191
     8.7.2.1. Scenario A ..................................191
     8.7.2.2. Scenario B ..................................192
8.8. Summary.............................................................................194
9. Future work..............................................................................196
  9.1. Further development of the DCM.................................196
    9.1.1. Process sequencing.............................................196
    9.1.2. Process control..................................................198
    9.1.3. Centralised system information............................200
    9.1.4. Inter-process communication protocols ..............201
    9.1.5. Seamless distributed heterogeneous
            architecture......................................................202
  9.2. Agents for monitoring and maintenance......................203
    9.2.1. Agents for monitoring hypermedia network
            content................................................................203
    9.2.2. Agents for maintaining the structural integrity
            of hypermedia networks..................................204
  9.3. The advisor agent.........................................................207
10. Conclusions............................................................................210
  10.1. The need for navigation tools....................................210
  10.2. Integration of navigation tools.................................211
  10.3. Navigation advice.....................................................213
  10.4. Communication models within Microcosm.................214
  10.5. The use of agents in hypermedia systems...............216
Bibliography.............................................................................220
List of Equations

Equation 3.1: Calculating no. messages generated in the SCM.......................................................... 52
Equation 7.1: Calculating no. messages generated in the SCM......................................................... 151
Equation 7.2: Calculating no. messages generated in the DCM...................................................... 151
List of Figures

Figure 2.1: Relationship between the number of pages on either side of the current page. .......................... 12
Figure 2.2: Cognitive clues within a book. .................................. 13
Figure 2.3: Simplified renditions of two pages. .......................... 14
Figure 2.4: The four types of basic explicit link. .......................... 19
Figure 2.5: The two types of compound explicit link .................... 19
Figure 2.6: An overview with local scope. ................................. 25
Figure 2.7: An overview with global scope. ............................... 26
Figure 3.1: The model for Microcosm V1.0. ............................... 40
Figure 3.2: The logical model of Microcosm. .............................. 43
Figure 3.3: Topological view of the current communication model. ........................................................................ 47
Figure 3.4: Real view of the current communication model .......... 47
Figure 3.5: Route for a message from a viewer to a filter. ............ 48
Figure 3.6: Route for a message from a filter to a viewer. .......... 49
Figure 3.7: Route for a message from a filter to a filter. .......... 50
Figure 3.8: Route for a message from one viewer to another viewer. ...................................................................... 50
Figure 3.9: Number of messages against position of first linkbase in chain. ......................................................... 52
Figure 3.10: Layered architecture of Microcosm ini. files. ............ 57
Figure 3.11: The Microcosm mimic system. ............................... 59
Figure 4.1: Microcosm 'IS.OPENING' message. ......................... 63
Figure 4.2: Microcosm 'AUTO.DISPATCH' message ................. 63
Figure 4.3: The history footprint interface. .............................. 64
Figure 4.4: Microcosm 'CLOSE.FILE' message. ....................... 65
Figure 4.5: A history list showing both open and closed documents. ...................................................................... 65
Figure 4.6: Microcosm 'LOCK.FILE' message ......................... 66
Figure 4.7: Microcosm 'UNLOCK.FILE' message ..................... 66
Figure 4.8: The relationship between the three parts of the mimic system. ....................................................... 68
Figure 4.9: The Microcosm mimic viewer. ............................... 69
Figure 4.10: The mimic progression methods ........................................ 70
Figure 4.11: Diagram showing the dynamic construction of 
local maps .................................................................................. 74
Figure 4.12: Relationship between the document list and the 
link matrix ................................................................................. 78
Figure 4.13: The lines $x = N$ and $y = N$ in the link matrix ...... 78
Figure 4.14: Local map showing links into and out of a 
document ...................................................................................... 79
Figure 5.1: Microcosm 'Available links' filter ................................. 85
Figure 5.2: Default colours used by the short term memory 
filter and their meanings ............................................................. 88
Figure 5.3: STM filter showing destination documents in 
different memory bands .............................................................. 90
Figure 5.4: Percentage of links whose start anchors are less 
than three words in length .......................................................... 92
Figure 5.5: Microcosm 'SHOW.LINKS' message ......................... 93
Figure 5.6: Example of a scatter graph representation of 
links ............................................................................................. 95
Figure 5.7: Structure objects within a hypermedia network .... 97
Figure 5.8: Changes to the structure due to the inclusion of 
generic links .............................................................................. 98
Figure 5.9: Scatter graph of hypermedia structure from 
figure 5.7 .................................................................................... 100
Figure 5.10: Examples of band structures in a scatter graph .... 102
Figure 5.11: Scatter graph representing the Cell Biology 
hypermedia network .................................................................. 103
Figure 5.12: Scatter graph representing the Yugoslavia 
hypermedia network .................................................................. 104
Figure 5.13: APG association matrix ............................................ 107
Figure 5.14: The APG advice combination algorithm ................. 112
Figure 5.15: An example of APG advice combination ............... 113
Figure 6.1: Overall structure of the COKES system .................. 127
Figure 6.2: The Envoy framework take from (Palaniappan et 
al, 1992) .............................................................................. 128
Figure 7.1: Microcosm SCM architecture .................................. 145
Figure 7.2: Microcosm DCM architecture ................................. 146
Figure 7.3 : Messages generated along route from process A to B, (a) for SCM and (b) for DCM .........................146
Figure 7.4 : Real and abstract communication networks ...........148
Figure 7.5 : Number of messages against position of first linkbase in chain ......................................................152
Figure 7.6 : A hybrid communication model (HCM) ...............160
Figure 8.1 : Advice source locations within the Microcosm system ....................................................................171
Figure 8.2 : The agent control interface ........................................177
Figure 8.3 : The advice combination algorithm .......................179
Figure 8.4 : Example of advice combination .........................180
Figure 8.5 : Routes for advice from advice presenter to the memory filter .........................................................183
Figure 8.6 : The bar graph list advice presenter .......................185
Figure 8.7 : The circular plot advice presenter .......................186
Figure 8.8 : Relationships between the Advisor agent and Microcosm ............................................................187
Figure 9.1 : The simulation of process sequencing ....................197
List of Tables

Table 7.1 : Table showing number of messages in system ..... 157
Table 8.1 : Types of advice generated from navigation
centred sources .......................................................... 172
Table 8.2 : Types of advice generated from filter centred
sources ................................................................... 173
Table 8.3 : Types of advice generated from system centred
sources ................................................................... 173
Acknowledgements

The work in this thesis has been produced within the invigorating environment provided by the Image and Media Laboratory of the University of Southampton. I would like to thank the team of programmers involved with the PC implementation of the Microcosm project, of which this work forms a part, namely Ian, Gary, Mark, Nick, Stuart and Simon. I would also like to thank some of the other members of the group namely Hutch, Andy, Rupert, Rory, Jon, Richard and Dr. Steve Rake. I thank Dr. Mark Dobie for his eagle-eyed correction spotting of the final drafts of this thesis, and Skippy for helping me maintain my sanity during the final stages of this thesis’s preparation.

Finally, I would like to thank Prof. Wendy Hall for allowing me to undertake this PhD. as part of the Microcosm team and for being my supervisor during the three and a half years that it has taken.
1. Introduction

This chapter begins by providing a brief history of hypertext/hypermedia. It then continues by discussing the problem of navigation within non-linear information such as that presented by a hypermedia network, offering agents as a possible solution. The chapter is concluded by a chapter based summary of the content of the remainder of this thesis.

1.1. Hypertext then and now

Hypertext is not so much a new idea, rather it is a refinement of information systems that have existed since the first use of writing. Countless people have contributed to its growth and development, each holding different conceptions about its uses and functionality. This section serves to present a historical perspective of the evolution of hypertext from its embryonic origins up to its current state of maturity.

The essence of hypertext is the ability to allow arbitrary fragments of information to be linked together. Such linking can be seen in many documents in which references to other parts of the document or other documents, constitute a major portion of the work. Such features can be seen in the Talmud (an ancient religious work) with its heavy reliance on nested commentary and annotations, or in works by Aristotle with their dependence on references to other sources (Conklin, 1987). Looking at more modern works we can see examples of hypertext links in reference books, scientific journals and more obviously in the encyclopaedia. In the sense that each work can be viewed as a graph of textual nodes joined by referential links, they are very old forms of hypertext. As one reads an article or definition, explicit references to related items indicate the locations of more information about those items.

The majority of modern proponents of hypertext insist that navigation through the hypertextual space generated by the hypertext must be computer supported in order to qualify as true hypertext (Conklin, 1987). If this is the case then the field of hypertext is narrowed considerably and its history is subsequently shortened.
The earliest visions of hypertext focused on its ability to integrate vast volumes of information, allowing it to be readily accessed via a simple and consistent interface. Furthermore, it was aimed at providing a networked authoring system allowing its readers to extend dynamically the corpus of material without defacing the originals. The latter capability diminished the differences between authors and readers by providing either with the ability to create relationships between arbitrary sections of information.

The first true hypertext system was designed by Vannevar Bush, President Roosevelt's science advisor. He described a system he called Memex in his article "As we may think" (Bush, 1945) in which he also called for a major post-war effort to mechanise the ever increasing scientific literature system. The Memex contained a vast library in addition to personal notes, photographs, and sketches. It would have several screens and most importantly the facility for constructing labelled links between any two points in the entire library. However, Bush's Memex used microfilm and photocells to implement its various mechanisms.

Just less than two decades later, Douglas Engelbart at Stanford Research Institute was influenced by Bush's ideas. In 1963 he produced "A conceptual framework for the augmentation of man's intellect" (Engelbart, 1963) in which he envisioned that computers would usher in a new stage of human evolution. Five years later, in 1968, Engelbart's ideas about human augmentation had become more focused, resulting in the implementation of the NLS (oN Line System) (Englebart & English, 1968).

All files within the NLS system were structured into hierarchical segments called statements (limited in length to three thousand characters). Each segment bore an identifier that allowed it to be uniquely positioned within the file. Any number of links could be established between statements within files and between files. Thus the structure was primarily hierarchical but allowed non-hierarchical links as well. NLS was evolved over the years. It is now called Augment (or NLS/Augment) (Engelbart et al, 1973) and is marketed by McDonnell Douglas as a commercial network system.

Whilst Engelbart was developing Augment another hypertext visionary, Ted Nelson, was developing his own ideas about augmentation. However, the emphasis of his work was to produce a unified literary environment on a global scale. He
named his hypertext system Xanadu (Nelson, 1980; Nelson, 1987), after the "magic place of literary memory" in Samuel Taylor Coleridge's poem *Kubla Kahn*.

In Xanadu, storage space is reduced by heavy use of links, only the original document and changes made to it are stored. The system then reconstructs previous versions of documents from these. Xanadu's long range goal has been to facilitate the revolutionary process of placing the entire world's literary corpus on-line, allowing it to be accessed by anybody with a computer. Nelson predicts that the advent of on-line libraries will create a whole new market for the organisation and indexing of this immense body of information.

Many other hypertext systems have been designed and implemented. For example, Textnet (Trigg et al, 1986), gIBIS (Conklin & Begeman, 1988), KMS (Aks cyn et al, 1988), Hyperties (Shneiderman, 1987; Shneiderman, 1989), Notecards (Halasz et al, 1987; Halasz, 1988) and Intermedia (Haan et al, 1992). Reviews of these can be seen in "Hypertext: An introduction and survey" (Conklin, 1987).

The focus of hypertext research has shifted from aiming to produce systems that encompass the hypertext data to producing systems that work together with the user's computing environment, allowing linking as a pseudo operating system function. The shift in research has been driven by the needs of hypertext in dealing with large collections of data (Halasz, 1988; Malcolm et al, 1991; Halasz, 1991; Davis et al, 1992). Peter Brown summed up the problems involved (Brown, 1987) with the following text:

"A second reason for failure is that the tool is an island to itself and cannot be combined with other tools. Those of us that expect the whole world to rewrite its documentation to fit the needs of our new hypertext system are unlikely to have our expectations fulfilled. Instead we must capture existing documents and have some way - even if crude - of automatically imparting structure to it. We must also work with existing tools such as spell checkers or encryption programs. This is to some extent a research area but more, I expect, a question of curbing some of our wilder aspirations so that, following a recurrent theme of this paper, we fit the world as it is rather than the world as we would like it to be."

The class of hypertext system designed to counteract the problems discussed above has been called open. As yet no concrete universal definitions for exactly what open really means have been agreed upon. Various aspects of exactly what open
really is have been discussed in the hypertext literature (Pearl, 1989; Shipman et al, 1989; Creech et al, 1991; Fountain et al, 1990; Davis et al, 1992; Maurer & Tomek, 1990).

Generally, a system is deemed to be open when it allows users to link between material without needing to change or alter that material in any way. Open systems in effect act as a virtual linking layer situated above the user's existing applications and information. By utilising this layer users can effectively transform their set of discrete tools into a fully integrated information environment.

Following on from this is the aim to integrate the linking functionality provided by open hypertext into the user's environment at an operating system level. It is only at this level that it becomes economic for all application developers to begin to take advantage of the hypertext functionality.

1.2. Navigation in hyperspace

Since hypertext's conception, people have been citing disorientation as one of its major drawbacks (Conklin, 1987). Although hypertext provides a powerful mechanism for the organisation of information into complex structures, it also introduces problems in answering the following two questions:

- Where is the user within the information network?
- How does the user get to some other place that they know (or think they know) exists in the network?

These two questions have become the classic definition of the navigation problems inherent in the hypertext ethos. Of course users can suffer the same disorientation problems described above in linear text. However, they can resolve them by simply looking for the desired text before or after their current location. Hypertext provides more degrees of freedom, more dimensions in which one can move, and hence a greater potential for the user to become lost or disorientated.

Two major solutions for coping with disorientation have been suggested: graphical browsers and query/search mechanisms. Browsers rely on the extremely highly developed visuospatial processing of the human visual system (Utting &
Yankelovich, 1990). By placing the nodes of the hypertext in a multi-dimensional space and making them individually distinguishable (e.g. varying their colour, shape, size or texture) designers are able to create quite viable virtual spatial representations of the structure inherent in the hypertext information. Users can then use the same visual clues to orient themselves as they do when walking through a familiar city. However, browsers are only functional in certain restrictive circumstances, for example, in small sparsely connected hypermedia networks (Utting & Yankelovich, 1990). Query/search mechanisms allow users to jump from location to location through the hypertext structure without following its links. In doing so they will risk the possible failure to gain the additional information provided by that structure in the form of linked relationships between the nodes of information.

The problems of disorientation are not unique to hypertext, they have been shown to exist in many data structures. Furthermore, it has been argued that studies of navigational strategies are needed in order to support such strategies within new generations of software (Canter et al, 1985).

A third solution to the problems of user disorientation has been suggested in the introduction of metaphors to ease their navigation. The theory behind the use of metaphors is that users of a new system do not start with a tabula rasa (McKnight et al, 1989); rather they bring to that system their past experience. They attempt to construct a mental model of that system in terms of this experience (Norman, 1983).

It has been suggested (Carroll & Thomas, 1982) that:

"If people employ metaphors in learning about computing systems, the designers of those systems should anticipate and support likely metaphorical constructions to increase the ease of learning and using the system"

However, rather than anticipate such metaphor construction, the general approach has been to provide a metaphor and hope that the users can use it. As the term navigation suggests, the most commonly provided metaphor is that of travel. The use of different travel metaphors has been studied (Hammond & Allinson, 1988) and it was found that users of such metaphor embedded systems were able to employ those metaphors with little difficulty.
The problems of navigating through hypertext structures are discussed in more detail in chapter two.

1.3. The agent paradigm

An agent, in the context of computer technology, is defined as a bundle of functionality that performs some task for a person, either in real-time or off-line (Laurel, 1991). Examples of such processes can be seen in the mail sorting agent within the Object lens system (Crowston & Malone, 1988; Lai et al, 1988), the NewWave system (Hewlett Packard, 1989) or within the Envoy framework (Palaniappan et al, 1992). Chapter six is devoted to the examination of such processes in more detail.

Agents may bring about a shift in user behaviour by relieving them of the responsibilities associated with sifting and sorting vast collections of information. Instead, these tasks will be delegated to the agents. People will be able to dispatch them to monitor sources of data likely to contain useful information. For example, as part of an experimental evaluation of the Clipping Service project (Gifford & Francomano, 1990) it was demonstrated that an active, user programmed agent is more efficient in retrieving information of interest to the user than the user is in actively seeking out the same information.

The benefits of changing to agent based systems, thereby shifting from sifting and sorting behaviour to specifying and delegating behaviour can be listed as:

- Automation of repetitive tasks
- Delayed execution of tasks
- Autonomous monitoring of constantly changing parameters (e.g. disk space or file ownership)
- Additional services (for example, monitoring of collections of documents edited by a group of people)

Agent based systems offer users opportunities to change the whole manner in which they interact with their computer environment. They will significantly
improve users' ability to manage information within their computer environments. However, their impact to date has been minimal. Perhaps the biggest contributory reason for such a slow uptake in their implementation is the difficulty of integrating them into the users' environment at a system level (a problem also shared by hypertext). It is only at this level of integration that they can provide the substantial benefits needed to make the agent programming paradigm economic.

1.4. Thesis overview

The second chapter of this thesis begins by outlining the cognitive clues provided by linear text that allow readers to orient themselves within its content. It continues by discussing how these clues are lost when the same content is placed within a non-linear hyperspace as provided by a hypermedia system, and then proposes that hypermedia navigation tools provide a mechanism to regain such cognitive clues. Chapter two concludes with a taxonomy of navigation tools, allowing them to be classified under a variety of different categories.

Chapter three discusses the development of an open hypermedia system within the Department of Electronics and Computer Science at the University of Southampton. Microcosm, the system in question, is then examined in detail. The chapter focuses on the areas of the system’s design which affect the implementation of navigation tools within it.

Chapters four and five describe the implementation of a set of traditional and novel navigation tools, respectively. The traditional tools are those that can be found in other systems, but that have been adapted to utilise the facilities provided by Microcosm. In contrast, the novel tools are those that have not been implemented elsewhere and are hence only to be found within the Microcosm system. They are provided as partial solutions to many of the problems caused by the design of Microcosm, and other such open hypermedia systems, in the context of navigation tools.

Chapter six details the trend towards the implementation of agent type processes within modern computer environments. It includes a brief description of research work in the field of agents and then discusses how agents can be employed within a hypermedia system. The chapter hypothesises that Microcosm, due to its
co-operative modular nature, can be thought of as a community of agents. It is then proposed that an agent can be utilised within Microcosm, aiding users as they navigate through a hypermedia network.

The communications model implemented within the current version of Microcosm (the partitioned communications model, discussed in chapter three) is re-examined in chapter seven. The chapter continues by proposing a new direct communication model for the Microcosm system. The design and implementation of this model in the form of a prototype, constructed from a subset of the Microcosm process, is discussed. The performance and effect on the user of this prototype are then described. The chapter is concluded by proposing a third communication model for Microcosm, the hybrid model, which is constructed by combining the current segregated model with the direct model implemented within the prototype system.

A solution to the problems caused by the disparate modular nature of the Microcosm navigation tools is proposed in chapter six. This solution takes the form of an agent that arbitrates between the user and the navigation tools. However, the segregated communication model renders the implementation of such an agent impossible. In contrast, the direct and hybrid communication models described in chapter seven do allow such an agent to be implemented. Chapter eight describes the design and development of the Advisor agent whose function it is to aid the users in their navigation through the potential tangled web of the hypermedia information. The chapter concludes with two case studies illustrating the Advisor agent's ability to integrate dynamically advice drawn from a variety of sources within the Microcosm system. The Advisor agent then provides this combined advice through a suite of advice presentation modules.

Finally, chapter nine outlines several potential areas, in which future research can be carried out, that have been highlighted by the developments and research described within this thesis. Chapter ten presents a final overview of the work that has been undertaken and presents a summary of the conclusions that have been drawn.
1.5. Declaration

The work contained within this thesis has been carried out within a collaborative research group. It is all the author's work, with the exception of that described in sections 3.1, 3.2, 3.2.1, 3.2.2, 3.2.3, 3.3, 7.2, 7.2.1, 7.2.2, 7.2.4, 7.4 where other members of the research group have been involved.
2. Hypermedia navigation tools

The following chapter begins by examining the navigation clues provided by the linear nature of text (e.g. both fictional and non-fictional books etc.). It then discusses how these clues are lost when the text is placed into a hypermedia system and presents the concept of navigation tools as a mechanism to regain them.

The chapter then continues by presenting four categories for the classification of navigation tools. This includes a discussion of the tools that exist within each category. The chapter then describes navigation clues that are obtainable from sources other than the navigation tools discussed so far. The chapter concludes with a summary of its key points.

2.1. Cognitive clues provided by the linearity of text

This section describes the cognitive clues that linear text provides, and how these clues help the reader to navigate through the associated information. The majority of navigation problems occur when the linear nature of the text is lost through its introduction to a hypermedia system. Such problems are discussed in much of the hypertext literature (Bernstein, 1988; Conklin, 1987; Duchastel, 1991; Nielsen, 1990; Nielsen, 1990b; Marchionini & Shneiderman, 1988; Oren, 1987).

The following sections discuss, in greater detail, the help provided by the linearity of text, its representation in a book and how similar navigation clues can be used in larger information collections.

2.1.1. How the linear thread aids navigation

Linear text, such as that found in the majority of books, provides readers with a set of very important subliminal cognitive navigational clues (McKnight et al, 1991; McKnight et al, 1993). The information is presented in such a manner that at any point in its content the author will know what has been explained to the reader and what has not. Any paragraph or sentence written will use context derived from terms and concepts explained previously in the text.
Readers will start at the beginning of the text and read until they reach the end. As they read, ideas and concepts will be presented to them by the author. These will be continuously expanded and reinforced by the succeeding text.

Readers are presented with this linear thread by the authors and it is the author who controls how the concepts are expanded and consequently how the reader will read them. If the reader attempts to read the text in an order other than the one intended by the author they will risk misunderstanding it. This mis-comprehension is a result of reading the text out of its original context.

The linear thread concept is well illustrated by the following extract from a paper by J. D. Bolter and M. Joyce (Bolter & Joyce, 1987):

'The temporal character of interactive fiction is also something new. In printed fiction the author is free to manipulate the time in which his story takes place, and every good author does so. However, the plot, the author's manipulation of story time, is itself static. Printed fiction is one-dimensional in the sense that we need only one dimension in order to represent the experience of reading it. The episodes (chapters, sections, cantos, books, volumes) are realised through time as we read. The links between the episodes are fixed in the course of writing, and the reader has no obvious and effective way to alter the order of reading. In electronic fiction multiple links among episodes allow our temporal experience of the plot itself to vary. Time may be fluid in a printed novel, but the presentation of time is fixed, as the fixed pages of the book mark the progressive stages of the narrative. The author manipulates words to create a single narrative structure. The author writes with words, not with structures. The electronic medium permits writing of a second order, a writing with narrative units, in which the structure of the text becomes truly fluid and indeed geometric. The author becomes a geometrician or architect of computerised "space" (as computer memory is in fact called by programmers); he fills his space with a special pattern of episodes and links that define a kaleidoscope of possible structures. The success of his work will depend upon the poetic rightness of the way in which the pattern is realised in the act of reading.'

The ideas and concepts included in the above quote deal largely with fictional work. However, there is no reason why they should not be applied to less linear material (e.g. technical reference books, or manuals). Even an encyclopaedia, a most
extremely non-linear book, contains discrete capsules of linearity. Each definition within the encyclopaedia is written as a linear thread.

The linear nature of traditional text has one further major advantage; it supports the random access of its content. It could be argued that hypermedia systems also support such an access mechanism. However, even though the hypermedia system allows such access it does not really support the reader in its use. For example, when the reader arrives at a node within a hypermedia system they have no easy mechanism for locating the pre-requisite nodes that will explain the current node's content. In contrast, if a reader of a book chooses a page at random, they can be sure that any ideas or concepts which they do not understand, will be explained in the previous pages. The exception to this would be a work that requires the reading of previous disparate works to fully understand its content.

2.1.2. The physical manifestation of cognitive clues within a book

Readers of a book have a wide variety of tools at their disposal allowing them to visualise their position within its content. To be more precise, these tools allow them to orient themselves within the thread of the book. In addition, these tools also allow readers to relocate their position, should they be interrupted during the reading process (e.g. if they have to answer a phone call).

The simplest of these tools is the relative thickness of the book to either side of the current page. For example, if they are near the beginning of the book they will see more pages to the right of the book than to the left. As they continue through the book the number of pages on the left will increase, redressing the balance. At the end the pages will all be on the left, as there are no more pages to read. This concept is illustrated by the following set of diagrams (see figure 2.1).

![Figure 2.1: Relationship between the number of pages on either side of the current page.](image)

The next tool is provided by the numbering of the pages within a book. This provides readers with both absolute and relative positioning of the current page.
Absolute positioning depends on the reader knowing the numbers of the first and last pages in the book. If a reader is presented with two pages they can immediately see the order in which they should be read, by looking at the page numbers.

Further tools that follow the page numbering principle are the chapter numbers and section numbers. These allow readers not only to position the page within the overall content of the book, but also to group pages together. If the book is well written then these page groups will contain related information and concepts.

The availability of these information sources varies from book to book. For example, pages of fictional books usually contain only a page number. In contrast, most technical books contain chapter, and even section, information on each page. These tools are illustrated in the following diagram (see figure 2.2).

![Diagram of a book's layout with labels for Chapter Heading, Page Number, Layout of page, Thickness of book on either side of the current page, and Page footer.](image)

Figure 2.2: Cognitive clues within a book.

These clues, when combined, provide the reader with a strong sense of context. The reader can lose track of the current context, either through an external interruption or a momentary lack of concentration. If this happens then they can use the clues to reorient themselves and thus regain the lost context.

The physical attributes of the book coupled with its linear nature allow readers to temporarily mark their position within its content and suspend the reading process. The reader places a bookmark at the current position and when they return they are able to regain the context of the book by re-reading the pages preceding it.
Finally, books allow readers to be serendipitous; they are able to leaf through the book's pages looking for items that may 'catch their eye' and prove interesting. If the reader is leafing quickly through a book, they will not be able to perceive the wealth of detail contained on the average page. Instead, they will perceive simplified renditions of each page as it flicks past. Examples of such renditions can be seen in the following diagrams (see figure 2.3).

![Figure 2.3: Simplified renditions of two pages.](image)

The above figure illustrates two pages from a book. Alongside each is an example of a simplified rendition of their content. The text blocks are coloured in black and the images in grey. The difference in colour represents the differences in density of the respective page objects. It is clear that even given the lack of detail contained within the simplified pages the reader can distinguish one from the other.

### 2.1.3. Extending the cognitive clues to large information collections

So far we have only examined cognitive clues provided by small collections of information in the form of books. However, similar clues can be found in larger collections, for example, a shelf of books or a library.

Books on a bookshelf are usually ordered alphabetically by the name of the author. If this is the case then the relative positions of the books can be used as an additional navigation clue. Books can be taken off the shelf and repositioned once again by inserting them at the relevant shelf position. Readers can also use the physical attributes of the book (e.g. the colour, shape, size or lettering on the spine) as an aid to navigation. These navigation clues are used to great effect within the WALT system (Frisse et al, 1991).
If we examine a library, we can see that books are grouped together by topic. Within these groups the navigation clues provided by the bookshelf can be used. Furthermore, books within a library are sorted by the use of some universal numbering scheme (such as the Dewey-decimal scheme). This can be used to help the reader navigate within the information collection.

2.2. What is a navigation tool?

Nielsen (Nielsen, 1990; Nielsen, 1990b) describes the user's disorientation while navigating through an information network as 'one of the major usability problems in hypertext'. This disorientation can be sub-divided into three areas. These are:

- Where am I?
- How did I get here?
- How do I get from here to my next destination?

The answers to the above three questions with reference to linear text become obvious. Readers are able to determine their current location by looking at the clues discussed previously (e.g. the page number or the thickness of the book on either side of the current page). They know that they arrived at the current page by reading the preceding pages (in the majority of cases). In addition, they know that in order to reach their destination all they need to do is to read more of the book.

If the answers are sought within a hypermedia system the problems become more apparent. The information within the hypermedia system is, by its nature, essentially non-linear. This non-linearity is the root of the navigation problems. Navigation tools are tools which help the user to answer one or more of the above questions in spite of the inherent non-linearity of hypermedia information.
2.3. Taxonomy of navigation tools

The navigation problems discussed above occur when text is placed into a non-linear information space (such as that provided by a hypermedia system). The subsequent sections discuss each of the following four categories of navigation tool:

- **Links** - These are one of the fundamental building blocks within all hypermedia systems.

- **Overviews** - These provide users with representations of the hypermedia network structure contained within the hypermedia system.

- **Histories** - These provide users with a list of their previous movements within the hypermedia information (or document set).

- **Tours** - These provide users with a trail through the information network. The majority of the user's navigation problems are removed whilst they are following such a trail.

The discussion of each category includes a description of a variety of tools that fall within its bounds. Furthermore, each section contains a description of how the particular class of tool helps the user to navigate within the hypermedia information.

### 2.3.1. Links

Hypermedia systems allow their users to link related pieces of material to one other. Thus, a hypermedia network is a collection of material that has been linked together, because parts of it relate to other parts. Therefore, the concept of the link is fundamental to the whole hypermedia paradigm. Links, together with the raw chunks of data (the documents) which they connect, form the basic building blocks of any hypermedia network.

The other navigation tools (discussed in the following sections) are strongly related to the link. For example, the history is a recording of a user's progress, mostly by link following, through the information network. An overview is a
A pictorial representation of the network showing both the documents and links. Finally, the tour can be thought as a super link (Nielsen, 1990) allowing documents to be chained together in a linearisation of the hypermedia network.

Links can be dangerous to the user, they signify that the documents at either end are related. More precisely, links convey the strong impression that there exists a coherent, purposeful and above all useful relationship between the documents at either end. Landow (Landow, 1987) discusses this danger and makes the following three points:

- Hypertext links condition users to expect purposeful, important relationships between linked materials.
- The emphasis upon linking materials in hypertext stimulates and encourages habits of relational linking in the user.
- Since hypertext systems predispose users to expect such significant relationships among documents, those links that disappoint such expectations will appear particularly incoherent and insignificant.

The time taken for the user to traverse a link will affect their perception of the relationship between the materials at either of its ends. For example, if the journey from the start of the link to its end takes long enough, the context provided by the link's start anchor will be forgotten before the user reaches its end. This premise was judged to be of great importance by the designers of the KMS system (Akscyn et al, 1988). As a result, all links in the system have to be traversable in less than one second, thus allowing users to view the end of the link whilst remembering the context of its start anchor. Creators of the system believed that such quick link traversal helped ease the burden of user navigation.

All links can be divided into two categories depending on whether their end points (or anchors) are defined when the link is created. These categories are:

Explicit - A link is explicitly defined if both its start and end anchors are defined when it is created.

Implicit - A link is implicitly defined if one or both of its anchors are not instantiated until the link is followed by a user. When this happens the anchors will be resolved and the link will become instantaneously explicit.
The following sections discuss both categories of link in more detail. Each section contains a description of different subclasses within each category and includes examples from actual systems where appropriate.

2.3.1.1. Explicit links

Explicit links are links that have their start and end anchors defined when the link is created. That is to say, when the link is constructed the author is creating a connection between two definite locations. The following four classes of explicit link can be defined (illustrated in figure 2.4):

**Document to document** - This is the most basic type, both the start and end anchors are represented by whole documents. A user can select the link from anywhere within the start document and the system will present the end document.

**Point to document** - This is the first of three refinements of the document to document link. The start anchor is defined at a higher granularity (e.g. a phrase or area of an image). The end anchor is still defined as a document, thus giving us a point to document link.

**Document to point** - This is the second of three refinements to the document to document link. This time the end anchor is defined at a higher granularity, with the start anchor still represented by a complete document.

**Point to point** - Finally, if both the start and end anchors are defined at a higher granularity we obtain a point to point link. With both anchors being defined in this way there is no reason why they should not be in the same document. This is the only one of the four link types described in which it makes sense to place both the start and end anchors in the same document.
So far we have only discussed one to one links (i.e. each link has one start and one end anchor). However, this basic building block can be used to construct the following two link types (illustrated in figure 2.5):

**One to many** - Each link has one start anchor but many end anchors. These can be constructed from a set of one to one links that have a shared start anchor but different end anchors.

**Many to one** - Each link has one end anchor but many start anchors. Such links can be constructed from a set of one to one links, having a shared end anchor and separate start anchors.

At this point we have discussed the basic building block link (point to point) and its use in the construction of more complex compound link structures (e.g. one to many and many to one). We now re-consider the point to point link, making it bi-directional. This is to say a user can travel from the end to the start anchor as easily as travel from the start to the end anchor. This ability renders the one to many and
many to one links equivalent. A many to one link can be viewed as a one to many link traversed in the opposite direction.

When users follow a one to many link one of two events can occur. Firstly, they could be presented with all of the end anchors simultaneously. This would be very system intensive and would be likely to cause confusion. Secondly, they could be presented with a list of the possible end anchors. Such a list would allow them to choose which of the many virtual one to one links they wish to follow, as opposed to following them all at once. The implementation of such a list device is discussed in more detail in section 5.1.

Creating explicit links is a very intensive task for the author, as they must define each start and end anchor for every link. As the number of documents in the document set increases so too will the number of links. The task of authoring every link as an explicit link soon becomes too large, thus the author will need to use implicit links to reduce the quantity of work needed to create large hypermedia networks (see the next section).

2.3.1.2. Implicit links

Implicit links aim to reduce the authoring effort associated with the creation of large hypermedia networks. The anchors of explicit links are defined when the link is created. In contrast, anchors for implicit links are instantiated when the link is followed by the user. Implicit links reduce the authoring effort by reducing the number of anchors that the author needs to create.

When an implicit link is followed, its anchors are resolved and it instantaneously becomes an explicit link. Example resolution mechanisms are:

**Database resolution** - e.g. All links within the Microcosm system are stored within link databases (or linkbases), following any link results in a linkbase search. This technique provides several advantages, for example all links can have an associated scope; the system provides a type of implicit link called a *generic link* (Fountain et al, 1990; Heath, 1992) which can be followed from any occurrence of the anchor phrase within any document (discussed in more detail later in this thesis). The linkbase effectively resolves anchors within the links allowing users to follow them.
**Full text retrieval** - This mechanism involves the statistical analysis of the document's content (Frisse, 1987; Frisse, 1988; Li et al, 1992; Salton, 1989; Salton & Buckley, 1989). All text documents within the hypermedia structure are first indexed. The index provides a relationship between a specific word and the documents that contain it. Once the indexing has taken place the system is able to statistically associate any piece of text to any text documents within the network. These associations represent a set of implicit links. Microcosm includes such a full text retrieval system in the form of the computed linker filter (Li, 1993).

**Knowledge based resolution** - This mechanism involves the use of a knowledge based engine and knowledge rules. The implicit links are contained in the rules. Their resolution is achieved through the use of the knowledge engine. For example, the LINCTUS PB prototype (Briggs et al, 1991) implements such an implicit linking system. Each document is represented by a set of formal logic statements concerning its content. The system is able to generate the links between the documents by using the knowledge engine to apply a set of knowledge based rules to the sets of logic statements.

Because implicit links are not instantiated until they are resolved (most commonly by the user traversing them) they become very difficult to represent within the system. For example, explicit links can be highlighted to the user (in the form of buttons) but implicit links cannot. The unresolved nature of implicit links causes problems for the navigation tools discussed in chapters four and five. For example in order for a navigation tool to construct a complete representation of the hypermedia network it must resolve all of the implicit links itself.

2.3.1.3. Typed links

The links discussed so far (i.e. explicit and implicit) contain no meaning, they simply indicate that document A is connected to document B. It has been postulated that this lack of inherent meaning within a link causes the majority of the navigation problems suffered by users (Halasz, 1988).

The content of a document contains its meaning, the link, however, has no such content and so has no meaning. Such content can be introduced into the link in the form of keywords (or link types). For example the Notecards system (Halasz et al,
1987; Halasz, 1988) contains such typed links. Two link types from within the system are:

**Source** - Document A is the source of the information in document B.

**Support** - The content of document A supports that of document B.


2.3.1.4. Conditional links

Finally, the concept of a link can be extended to allow for the association of conditions with each link. For example, the hypermedia network could be changed depending on the status of the user; tutors may wish to see different links than their students. Furthermore, the links provided for users could depend on which documents they have already seen. A use of such system state can be seen in the StorySpace system (Bolter & Joyce, 1987; StorySpace, 1991). Section 5.2 discusses a navigation tool that effectively generates conditional links by applying such a concept of system state to non-conditional links in the Microcosm system.

2.3.2. Overviews

Hypermedia promises the ability to produce complex, richly interconnected and cross referenced bodies of multimedia information. In reality, these information networks can be a confusing, disorganised tangle of haphazardly connected documents. Users that are attempting to make sense of the information network will need a sense of spatial context.

The overview navigation tool will provide users with this spatial context, allowing them to re-establish a sense of location. Furthermore, it will provide users with answers to the following two navigation questions:

- Where am I?
- Where can I go from here?

When the information in question already has an inherent spatial nature (e.g. a machine manual (Wilkins & Weal, 1993; Crowder et al, 1993; Hall et al, 1993), model
of a city (Wilkins & Weal, 1993; Day, 1992) or information from an archaeological site (Wilson, 1988; Mylonas & Heath, 1990; Mylonas, 1993; Mylonas, 1993b)) it can be used as the structure of the overview. For example, the machine manual information can be attached to the related parts of a model of the machine. These types of overview are very effective since users can easily locate themselves within the information as it is represented by a real world object. However, not all information has such an underlying world metaphor. In addition, the overviews become difficult to update and are impossible to construct automatically, because they are created by the user rather than the system.

Information that is hierarchical in nature can be represented by a hierarchical overview. Such overviews are possible to construct automatically and users are familiar with their structure. However, they are only good at displaying links within the branches of the hierarchical tree. If links need to be shown that cross from branch to branch then problems will arise. The hierarchical structure quickly becomes an arbitrary graph and the hierarchical overview will no longer suffice.

If the information is neither hierarchical nor representable by a real world object then there are two alternative mechanisms that can be used to construct the overview. These are:

**User constructed abstract overview** - As the name suggests, the user (the author in this case) is responsible for creating the overview. These are generated using a drawing package, each object in the overview will usually represent a group of documents, not a single document. The creation of the overview is very labour intensive and requires a good knowledge of the information held in the document set. Furthermore, links between documents are depicted by the spatial positioning of the document's representations. In some situations the author will have already created some form of abstract overview, for example during the original design of the hypermedia network.

**System generated overview** - If the information has no inherent structure, and the author does not wish to generate an overview by hand, then the only remaining alternative is to allow the system to generate the overview. Each document in the document set is assigned a position in a virtual space. The algorithm for generating these positions, and the dimensionality of the space varies from system to system. No perfect positioning algorithm exists. As a
result, when the overview has to display a large number of links and documents it will become too confusing to be of any use.

Overviews attempt to make use of a user's natural spatial awareness, allowing them to recognise clusters of documents not just single objects. In the case of the system generated overview, each document is represented by a uniquely identifiable object, for example, an icon to represent its document type and a label to display its document title. Links are usually represented by lines drawn between these document objects.

The size of the document object makes the representation of point-to-point links impossible. As a result all lines on the overview represent document-to-document links. Furthermore, if more than one link exists between documents A and B they are only represented on the overview by one line. The removal of such duplication serves to greatly reduce the complexity of the overview. The decrease in link granularity allows the use of abstract icons to represent the documents. If this was not the case then the icons would have to represent the inner structure of the documents.

The amount of the information network that is represented on an overview is called its scope (Utting & Yankelovich, 1990). The following sections discuss the possible scope values in more detail.

2.3.2.1. Local scope

These overviews display one specific focal document and all the documents connected to it. They are simple to construct and provide the user with a localised representation of the information network. However, they do not provide them with a global sense of position or the overall size and complexity of the complete information network. The local map filter within the Microcosm system (see section 4.3) is illustrated below as an example of this type of overview (see figure 2.6). Further examples of such local overviews can also be seen in the Intermedia system (Yankelovich et al, 1988; Haan et al, 1992), and the Stackmaker authoring system (Hutchings et al, 1992).
2.3.2.2. Global scope

These overviews attempt to display the complete information network (e.g. every document and link). These are sufficient for small information networks, say tens of documents and links. However, as soon as the number of documents or links increases the overview becomes too complex to be of use. The global map from an information network on cell biology (Hall et al, 1989; Hutchings et al, 1992) is included below to illustrate this type of overview (see figure 2.7). Further examples of such global overviews were implemented in the original versions of the Intermedia system (Conklin, 1987), but were later removed from the system due to their inadequacies with respect to representing large complex hypermedia networks (Utting & Yankelovich, 1990).
The global overview shown above (see figure 2.7) does not display all links within the hypermedia network that it represents. Displaying all links would result in the complexity of the overview being too great, confusing the users instead of helping them.

The links within the above network are typed by hand, sub-dividing them into primary and secondary categories. It is only the primary links that are shown on the overview. This reduces the average length of the lines used to represent the links and diminishes the complexity of the node layout problem (Conklin, 1987, Utting &
Yankelovich, 1990). In addition, each document within the network appears only once within the overview and the document at its centre (an introduction to cell biology) is chosen by the author.

2.3.2.3. Dynamic scope

A third option, is to allow the user (or the system) to dynamically change the scope of the overview. For example the THOTH II system (Collier, 1987) included overviews that start off with local scope. As the user moves around the information network the overview expands to include the new documents. If the user moves to every document then the overview will be equivalent to a global representation of the complete information network.

The previous sections dealt with variations in one particular feature of overview implementation, namely the scope. The following sections discuss further implementation issues of overview navigation tools.

2.3.2.4. User customisation

One further variation on the system generated overview is to allow the system to initially position the nodes on the overview. Once this has been done, the user is allowed to move the nodes around. Such customisation has two effects, firstly the user can correct any faults in the system layout of the overview. Secondly, users will be able to remember the structure of the overview if they have helped create it. This customisation can be taken one step further by allowing the user to annotate the overviews.

2.3.2.5. Dimensionality

Overviews can be constructed that utilise varying degrees of dimensionality. For example, a timeline overview (Haan et al, 1992) uses just one dimension to present relationships between documents in the information network. This timeline does not represent the links between the documents, but it shows the relationships by positioning them on an axis representing a period of time.

The most popular type of overview uses two dimensions, for example both the global and local overviews discussed earlier use this level of dimensionality. With the increase in computing power, and the advent of such technologies as virtual
reality, comes a move towards three dimensional overviews. An early example of this can be seen in the Semnet system (Fairchild et al, 1988).

2.3.2.6. Complexity reduction

Overviews can suffer from being overly complex. If users cannot easily ascertain the structure that the overview is portraying then it has failed in its primary task of providing the users with spatial context. Overviews become complex when either the number of links or documents represented exceeds an implementation dependent threshold.

The answer to avoiding such complexity is to reduce the number of links or documents. However, such a reduction has to be achieved without sacrificing the functionality of the overview. Furthermore, the simplification must not reduce the overview's capability to provide users with the spatial context which they need.

A variety of mechanisms have been used in order to reduce the overall complexity of the overview tool, some of these are:

**Link and document filtering** - Links and documents are removed from the overview by filtering out those that do not match a specific set of criteria. A simple example of this can be seen in the global overview presented in figure 2.7. The links were sub-divided into two classes: primary and secondary. It is only the primary links which were drawn on the overview. Other systems which employ such filtering are the Notecards system (Halasz, 1988) and Neptune (Delisle & Schwartz, 1986). All links and nodes within Notecards are typed and users are able to specify which types to include in the overview when it is generated. The overviews in the Neptune system are always constructed as a result of a query. Thus, if the query is broad enough the overview will show the entire network. If the query is well defined by the user, then the overview is likely to be sufficiently simple, allowing them to make full use of its functionality.

**Fisheye views** - A fisheye view is a view that provides more detail on the focal object than on the ones surrounding it. In addition, as one, moves away from this focal document the detail is gradually reduced. The idea is to generate a full global overview, reducing its complexity by presenting it to the user via
a fisheye view. Examples of such overviews can be seen in (Furnas, 1986; Noik, 1993).

**Structural analysis** - The hypermedia structure is analysed before it is represented on the overview. Such methods as *hierarchisation* and *cluster identification* (Botafogo & Mosse, 1992; Botafogo et al, 1992) or *aggregate identification* (Botafogo & Shneiderman, 1991) can be used to reduce the complexity of the structure before the overview is generated. Therefore, the overview presents users with a portrayal of the reduced complexity structure as opposed to the original complex structure.

The above three mechanisms show the variation in the possible approaches to complexity reduction that have been implemented. Furthermore, there is no reason why combinations of these approaches cannot be implemented reducing the complexity even further.

### 2.3.3. Histories

As users move from document to document within the information network they will lose track of their temporal context; they will forget which documents they have already seen, and when they saw them. Furthermore, if they cannot remember which documents they have seen then they cannot decide which ones to see next. A history is a mechanism that allows the system to provide users with this temporal context. Two mechanisms have been implemented to act as histories these are:

**Backtracking** - This allows users to step back through the previous documents, one document at a time. However, this type of history mechanism can cause its own type of disorientation. There is no corresponding *next* operation to counteract the step back action. As a result users can travel back through the documents only to lose track of which document they originally stepped back from. Examples of this type of history can be seen in the following systems: FRESS (Van Dam, 1988), Guide (Brown, 1987; Brown, 1990; Fersko-Wiess, 1991; OWL, 1987; Fiderio, 1988; Kahn, 1987; Swarbrick, 1990) or HyperCard (Kahn, 1987).

**History list** - This provides users with a list of documents that they have already seen. They are free to select any of the documents from this list to view again. As users are always presented with a list of documents, and are able
to select any document to view, they will not suffer the disorientation inherent in the backtracking mechanism. HyperCard extends this concept by showing users thumbnail representations of the last forty-two cards visited, called recent. This takes advantage of the user's visual memory, it is hoped that even though they cannot remember the name of the card they will be able to remember what it looked like.

The HyperCard recent mechanism does not show more than one thumbnail for each card. For example, if the user moves to cards in the following order: A, B, C, B, D, the recent mechanism will show cards in the following order: A, B, C, D. As such it does not show users a strictly historical trace of their movements, instead it is simplified to remove duplication.

Finally, a simple implementation of a history can be seen in the Hyerge system (Bernstein, 1988). This places a mark on each document as it is seen (called bread crumbs). However, this does not help the user significantly as they must open the document again in order to determine whether it has already been viewed.

2.3.4. Tours

Guided tours, or paths, have been part of the hypertext vision from its onset. The Memex system (Bush, 1945) was designed to include paths called trails. Authors would blaze trails through the information maintained by the Memex. Furthermore, it was envisaged that the authoring of such trails would become an industry in its own right.

A path is defined as (Zellweger, 1989):

"... a presentation of successive entries, ordered such that all of the decisions about the order of presentation are made by the author in advance, rather than by the reader during path playback."

An important component of the information conveyed by an author to a reader in a traditional setting is the order in which the material appears. In most current hypertext systems, readers may fail to understand the material presented because they view it in the wrong order, or more precisely, in the wrong context. However, paths allow authors to determine an appropriate order of presentation for a given audience.
A path should be provided as a guideline to the user's exploration of the hypertext network. They should be free to stray from the path, following links or using other navigation tools, if they wish. If users are restricted to following the linearisation of the hypertext network provided by the path then they will gain none of the benefits of placing the information within the hypertext network.

The following sections deal with some of the design and implementation issues raised by the tour as a navigation tool, in more detail.

2.3.4.1. Tour creation

Tours can be created manually by authoring a sequence of links from one document to the next. However, this method is very tedious, time consuming, and due to its complexity, often error prone.

Any system that wishes to support tours should provide some form of tour creation and editing process. Tours can be created using three major mechanisms these are:

Form based - Authors fill in forms, one per node in the tour. The information entered into the form describes not only the document to be used as the tour node but also what to do with it when the tour moves on. Furthermore, in systems such as Microcosm (Fountain et al, 1990; Davis et al, 1992; Hall et al, 1992), the form also contains information about how the tour is to progress (a subject discussed in more detail in the next section).

Scripted - A tour is created by the author writing a script (Zellweger, 1988; Zellweger, 1989) in some structured language. Each statement in this script represents the same information as that entered in each form of the form based method. The script is executed by the tour navigation tool, providing its users with tours.

Translation of user created histories - The third method is to translate a history file that has been created by a user. Users create history files by moving through the hypertext network viewing the documents in the order they require them to be viewed within the tour. They then save the history in a history file for later translation to a tour. The tour system should provide a tool that effects the translation for its author.
Whatever method of tour creation the tour system provides it should also provide tools that allow authors to edit the tours, rearranging, adding or deleting nodes.

2.3.4.2. Tour progression

The tour mechanism can provide different methods for progressing from one node to the next along the length of the tour. For example, the tour system described in section 4.2 allows tour progression to be controlled by any mixture of the following three methods:

**User driven** - The user controls the progression of the tour (e.g. by clicking their mouse on a button). The tour system will wait until the user clicks before progressing through the tour.

**Clock driven** - The progression of the tour is controlled by a clock. The tour will only progress to the next node when a pre-set quantity of time has elapsed.

**System driven** - The progression of the tour is controlled by the system itself. For example if the node in the tour is of a temporal nature (e.g. video or sound) then the end of the node will trigger the progression of the tour. This provides a powerful progression system which allows the tour to progress completely automatically without requiring the calculation of the lengths of the temporal nodes in advance.

Tours can contain mixtures of the above progression methods. The most appropriate method, chosen by the author, will depend on a combination of the node type and intended target user type. For example, if the target user was a complete novice then the clock and system controlled progression methods may be more appropriate.

2.3.4.3. Extensions to the linear tour concept

The basic path contains a linear sequence of nodes, presented one after the other. The progression method that controls the flow from node to node can vary but the nodes will be presented one at a time. However, the concept of the basic path can be extended in the following ways:

**Parallelism** - The tour can contain sections that are presented in parallel. This would allow a diagram and its description to be presented simultaneously.
In the basic tour, either the diagram or description would be presented first followed by the other, but not both at the same time. The author would dictate which nodes are to be presented in parallel at the point when the tour is created.

**Procedural paths** - The tour can contain other tours as nodes. This eases the tour author’s job by supporting modularity and re-use.

**Branching** - The tour can contain points that offer the user choices for different subsections of its content. The user chooses a route at these points and so dictates the overall content of the tour.

**Conditional paths** - The tour contains sections that will be shown only if certain criteria are met. For example, students could be shown more detail than lecturers when following a tour in the context of education.

Tour creation and manipulation tools should allow authors not only to create tours with the above extensions, but also to visualise their structure once created.

2.3.4.4. Tours: a link or document?

Opposing views have been expressed as to whether a tour is a link or a document. For example, Nielsen (Nielsen, 1990) suggests that a tour is really a superlink allowing sequences of documents to be joined together. However, it could also be argued that a tour is in fact a meta-document, allowing documents to be viewed as a simple document containing many parts. The tour system in Microcosm (see section 4.2) provides users with a document viewer as an interface to its tours. Therefore, tours within the Microcosm system are treated as documents not links.

It is true that tours display characteristics of both the link and the document. They are certainly a very important part of the hypermedia paradigm and it has been suggested that they should become first class citizens along with the link and the document (Zellweger, 1989).
2.4. Navigation clues from other sources

It is not only the specifically designed tools, described earlier, that aid users in their navigation through the hypertext work. User's navigation is also affected by the following areas of their computing environments:

**Hierarchical file system** - Users can take advantage of the hierarchical nature of their computer's file system. Providing a one to one relationship exists between documents in the hypertext system and the files used to represent them, then the tree structure common to the majority of file systems can be used as a navigation aid. Authors can group related material together by placing it and the files that represent it into a common directory. Furthermore, the labelled path from the root of the tree to the directory can be used to describe the reason for such grouping.

**Multiple windows** - The ability of the user’s computer display to support multiple, re-sizeable and overlapping windows eases the cognitive load associated with traversing links. Systems that can only present one node at a time force their users to remember the content of the start node until they read the content of the end node. If both the start and the end nodes can be shown simultaneously this burden is removed from the user.

**Spatial layout of document content** - The spatial nature of the representation of the document can act as a visual clue to its actual content. Users will be able to recognise visually distinctive documents with greater ease. Such an effect is put to use in the recent mechanism provided by the HyperCard application discussed earlier.

**Granularity of the hypertext network** - The size of the documents within the hypertext network can also be a contributing factor to the problem of user disorientation. If the size is too small then the hypertext will seem fragmented and users will need to travel a long way to absorb a small quantity of information. However, if the size is too large users will run the risk of being lost inside single documents. In addition, links into the middle of documents will become difficult to create as the sections linked to will...
often utilise context derived from previous sections within the same document.

As the above list illustrates, it is not simply the navigation tools of the hypertext system that provide a complete solution to the navigation problems suffered by its users. In reality, if a solution exists, it will be due to contributions from the user's complete computing environment and not simply the small set of navigation tools provided by the hypertext system.

2.5. Summary

Linear text provides its readers with a plethora of cognitive navigation clues constructed over the vast period of time that it has been in use. Placing such text within a hypertext system leads to navigation problems due to the de-linearisation of its content.

The cognitive clues provided by mechanisms such as page numbers, chapter headings and the physical location of the current page within the body of the book are lost when the book's content is introduced into a hypertext system. Navigation tools are presented as a mechanism to regain these cognitive navigation clues.

Navigation tools help the user to answer the three classic navigation questions, these are:

- Where am I?
- How did I get here?
- How do I get from here to my next destination?

Navigation tools can be sub-divided into four basic categories:

**Links** - one of the fundamental building blocks within all hypermedia systems.

**Overviews** - provide users with pictorial representations of the information network contained within the hypermedia system.
Histories - provide users with a list of their previous movements within the hypermedia information (or document set).

Tours - provide users with a trail through the information network. The majority of the user's navigation problems are removed whilst they are following such a trail.

However, it is not simply the navigation tools, classified by the above categories, which aid a user's navigation through the hypertext network. Users are also helped in their navigation by the hierarchical file structure that underlies most hypertext networks, the use of multiple windows, spatial layout of a document's content and lastly by the granularity of the hypertext network itself.
3. Microcosm: a model for open hypermedia

This chapter begins by outlining the reasons behind the design of the Microcosm open hypermedia system. It then continues by describing the design and implementation of the original system, Microcosm V1.0. The problems, and associated solutions, contained within this implementation are discussed.

Finally, the chapter concludes with a description of the current system, Microcosm V2.0, highlighting the main differences between it and V1.0. The description focuses on the advantages and disadvantages of implementing navigation tools as filters within the system.

3.1. The design of microcosm

The key motivation behind the Microcosm project is to produce a fully open and user-configurable workbench for research into hypermedia. Research into the desirable features of such a system identified the following problems within the majority of existing hypermedia systems:

**Authoring effort** - In the majority of hypermedia systems every link has to be hand authored. This can be a trying and laborious task at the best of times. As the size and complexity of the hypermedia network increases the task will become a lot less manageable.

**Closed systems** - The majority of current systems exist as stand alone applications. In some cases they allow uni-directional communication between themselves and other applications. The addition of new media or extra functionality is made very difficult by such isolation.

**Proprietary document formats** - Documents produced in one application are not easily used in others.

**Embedded links** - The majority of systems need to embed linking information into the documents. This will be impossible in areas where the documents are held in a read only form (e.g. CD ROM storage).
Although it is true that not all systems suffer from the points outlined above, the aim of Microcosm is to address them all. In addition, the following principles were used to further guide the design:

**No distinction between author and user** - Both would have equal access to the full functionality of the system (e.g. both would be able to create links). This would allow a large part of the modality of the system to be removed. Tools would present the same interface to all users of the system, thus simplifying their construction.

**Loosely coupled system with minimum interdependencies** - The system would be constructed from a set of communicating tasks, allowing it to be easily analysed and tailored to suit the needs of each individual.

**Separation of links from the documents** - The links would be kept separately from the documents. This would allow them to exist at same level as the documents, opening the way for easy analysis and manipulation.

Research into the desirable features of an open hypermedia system based on the principles outlined above involved a survey and analysis of existing hypertext/hypermedia systems such as Guide (Brown, 1987), Notecards (Halasz, 1988), Intermedia (Yankelovich et al, 1988; Haan et al, 1992), KMS (Aksryn et al, 1988) and Sun's Link Service (Pearl, 1989).

In addition to the three principles described above, Microcosm's design was driven by the use of a selection/action model as the primary means of user interaction. For example, the user selects an area of a document (e.g. words in a text document, or an area of an image) and then chooses which action to perform upon that selection. These actions may vary from simply following a link to performing some form of image processing.

Within this model the following of a link is viewed at the same level as any other action. The user makes a selection and instructs the system to follow a link from it (if one exists). Users are able to create links in the same way: they make a selection and tell the system to create a link instead of follow one. The button (common to the majority of hypertext systems) is modelled purely by the binding of a particular selection with the action of link following. New functionality can be added to the system by adding new actions.
Microcosm links are stored in separate link databases (linkbases) and all link following is resolved by a plain database query. In this scenario a button becomes a pre-defined linkbase query. From the user's point of view a button in the Microcosm system has the same functionality as in any hypertext system.

Users are not restricted to selecting the buttons from within a document. They are perfectly free to make any selection. If the follow link action is then chosen the system will still perform a linkbase query. In Microcosm a selection is defined by three fields:

- The document name
- The location of the selection within the document
- The selection itself

In the case of a button all three fields must be matched in order to locate the associated link. If, however, the position in the document is relaxed then the link can start anywhere in the named document that matches the selection (called a local link as the anchor is local to the document). If we relax the name of the document as well then the link can start anywhere in any document that matches the selection (called a generic link). Resolving links by executing a linkbase query allows the author to use implicit links. The advantages and possible side-effects of such an implicit linking mechanism are discussed in section 2.3.1.2.

A full explanation of the system objectives and the solutions to the problems raised by the principles outlined above was first presented in (Fountain et al, 1990).

3.2. Microcosm V1.0

The Microcosm design was aimed at separating the displaying of documents and the processing needed to perform any actions on them (e.g. pre-indexing for full text retrieval). With this in mind, Microcosm V1.0 was constructed from two main modules, the Document Control System (DCS) and the document viewers. The DCS was responsible for controlling the document viewers, routing messages between the document viewers and providing the user with a centralised linking
The document viewers were responsible for displaying documents and allowing the user to interact indirectly with the centralised linking mechanism within the DCS. Each document viewer was specifically designed to display and manipulate a specific type of document.

Links were stored in a flat-file database, with each link represented by a single record. When the database received a message telling it to follow a link (usually generated by the user highlighting a button) it attempted to match one of its records with the details in the message. If a match was found then a message was sent to the DCS telling it to open a document in the appropriate document viewer. Users would see a document appear and perceive that this was due to a link being followed. They need not know what had occurred in the underlying system.

The document viewers and the DCS communicated with each other via a system of software channels and messages. These messages were carried by the Dynamic Data Exchange (DDE) protocols provided by Microsoft Windows. A library of channel functions was created to hide the use of the DDE from the various parts of the system. This had the advantage that the underlying communication mechanism could be changed without changing the applications themselves. The model of Microcosm V1.0 is shown below in figure 3.1.

![Figure 3.1: The model for Microcosm V1.0](image)

Points of interest in figure 3.1 are the multiple document viewers (on the left of the diagram) and the multiple actions (embedded within the DCS) in the middle of the diagram. The design decision to embed the action management code into the DCS proved to be wrong. The centralised linking mechanism provided by the linkbase, embedded in the DCS, presented a fixed functionality and proved fairly limited.
The problems caused by this and other such decisions are discussed in the following sections.

3.2.1. Fixed message system

Microcosm V1.0 messages were of a fixed nature. Each contained fields for the description of the source of a link, together with the link selection and the action to be carried out. This fixed nature meant that the system itself would need to be changed every time information had to be added to the message structure. This was deemed undesirable and so a more flexible message format was designed.

A totally flexible message format, capable of storing any set of fields required and being of any size, was developed. In addition a Dynamic Link Library (DLL) was constructed that allowed the applications to construct, interpret and manipulate these new messages. The DLL also had the effect of providing a consistent interface, hiding the underlying message format and transport mechanism.

3.2.2. Abstraction of document types

Microcosm V1.0 used the filename extension of a document file to determine what type of document was being used. This method worked for the IBM-PC implementation but caused problems when the system was moved to other filing systems. In file systems such as UNIX the user is not restricted to naming files so that they have a three letter file extension which denotes their file type.

The solution was to hold the type of document that was at the source and destination of the link within the link itself. It was passed back to the system when a link was followed, to determine the correct viewer to use.

3.2.3. Limited action configurability

Microcosm V1.0 had all action processing (e.g. following links) carried out in the DCS. This produced problems as, from the user's point of view, the functionality of the system was fixed.

The solution came from modelling the action processing on the document control side of the DCS. In this, we have a set of viewers (each one dedicated to a
particular task of displaying and manipulating a specific document type). The DCS manages these viewers and maintains the inter-viewer communication. This design was adapted to allow the action processing to take place in the same manner. Each action would be handled by a separate process and some action management system would look after these processes.

The action processors would be able to create new messages, allow existing messages to pass through, or be stopped. The ability of an action processor to dictate whether a message is stopped or not imitates the behaviour of a filter, and this was the name given to them. Thus, Microcosm V2.0 was split into two co-operating process groups:

**Document Control System (DCS)** - Which is responsible for managing the document viewers, making up the front end (same as V1.0).

**Filter Management System (FMS)** - Which is responsible for managing all the action processing, making up the back end (new to V2.0).

Both process groups are discussed in greater detail in the following sections. In addition, the communication mechanism that allows the two groups to communicate is also discussed.

### 3.3. Microcosm V2.0

Microcosm V2.0 is the result of a substantial re-design of the original system, Microcosm V1.0. The problems outlined above were solved and the solutions implemented to form the current system. From now on within this thesis the term Microcosm will be used to denote Microcosm V2.0. Any references to previous or future versions of the system will be made using the full system name (e.g. Microcosm V1.0).

Microcosm is a distributed open hypermedia system (Fountain et al, 1990; Davis et al, 1992) designed to support the integration of many different applications. It exists as a set of co-operating processes. It is from these separate entities working together that the power, and the open ended functionality of the system are drawn.
Currently these processes are sub-divided into two classes, the viewers and the filters. The viewers allow the user to interact with the information within the system and the filters provide a variety of functions ranging from navigation tools to link construction and storage tools. The current model can be seen in figure 3.2.

![Diagram of the logical model of Microcosm](image)

Figure 3.2: The logical model of Microcosm

The above diagram (see figure 3.2) illustrates the filter chain; the mechanism that allows each filter within Microcosm to communicate with the other processes in the system. The chain allows communication in one direction, allowing filters to send messages for later filters but not for earlier ones. Such a system was designed to allow the filters to carry out their primary task of filtering messages travelling through the message system. The disadvantages of such a design are discussed in more detail in section 7.1.

Each process in the Microcosm system can communicate with other processes through a custom\(^1\) messaging mechanism. Unlike the messages in Microcosm V1.0, these contain plain text subdivided into tagged sections. This message format has the advantage that new tags can be added at any time. If a filter receives a message with tags it does not recognise then they will be ignored. The plain text format and easy extendibility mean that the message system is open. The filters can perform one of three actions on a message:

---

\(^1\)Microcosm uses its own message passing system, as the message passing system within Microsoft Windows is not flexible enough for the requirements of the system.
Block - The message is removed from the communication system and no further processes will receive it. For example, a link database will block a message generated when the user wishes to create a link once it has been created.

Modify - The contents of the message are altered. All other processes will receive the new message as the original message has been replaced.

Pass - The default is to pass the message on to the next filter in the chain, without changing it in any way. The remaining processes will receive the original message. In practice, the message is actually passed back to the FMS which then passes it on to the next filter. This procedure has been simplified within the diagram in figure 3.2 to show one filter passing a message directly to another.

In addition to the three actions listed above a filter is free to generate new messages. These are usually produced as a side effect of processing the current message. All message creation and interpretation functions are contained within a single DLL, enabling any program to become a filter by including a few additional lines of code. More detail on the message system employed by Microcosm can be seen in (Fountain et al, 1990; Hall et al, 1992; Heath, 1992; Wilkins et al, 1993; Hill, 1994).

The FMS is now entirely customisable, both the user and the system are completely free to stop any of the currently running filters or add new ones into any point on the filter chain (see section 3.3.4). In addition, because the filters are arranged in a sequential chain, the user is able to re-order them if necessary.

The DCS, the other half of the system, still handles all interaction between the document viewers. However, it now passes all other actions (e.g. locate buttons) on to the back end of the system since it is no longer responsible for their processing. The majority of document viewers have been written as part of the Microcosm system. However, in order to keep the front end of the system open, any application can be used as a Microcosm viewer. Viewers can be subdivided into three categories:

Fully aware - These are viewers that have been written explicitly to communicate with the DCS. A bi-directional message channel exists between the viewer and the DCS. This allows the viewer to issue requests to the linkbases for
buttons etc. In addition, it is able to receive replies (for example, indicating the location of any buttons).

**Partially aware** - These are usually applications that can be tailored by the user, for example the Guide system (Brown, 1987) and Microsoft Word for Windows (a word processor within the Microsoft Windows environment). The DCS is able to control which documents are opened by the application. In addition the user is able to follow, or create, generic links from within them. The viewers cannot display, or create, buttons as the application is not capable of full communication with the DCS.

**Unaware** - These are applications that have no direct communication with the DCS. In some cases they can be launched to view the correct document. If this is not the case then the user has to select the document for the application to display. The user is able to follow generic links by using the clipboard (a mechanism for transferring data from one application to another) as a communication channel between the application and the DCS.

The ability to communicate with applications that are completely unaware of the existence of Microcosm effectively integrates the hypermedia functionality into the user’s everyday environment. This Microcosm feature and the lack of link anchor embedding are two of the key features that enable the system to remain open.

By implementing filters and document viewers as separate tasks, it is possible for both developers and users quickly and easily to change or extend both the functionality and document types available. Users can update their personal configuration to use any new features, whilst the developer does not require any recompilation of existing code to provide new functionality. The new task is simply required to adhere to the communication protocols defined for a viewer or filter.

In addition, the fact that both the DCS and FMS are also separate tasks means that they can also be easily modified to experiment with different aspects of the

---

2As an aside, Microcosm itself does not need to understand the format of the document. The displaying application takes over the responsibility for translating any selections made by the user into forms understandable by Microcosm.
system. For example, only the FMS needs to be altered in order to vary the way filters are managed. The remainder of the system need not be changed.

This filter-based approach allows maximum flexibility in the configuration and future development of hypermedia systems based on the Microcosm model. Further details of the design and implementation of Microcosm can be found in (Fountain et al, 1990; Heath, 1992; Hill et al, 1992; Wilkins et al, 1993).

The remainder of this chapter discusses the areas of the Microcosm model that have the greatest effect on the implementation of navigation tools as filters in the Microcosm system.

3.3.1. Communication

The overall efficiency and flexibility of the communication system has a dramatic effect on the speed, integration and flexibility of processes (e.g. the navigation tools and agents) that exist within it. In order to fully understand the environment in which these processes exist we must first examine the current communication system. Such an examination highlights the areas which have the greatest impact on the processes. Later chapters within this thesis will provide solutions to the problems highlighted within the following sections.

The diagram below (figure 3.3) shows the conceptual communication channels which exist between the processes. The processes are segregated into two classes, the viewers and the filters. Each class has its own control process; the Document Control System (DCS) for the viewers and the Filter Management System (FMS) for the filters. The two features which slow down inter-process communication can be clearly seen; the communication bottleneck between the DCS and FMS, and the unidirectional filter chain. Both features are discussed in greater detail in the following sections.
In contrast, figure 3.4 shows the actual channels which exist. The topological differences between the two diagrams are due to the two routing processes, which dictate the distribution of each message. Four different message pathways exist, depending on the locations, and classes, of the communicating processes. These are discussed in more detail in the sections below.

The current system contains four different communication scenarios, which are due to the filter/viewer process distinction (the two classes of process imply four possible combinations). These are:

- Viewer to filter.
- Filter to viewer.
• Filter to filter.

• Viewer to viewer.

The sections below describe the routes through which messages must currently travel (and the problems with those routes) for each scenario. The efficiency of these communication routes has a direct effect on the efficiency and flexibility of the entire Microcosm system.

The sending and receiving processes in the diagrams are marked with bold outlines. The solid arrows show the path which a message has to travel if there is a receiving process. If there is no receptor, the extra path along which the message travels, before it is deleted, is shown by the dashed arrows. The following four diagrams consider a configuration containing two filter processes. The addition of more filter processes will result in longer message pathways. This will have the most impact on the message pathway length in the viewer to viewer scenario (see section 3.3.1.4). Longer message pathways effectively slow down any communication. As the complexity and integration of the navigation tools increases this speed will become more important.

3.3.1.1. Viewer to filter Scenario

When a viewer process wishes to communicate with a filter process it must send a message to the DCS. The message is then sent to the FMS and on down the filter chain until it reaches the appropriate filter process. This route is shown in the diagram below (figure 3.5).

![Diagram](image)

Figure 3.5: Route for a message from a viewer to a filter.

If there is no receiving process for the message, upon reaching the end of the chain it will be sent back to the DCS. The DCS will then send the message to the viewer process that originated the message. As the viewer process was sending the message to a filter process the latter half of the message path is always redundant.
For example when a document is opened within Microcosm, the viewer process sends out a request for any links that need to be displayed. This request is received by all linkbase processes within the system.

3.3.1.2. Filter to viewer Scenario

When a filter process wishes to send a message to a viewer process it must first pass through the remainder of the filter chain. The FMS then sends the message on to the DCS, which attempts to send the message to a viewer process. This route is shown below (figure 3.6).

![Figure 3.6: Route for a message from a filter to a viewer.](image)

The DCS determines which viewer process the message will go to by examining its contents. If the DCS is unable to do this then the message is deleted. In the current system there is no explicit method for uniquely identifying a viewer process which greatly increases the complexity of this scenario.

For example when a linkbase receives a request for links from a viewer process it responds, sending back one message for each link found. The linkbase is able to extract the identity of the viewer process from the original link request message. This identity is used to route the link messages back to the correct viewer.

3.3.1.3. Filter to filter Scenario

When one filter process wishes to communicate with another it sends the message to the next filter process in the chain and so on until it arrives at the receiving process. This route is shown in the diagram below (figure 3.7).
If the receiving filter process is not later in the chain then the message will follow the same path as an unwanted message in the viewer to filter scenario (see section 3.3.1.1).

For example when a linkbase receives a message to search for a set of links which match a specific selection it replies by sending off a set of messages. These messages are received by another filter process which displays them as a list of destinations for the user to choose.

3.3.1.4. Viewer to viewer Scenario

If a viewer process wishes to send a message to another process of the same class then the message has to go to the DCS. From there it goes to the FMS and along the complete filter chain. It is then sent back to the DCS and finally to the receiving process. The message route is shown in the diagram below (figure 3.8).

The message has to travel through the filter chain because the DCS has no mechanism for directly routing it to the other viewer process. This means that the message has to travel along a much longer path than is necessary.

The current Microcosm system contains no examples of communication between viewer processes. It is envisaged that such communication will become necessary if the system is to allow for viewer processes to be synchronised.
3.3.1.5. Scalability of the segregated communication model

The current communication model contains two features that will restrict its speed and scalability: a bottleneck between the DCS and FMS, and a serial uni-directional filter chain. Such restrictive features will have a direct effect on the performance of the communication system. This performance will, in turn, affect the speed, scalability and integration of the processes using the communication system. More specifically, as the following chapters will illustrate, the performance and functionality of the communications system will affect any navigation tools or agents that are implemented within it.

Every message from any filter to any viewer must pass through the single communication channel between the DCS and FMS. This bottleneck restricts the speed of the messages passing between processes of differing classes (e.g. filter to viewer and visa-versa).

The majority of the communication overhead in the current system is due to the messages travelling along the filter chain. This is not due to the generation of a large number of messages, but to the serial nature of the chain mechanism. Filter processes must pass on all messages to the next process in the chain.

Consider a multi-tasking environment in which viewer processes can be accurately synchronised. Suppose the user causes five viewer processes to open at the same instant. Upon initialisation those five processes send messages to all linkbases present (e.g. requesting the locations of any text buttons). All of those messages must be funnelled through the channel between the DCS and the FMS. They must then be bounced through all the filter processes to the linkbases themselves. The linkbases process the five messages, producing vast numbers of messages which must be returned to the five viewer processes. The messages produced in parallel must be serially sent down the filter chain, then between the FMS and the DCS. They are only distributed at the last stage when the DCS sends the messages to each of the viewer processes. The net effect of this is to slow down the whole system.

Such slowing down is a direct result of the number of messages generated within the system. The equations shown below can be used to calculate the number of messages generated (in the current system) for the events outlined above.
Equation (3.1a) is the number of messages generated when the $v$ viewer processes send off link requests to the linkbases. Equation (3.1b) is the number of messages generated when the linkbases at consecutive positions $p$, $p+1$, $p+2$ in the filter chain respond with $k$ links. Equation (3.1c) is the total number of messages generated in the above situation.

\[(a) \quad v(2f+3) \quad f = \text{number of filters in filter chain}\]

\[(b) \quad \sum_{p=0}^{q=L-1} v_k (2(f-(p+q))+3) \quad k = \text{number of links returned per linkbase}\]

\[(c) \quad \text{total} = (a) + (b) \quad \text{L} = \text{number of linkbases}\]

Equation 3.1: Calculating no. messages generated in the SCM.

For example, suppose we have ten filter processes (including three linkbases) and five viewer processes in the current system. For simplification we can assume that each viewer process will open the same document and each linkbase will contain the same set of links. The graph (figure 3.9) below shows the number of messages produced in the following situations (in all cases $k=100$ and $L=3$); the number of messages generated against the position of the first linkbase process in the chain.

![Graph showing number of messages against position of first linkbase in chain.](image)

Figure 3.9: Number of messages against position of first linkbase in chain.

It can be clearly seen (from the above graph 3.9) that the number of messages, and hence the performance of the system, is directly related to the positions of the linkbases in the chain. As the linkbases are moved nearer to the end of the chain the number of messages decrease. This is due to the messages they produce having to travel through less filters before they reach their destinations. This analysis indicates
that filters which wish to send large volumes of messages to viewers should be placed at the end of the filter chain. However, if the linkbases (an example of such a filter) are placed at the end of the chain no other filter can examine the messages they send to the viewers. This and other such problems render the implementation of certain navigation tools and agent type processes impossible. Chapter seven discusses an alternative communication system that allows the implementation of such tools.

In the current system there is no concept of direct communication between two viewer processes. For example, if one viewer process wishes to synchronise with another, messages must travel along the path illustrated in section 3.3.1.4. The extra time lag and processing overhead will make synchronisation of viewer processes very difficult.

If one filter process wishes to hold a bi-directional conversation with another, it must physically exist in two positions within the filter chain, since the chain is inherently uni-directional. The first instance must exist in front of the receiving process and the second instance after it. An example of this is an intelligent search engine, which queries the linkbase for a particular set of links, filtering the results before passing them to the rest of the system.

In the current communication model there is no concept of either a secure channel or a broadcast message. A secure channel is a channel along which one process can send a message to another ensuring it is not changed or deleted. A broadcast message is a mechanism that allows one process to send a message to all the other processes in the system. In the current system a broadcast message would be impossible as there is no mechanism for distributing a message to multiple viewer processes. A secure channel is impractical as the filter chain model means that a message must pass through many filter processes along its route. The only way this could be achieved is by ensuring the co-operation of all of those processes. Without this, any process is able to change or delete the message as it passes along the chain.

The various components of the current model can be categorised as either message routers or interpreters. The message routers (DCS and FMS) rigidly define the communication topology. The message interpreters (viewer, and filter, processes) send and receive information down a single channel; they have no understanding of the rest of the system. This separation is advantageous because it
makes the development of message interpreters easy. However, the performance of
the system is heavily dependent on the implementation of the message routers and
the topology of the communication system is static.

In summary, the current communication system contains two features which
will restrict its scalability and speed. These are the filter chain and DCS to FMS
bottleneck. Furthermore, processes using it lack the ability to hold bi-directional
conversations. No single process can broadcast a message to all others, and no
process can send a message to another named process. The need for such features
and a new communication model that provides them is discussed in chapter seven.

3.3.2. Implicit links

An implicit link is a link which is not resolved until the user attempts to follow it
(see section 2.3.1.3). The Microcosm system implements a form of implicit link called
the generic link. All link following within the Microcosm system is achieved by
executing a database query. In the case of a button (explicit link) the following three
fields must match exactly for the link query to be resolved:

- The document name
- The location of the selection within the document
- The selection

However, in the case of a generic link (a type of implicit link) only the selection field
must match. The matching on the other two fields is relaxed. For example, suppose
a generic link has been created from the word 'Southampton' to a map of
Southampton. Users can select the word 'Southampton' anywhere in any text
document. On following the link they will be shown the map.

Such generic links have two main side effects; the link anchors cannot be easily
highlighted for the user, and the navigation tools cannot represent the links without
resolving them first. In order for an overview tool (see section 2.3.2) to present the
user with a complete picture of the hypermedia document (e.g. a global map) it
must carry out a computationally expensive search of the complete set of all implicit
link anchors against the set all relevant documents. If the system is truly open then
how does one define such a set? If the tool is constrained to some subset of
documents then its information will need to be dynamically updated each time the subset is changed. In addition if one of the implicit links is removed, then this expensive search must be carried out once more. Such a situation raises the question of whether the system is still truly open (see section 3.4).

3.3.3. Link separation

No linking information is embedded within the documents, instead it is maintained in separate linkbases. This has a number of beneficial side effects:

**Link information is easily analysed** - As any process that understands the format of the link database is able to access the link information contained in one central location.

**Original documents are not altered** - This allows read only material to be used within the system (e.g. CD ROMs or networked files that the user does not own).

**Documents can be manipulated by their creating applications** - As the format of the documents are not altered they can still be manipulated by the applications that created them.

**Multiple link sets can be applied over one set of hypermedia documents** - Viewed in an educational context, a tutor could provide one set of links that every student would use. Each student is able to create and use their own links which will be stored in a separate linkbase. When the student uses the system they will see both, the tutor's links, and their own, superimposed on the documents.

However, such link separation also has the following disadvantages:

**Inconsistent links** - Storing the links and documents as separate entities makes the current system more susceptible to inconsistency. No mechanism exists to reconcile the links against the documents if either are changed. For
example, a link would be left dangling\(^3\) if the document at either end is deleted. Such a reconciliation mechanism is needed for Microcosm, more specifically the navigation tools, to cope with such inconsistencies, shielding the user whenever possible. To a certain extent, Microcosm avoids the problem by treating documents as 'published' (i.e. they will not be changed or removed once placed in the system).

**Updating local knowledge** - Navigation tools need to be able to modify their local knowledge, reflecting any changes in the hypermedia document set. For example, a local map displays all links into and out of a particular document \(x\). If the hypermedia document is changed so that a document is removed from the hypermedia network then the local map must be able to change to reflect this new situation. In the scenario where links are embedded within the documents the local map only has to examine the documents themselves to detect any changes. If the links are separate, the local map must carry out two processes: extract a set of possible link roots from the linkbases, and examine all the documents to construct a set of all possible anchors. The local map would then display the intersection of these two sets.

**3.3.4. Dynamic re-configurability**

In Microcosm filters exist as autonomous processes centrally controlled by the FMS. The user is able to radically change the functionality of the system by simply interacting with the FMS. For instance, if the user wishes to use the local map filter or the history filter (see chapter four) they simply insert them into the current filter chain using the FMS. This ability to dictate which filters are running at any time is very powerful.

All navigation tools require a certain level of operating continuity. For example, if users plug in the history filter half way through a session it will have no knowledge of their previous navigational movements. Should they then attempt to retrace their steps through the hypermedia document set, they will soon discover

\(^3\)If either end of the link is removed then the link can be thought of as dangling, it leads to or from nowhere.
that they cannot do so. As far as the history filter is concerned they have not viewed any documents.

Suppose we have a navigation tool such as the active path guide (see section 5.4) which needs to constantly monitor all user movement. If the user unplugs this tool then its local knowledge of user movement will become incomplete and fragmented. The longer that it remains unplugged the larger the disparity between its knowledge and the real events will become. The advice that such a guide tool gives to the user is derived directly from its local knowledge. As a consequence the advice supplied to the user will become increasingly inaccurate.

Such continuity problems can be avoided, to a certain extent, by ensuring that the users have a full understanding of each navigation tool's needs, and restricting such tailorability to system administrators.

3.3.5. Precedence of settings

The FMS is responsible for starting up the filters at the beginning of a session. Microcosm uses a series of initialisation files (ini. files) to store static system settings from session to session. There are three levels of ini files, System - every application and user will see any settings from this level, Application - every user will see any settings from this level for the current application, User - any settings from this level will only be seen by the current user in the current application. Figure 3.10 below shows these layers and how they relate to each other.

![Layered architecture of Microcosm ini files.](image)

Figure 3.10 shows that any entry in the user ini. file will override the same entry in either the application or system ini. files. In the same way any entry in the application ini file will override the same entry in the system ini. file.

\[\text{Overriding order, read order}\]

4The active path guide is a navigation tool that is able to help the current user navigate through hypermedia document sets by using its knowledge of previous users' movements.
Many filters store their current state in the ini. files. Users can change this state by interacting with the filter and changing a variety of option settings. For example one user might want the history filter to save their history at the end of a session and another might not. Each is free to change the behaviour of the history filter to suit their needs. All users who do not change any of a filter's settings will use that filter with the settings defined at an application or system level. If they do change one of these settings then they will use that filter with the settings defined at their user level.

These layered ini. file settings can have undesired side effects in certain situations. Consider the scenario where a local map filter has been started and contains an application wide view of the hypermedia document. Every user who uses this application will share the same map. Links discovered by any user will be placed on the local map\(^5\) (see section 4.3). As a result the map will quickly accumulate links. Suppose a user sets up the local map at their user level. When they start Microcosm the FMS would start their private map instead of the application map. This would have two side effects; the user would only see links on the local map that they have discovered, and the user would not contribute to the application map. The net effect would be that the application map would take longer to accumulate the same number of links. It would be updated by all users except the user with the private map.

### 3.3.6 Viewer and filter segregation

All processes within the Microcosm model are divided into two classes, viewers or filters. This segregation is continued throughout all areas of the model. For example, processes of differing class cannot directly communicate with each other, as their conversations are routed through both the DCS and the FMS. This division has the advantage that the processes are optimised for their particular areas of functionality. The viewers are dedicated to displaying documents and handling user interaction, whilst the filters are dedicated to action processing.

\(^5\)A local map is a navigation tool that provides users with a graphical representation of a specific document and all documents linked to or from that document.
This division has several disadvantages in the area of navigation tool implementation. Navigation tools such as the guided tour (mimic), history and the local map (see section 4.3) should ideally be implemented as viewers. They should all be accessible through the standard document selection interface provided by the DCS, as the user interacts directly with them. In order for this to happen the tools would have to be implemented as document viewers. However, if this is done then they would not have access to the vital flow of information present within the filter chain.

This problem is best illustrated by looking at the implementation of the mimic system (see section 4.2). This is constructed from a mimic viewer and a mimic filter. The viewer provides a front-end to the system and the filter the back-end. The system is shown below in figure 3.11.

The mimic filter is responsible for both monitoring the message system, and issuing messages to control the progress of the mimic. The two halves of the system communicate using a set of private messages that nothing else in the system currently understands. The mimic system is divided into two processes as it needs both the filter functionality and the interface provided by the viewer. The mimic system could be implemented as one process if the segregation of the system processes was removed. The implementation of the system is described in more detail in section 4.2.
3.4. Summary

This chapter has shown that using filters within the Microcosm system to implement navigation tools has definite benefits. The filters are able to access all the messages travelling through the system, allowing them to utilise existing functionality within it. They can be designed and implemented in a very modular fashion. Users are able to add or remove filters at any time, allowing the system to be easily tailored to their needs. The flexible message system allows new functionality to be added without changing any of the existing system.

However, using filters as navigation tools also has several disadvantages. The message system is relatively inefficient and contains no concept of secure or broadcast communication. In addition, a filter has no control over the destination of the messages it sends. If one filter wishes to interact with others as would be necessary for an Advisor agent\(^6\) (see chapter eight), then the current message system will be inadequate.

The fact that the links are held separately from the documents themselves complicates the construction, maintenance and consistency of a filter's local knowledge\(^7\). Furthermore the system's ability to support implicit links makes the generation of a structural representation of the hypermedia document set computationally expensive.

The fact that the user is able to start or stop filters at any time may lead to its local knowledge becoming fragmented. Finally, the segregation of the model means that a navigation tool cannot have the functionality of both a viewer and a filter and remain a single process. In addition a navigation tool's local knowledge (e.g. the current user history) cannot be treated as a document by the rest of the system.

---

\(^6\)The Advisor agent is a process that interacts with the navigation tools, on the user's behalf, to construct an overview of the available navigation information.

\(^7\)Local knowledge is the private information that a process maintains about the state of the system and its model of the user's behaviour.
because it is managed by a filter. These implementation issues are discussed in more
detail in chapter four.
4. Implementing traditional navigation tools using filters

This chapter begins by describing the implementation by the author of a set of traditional navigation tools within Microcosm, comprising a history filter, guided tour system and a local map filter. Descriptions and discussions of similar navigation tools can be seen in chapter two.

The traditional tools are those which are widely implemented in other systems. Much discussion about each type of tool in this section can be seen in the current hypermedia literature (Bernstein, 1988; Conklin, 1987; Gloor, 1991; Haan et al, 1992; Hutchings et al, 1992; Utting & Yankelovich, 1990; Marshall & Irish, 1989; Nielsen, 1990; Nielsen, 1990b; Trigg, 1988; Yankelovich et al, 1988b; Zellweger, 1988; Zellweger, 1989) and in chapter two. This chapter focuses on the aspects of their implementation that are Microcosm specific, providing solutions to some of the problems caused by both, the communication model and the flexibility that the Microcosm modular approach offers. The chapter is concluded by a summary of its key points.

4.1. History filter

The primary function of the history filter in Microcosm is to provide the user with the ability to retrace their steps through the hypermedia document set. In order to achieve this it must be able to do two things:

- Receive Microcosm messages containing information about documents that are opened by the user.
- Send messages to the rest of Microcosm requesting documents to be opened.

When Microcosm displays a document (as the result of an action from the user or some part of the system) messages are sent between various processes along the communication channels. These messages originate from the DCS and propagate through all the currently executing filters. When the messages reach the end of the
chain they are returned to the DCS, where they are deleted (more detail was given in section 3.3.1).

The history filter monitors the communication system looking for a specific message called 'IS.OPENING'. Every document opened causes such a message to be transmitted through the system. Figure 4.1 shows the partitioned contents of one of these messages. The history filter records the DoctName, DoctType, Offset and Description tags, holding them in an internal data structure.

```
\Action IS.OPENING
\DoctName 400.04.02.92.16.39.18
\DoctType TEXT
\Offset 0
\UniqueDocID 2
\Description Cell Biology Introduction.
\OpenedBy Memory
```

**Figure 4.1 : Microcosm 'IS.OPENING' message**

The history filter, at the request of the user, will extract the information from the internal data structure and create a Microcosm 'AUTO.DISPATCH' message shown in figure 4.2. This message will originate from the history filter, pass through the remaining filters in the chain, and finally arrive at the DCS. The DCS then interprets the message, determines which document has been requested and opens it.

```
\Action AUTO DISPATCH
\DoctName 400.04.02.92.17.16.01
\DoctType TEXT
\Offset 0
\Description Cell Biology Introduction.
```

**Figure 4.2 : Microcosm 'AUTO DISPATCH' message**

Users are provided with the two mechanisms discussed in section 2.3.3, allowing them to retrace their steps to previously viewed documents:

- **Backtrack** - go back through the history one document at a time, starting from the last document seen.

- **History list** - the history filter presents users with a list of all documents seen this session, ordered by time (i.e. the first document seen is at the top of the list and the last seen is at the bottom). They are able to select and re-display any document in the list. In addition, users can choose to have every document (including duplicates) shown in the list, or they can have the list show only one occurrence of each document.
The history filter provides three additional areas of functionality to those described above. The following sections discuss these areas.

4.1.1. Footprint facility

The history filter can optionally inform the user if they open a document that they have already seen. It maintains an internal data structure containing an ordered list of all documents opened so far. The system broadcasts an 'IS.OPENING' message every time users open a document (see figure 4.1). The history filter searches for the DoctName and Offset, from the current message, in its internal data structure. If a match is found then users are informed that they have already seen the current document. However, the history filter will not inform the users if they have used it to open the duplicate document.

![History footprint interface](image)

Figure 4.3: The history footprint interface

A small window, containing an 'OK' button, is used to inform users that they have seen the current document more than once in their session (see figure 4.3 above).

The original implementation of this interface required the user to move the mouse over the window and click on the button. This was changed so that the window automatically disappears when it loses the system focus. The latter method has the advantage that users can see the footprint information but do not need to make any conscious effort to remove it. This makes the interface less intrusive, allowing the users to register its appearance without completely distracting them from their current tasks.
4.1.2. Desktop state

The desktop metaphor promotes the display of documents within windows which users are allowed to stack on top of each other. When the desktop becomes cluttered it is not easy for users to know which documents are still available on the desktop.

An 'IS.OPENING' message is circulated around the system every time a document is opened. Furthermore, every time a document is closed, or replaced\(^8\), the system generates a 'CLOSE.FILE' message (see figure 4.4).

\begin{verbatim}
\Action CLOSE.FILE
\DoctName 400.04.02.92.16.38.29
\DoctType TEXT
\Offset 0
\UniqueDocID 1
\end{verbatim}

Figure 4.4: Microcosm 'CLOSE.FILE' message

The history filter already monitors the filter chain for 'IS.OPENING' messages, allowing it to maintain a list of opened documents. If 'CLOSE.FILE' messages are monitored then the history filter can maintain a representation of the system state (e.g. which documents are still open on the desktop, or have been already closed).

![History List](image)

Figure 4.5: A history list showing both open and closed documents

The history filter is able to display a list of documents (see figure 4.5) that users have already seen in the current session. Each item in this list corresponds to a

\(^8\)When a document is opened it will either appear in a new document viewer or a viewer that is already in use. The user is able to dictate which by setting a bookmark on each Microcosm viewer.
single document, and contains an icon (representing the document type) and a
textual description. A bold font is used to denote open documents and a plain font
to denote closed documents. This method allows the history filter to reflect the
system state within the user's current history list.

4.1.3. Save and load history files

The history filter is able to store the current history list in a file and retrieve previous
history lists from history files, allowing the user to maintain a history list across
multiple sessions. This provides users with the facility to see which documents they
have already seen in every session contained within the history file.

Furthermore, these history files can be converted for use by the guided tour
system (described in section 4.2.3). Users view the required documents in the correct
order, save the history into a file and finally convert the file into a guided tour.

Users have the ability to set bookmarks on any Microcosm viewer, signifying
that they do not want it to be reused by the system when a new document is
opened. Each time the user sets a bookmark the system generates a 'LOCK.FILE'
message (see figure 4.6).

```
\Action LOCK.FILE
\DoctName 400.04.02.92.16.38.29
\Offset 0
\Selection
\DoctType TEXT
\UniqueDocID 1
```

Figure 4.6: Microcosm 'LOCK.FILE' message

In addition, the system generates an 'UNLOCK.FILE' message (see figure 4.7)
when the user resets a book mark. The guided tour system uses these messages to
mimic the user's locking and unlocking of viewers.

```
\Action UNLOCK.FILE
\DoctName 400.04.02.92.16.38.29
\Offset 0
\Selection
\DoctType TEXT
\UniqueDocID 1
```

Figure 4.7: Microcosm 'UNLOCK.FILE' message
If the history only stored 'IS.OPENING' and 'CLOSE.FILE' messages then its files could not be used to produce fully functioning guided tours. In order for this to be achieved the history filter had to be extended to store 'LOCK.FILE' and 'UNLOCK.FILE' messages. With these changes in place, users are able to create tours that exhibit full control (opening, closing, locking or unlocking) over a sequence of document viewers. The conversion of history files provides users with an easy mechanism for tour creation.

4.2. Mimic system

The mimic system allows users to construct, edit and view sequences of documents (otherwise known as guided tours, see section 2.3.4). Users do not need to know where they are within the set of documents whilst they are following these tours. Furthermore, they do not need to think about which document they should see next, as the path they will follow has already been created for them.

There is, however, nothing stopping them from straying from the pathway (e.g. by following links). When they are ready to proceed along the tour they can continue from the last document read. The Microcosm mimic system is constructed from three separate processes:

- The mimic viewer
- The mimic filter
- The mimic generator

Each document type in Microcosm requires a process to view it. Mimics exist as meta-documents allowing individual documents to be grouped together. It is for this reason that the front-end of the mimic system has to be implemented as a document viewer.

The mimic viewer provides the front-end of the system, all interaction between users and the mimic system is achieved through this. However, the viewer does not receive the control messages that are needed to monitor the progress of the mimic. Therefore, part of the mimic system has been implemented as a filter.
The mimic filter receives the results of user interaction with the viewer through the Microcosm message system. The third part of the system (the mimic generator) is used to construct or edit mimic files and to translate history files into mimics. The manner in which the three parts of the mimic system relate to each other can be seen in the following diagram (see figure 4.8).

![Diagram showing the relationship between the three parts of the mimic system.](image)

Figure 4.8: The relationship between the three parts of the mimic system.

The following sections describe the implementation of the three separate processes in more detail.

4.2.1. The mimic viewer

Microcosm regards mimics as documents, they are accessed by users in the same way as any other document type (i.e. by a viewer). When users open a mimic document they are presented with a mimic viewer. All interaction between users and the mimic is through the mimic viewer. For example, when moving from one document to another along the tour there is no need to close the first document and open the second. All document control (connected with the progression of the user along the tour) throughout the duration of the tour is produced by the mimic system itself.

Mimics provide the mimic author with control over the following functionality within the Microcosm system:

- Which documents they wish the user to read.
• The order in which those documents are presented to the user.
• Whether a new document will replace an old one or appear along-side it.
• What mechanism is used to progress along the tour (user controlled, controlled by the document content, or by a clock).

Certain facets are still controlled by the user via an interface provided by the mimic viewer (shown below in figure 4.9). The main window is sub-divided into four areas: the progress indicator, the control buttons, the current progression method and an information window.

![Progression method](image)

The progress indicator starts off as a white bar which is filled with red (from the left to the right). When users are half way through the tour the indicator will be half red and half white. When the tour has been completed the indicator will be completely red.

The control buttons allow users to progress from one document in the tour to the next. For certain mimic progression methods the buttons will allow the tour to be paused or seen again (e.g. at the end of the mimic). The text on the buttons is updated dynamically to show the user their current function.

The current progression method is represented by the combination of an icon with the text on the control buttons. Figure 4.10 shows the available progression methods and the associated icon and button representations.

The simplest progression method is the user method (see top of figure 4.10). The mimic system opens a document and waits for the user to press a control button. The clock method (see left of figure 4.10) waits for a pre-determined amount of time
(set from within the mimic generator) to pass after the document has been opened, before continuing. The system method (see right of figure 4.10) should be used for media of a temporal nature. The mimic system will open a document then wait for a message from the document viewer to signify that the document has been seen. For example, a video sequence will play and then the mimic system will open the next document.

The user and clock methods can be combined (see top left overlapping region at the centre of figure 4.10) : the mimic will open a document then wait for a set amount of time to pass or for the user to press a control button. Similarly the user and system methods can be combined (see top right overlapping region at the centre of figure 4.10), resulting in the system waiting for the document to finish playing or for the user to press a control button.

Lastly, a method called pause (see bottom of figure 4.10) can be combined with either the system or clock methods. This allows users to temporarily stop the otherwise automatic progression of the mimic. All the possible method combinations can be seen in the following diagram (figure 4.10).

![Diagram of mimic progression methods]

The information window provides authors with a mechanism to present users with information pertaining to their current position in the mimic. In most cases this will be used to display the description of the current document. The information
window can be opened or closed by using an information button on the mimic viewer.

4.2.2. The mimic filter

The front-end of the mimic system is implemented as a Microcosm document viewer, allowing the mimic to be treated as a document by the user and the rest of Microcosm. The viewer has no knowledge of the user's current position in the mimic, nor of the mimic's content. All of this information is stored within the mimic filter.

The back-end of the mimic system must be implemented as a filter as it is only these processes that are able to receive and send the control messages required for the mimic's operation. For example, the mimic filter transmits 'AUTO.DISPATCH' messages to open documents.

The mimic filter can open, close, lock and unlock any document as part of a mimic. The progression of the mimic is controlled by the user through the front-end mimic viewer. This communicates with the back-end filter by sending a set of private messages (that only is only understood by the mimic system) through the standard communication system.

The mimic filter can open documents by broadcasting 'AUTO.DISPATCH' messages and close them by sending 'CLOSE.DOCUMENT' messages. It can control which document viewers are reused by the system by sending 'LOCK.FILE' and 'UNLOCK.FILE' messages. In this respect it is similar to the history filter described earlier. This similarity allows users to generate mimics automatically from history files.

The mimic filter maintains the name of the current mimic document along with the current position within that mimic. Users can open a second mimic when they are part of the way through the first. The mimic filter will then place the information about the first mimic onto a stack. When the user has finished with the second mimic this information will be retrieved and the first mimic will continue. This ability to suspend one mimic in favour of another provides users with the following features:

- They can follow mimics that contain other mimics
• They can venture from the mimic path, maybe looking at other mimics, without losing their position in the first mimic

In addition, users can close mimics at any point through their progression. If there are other mimics in the stack then they will be offered the option of continuing along these paths, starting with the one on the top of the stack. The ability to effectively nest one mimic inside of another means that they can be structured in a more modular manner, allowing them to be re-used in different contexts. Furthermore, this ability makes the creation of mimics easier by allowing them to be constructed as small self contained sections.

4.2.3. The mimic generator

The mimic generator is neither a filter or a viewer, it exists as a stand-alone program that does not use any of the Microcosm communication system. As such it can be used either with or without the rest of Microcosm being active. It provides authors with the ability to create, edit and translate mimics between different formats. The mimic generator provides authors with three different mimic creation mechanisms, these are:

Conversion of history files - History and mimic files share the same underlying binary format and contain approximately the same information. The exceptions to this are that the mimic file contains a title and the history file does not. Furthermore, both contain timing information relating to the opening and closing of documents. However, the times in the history files are elapsed times from the beginning of the file. In contrast, the times in the mimic file are used to store the length of time that a document is open. In order to translate a history file into a mimic some conversion is needed.

Language based creation - Mimics can be created from ASCII text files containing a mimic language script. This script is compiled by the mimic generator to form a binary mimic file. The process is reversible, allowing mimics to be edited using any available text editor.

Form based creation - This method allows users to create mimics without needing to know the syntax of the mimic language. Every operation provided by the language is duplicated within the form mechanism. Furthermore, the form
based method provides users with the editing facilities that are available within the language editor (for example, they can move mimic entries around and delete them).

A mimic must be stored in its binary format before it can be used by the mimic system. The mimic generator allows users to generate these binary files with these three mechanisms.

4.3. Local map filter

The local map filter in Microcosm provides users with a graphical representation of a specific document and all the documents linked into and out of it. Tools similar to the local map are discussed in section 2.3.2. The local map filter has been extended beyond the traditional implementation to allow it to operate within a dynamic open system. For example, it will construct its local knowledge as the user navigates through the hypermedia document set rather than requiring a complete set of local maps to be pre-constructed. Furthermore, it will show the user both explicit and implicit links. However, it does no additional processing to discover the existence of the implicit links, instead it relies on the rest of the system to provide them. For example, when the user highlights an area of text and asks the system to find links from their selection the local map will record the outcome of the associated linkbase search.

The primary task of the local map filter is to provide the user with a graphical representation of the current document together with the links between it and other documents. The use of implicit links and the ability to view any file in the file system as a document within Microcosm makes this difficult. The local knowledge necessary to achieve this task can be constructed in two ways (both methods capture the full range of link types; explicit and implicit):

**Complete pre-construction** - The set of link anchors can be extracted from the linkbases and the set of documents (that constitutes the current document set, or application) can be extracted from Microcosm. The intersection of the two sets can be computed and used to construct a complete set of local maps for the current hypermedia document set. Although it would be possible to implement this process it would be computationally intensive. Furthermore,
if either the links or documents are changed then this set intersection would need to be completely re-calculated.

**Dynamic construction** - As the user moves around the hypermedia document set they will discover more links than they actually follow. For example, consider the scenario when the user opens a document that contains five explicit links. They highlight a particular word and search for links from it, the system responds, providing them with five implicit links from their selection. They might follow one of these links to a second document, also containing ten links (this time they are all explicit). The user has now seen two documents and followed one link. The local map, however, has constructed partial maps for the two documents and all the links discovered. In addition, it has also constructed partial maps for all the documents that are at the end points of the links discovered so far. This is illustrated in figure 4.11, where black shading represents the area of the hypermedia document set explored by the user and grey shading represents the area stored by the local map filter.

![Diagram showing the dynamic construction of local maps.](image)

The local map tool needs to be implemented as a filter, allowing it to access the control information that is communicated between the filter components within the system. However, it could be argued that the maps it presents should be treated as documents. This would mean that the tool would have to be implemented as a viewer, or implemented in a divided manner like the mimic system. It was decided that, in order to simplify the implementation, it would be implemented wholly as a filter.
When any document is opened within the Microcosm system its document viewer sends out a 'FIND.BUTTON' message. When the linkbase receives such a message it will send of a 'FOUND.BUTTON' message for every explicit link that has a source anchor within the newly opened document. The local map tool is implemented as a filter, allowing it to receive both message types. Therefore the local map can use these messages to dynamically construct its knowledge of the structure of the hypermedia document set without any additional processing.

The local map filter will show users all links that have been discovered whilst it has been functioning; every link (explicit or implicit) that has been located by the current user (and previous users) will be represented by the local map filter. However, the local map filter itself does no work to uncover these links. Instead it monitors the communication system analysing messages that result from the user's exploration of the hypermedia network. The following sections discuss this analysis and other features of the local map filter in more detail.

4.3.1. Ability to show different types of links

So far we have only discussed the local map's ability to show explicit links. For example, it can analyse the 'FIND.BUTTON' and 'FOUND.BUTTON' messages to locate any buttons that exist within documents as they are opened.

By further extending the message monitoring of the local map filter it can be made to show all links in the system without requiring any extra work from the users. Furthermore, users are given the ability to select which types of link will be recorded by the local map filter.

Microcosm provides the user with two mechanisms for discovering implicit links; they can select an area of a document and instruct the system to either follow links from the selection or search for links contained within the selection (show links). The former instruction generates a 'FOLLOW.LINK' message, and the latter generates a 'SHOW.LINKS' message. In both cases the linkbases will respond with a stream of 'DISPATCH' messages, one for each link discovered. The local map filter can analyse these messages, allowing it to represent the explicit and implicit links that are discovered by the actions of the user.
The local map displays implicit links (e.g. generic links) as explicitly instantiated instances. For example, suppose a generic link is created on the word 'map' and this word exists in ten different locations within the document set. This one generic link would be fully represented by ten explicit buttons, it is this representation that the local map filter shows the user. This difference in interpretation allows the local map filter to represent implicit links without needing to fully instantiate them for every possible occurrence of their source anchors. Without this ability the local map filter would not be able to represent implicit links.

Microcosm also includes a full text retrieval filter (Li, 1993), allowing the user to select an area of text within a document and then instruct the system to compute links from it. Instead of the selection being passed to the linkbase (as in the case of a follow link scenario) it is passed to the full text retrieval filter. This then processes the selection and generates a stream of 'DISPATCH' messages, one for each document that closely matches the content of the selected text. This process can be thought of as generating implicit links between the selected text and the matched documents. The local map filter is able to represent the relationships generated by the text retrieval process in the same way as any other implicit or explicit link.

The user is able to select which types of links are recorded by the local map filter, and can change this selection and any point. For example, if the user decided that they only wished the local map filter to record the links generated by linkbases (i.e. not the ones from the full text retrieval process) then they could do so.

The current implementation of the local map filter does not monitor any of the messages generated when a new link is authored. Such links will only appear on the maps when they are discovered by the user after they have been constructed. For example, suppose the author creates an explicit link from their current document to another. The link will not appear on the map until they open the document or they execute some other action that results in its discovery (e.g. follow link or show links). However, the local map could be extended to monitor the required messages allowing newly authored links to appear upon creation.

4.3.2. Links into documents

As a user moves around a document set they will immediately see the explicit links in the form of buttons. In addition, they will discover implicit links of the generic or
full text retrieval type. All these links, irrespective of the type will displayed in the context of their source anchors. Users will not be able to see a link's destination anchor until it has been followed.

The local map is able to display links in the context of either their source or destination anchors. It is able to show users not only which documents are connected to the current document by links emanating from it, but, also those that have their destination anchors in the current document and their source anchors in others.

The local map filter stores one map for each document that has been viewed. The set of maps for the current documents are stored in two separate areas, these are:

**Document list** - A list that grows to contain one entry per document seen so far. Each entry contains: the document's unique identifier, its type and its description.

**Link matrix** - A matrix that grows to contain $n^2$ items (where $n$ is the number of documents seen so far). Each item is used to show whether any links exist between its two respective documents. For example, a link from document $A$ to $B$ will be stored in item $(A, B)$ in the link matrix.

In order to draw a map the filter must first extract the information about the focal document\(^9\) contained in the document list. The position of the document in the document list ($N$) is used to access the link matrix. The relationships between the document list and the link matrix are illustrated in the following diagram (see figure 4.12).

---

\(^9\)Focal document - the document at the centre of the map i.e. the focus of the map.
If a link exists from document $x$ to $y$ it is stored as a non zero value at position $(x, y)$ within the link matrix. The filter searches through the line $x = N$ ($N$ contains the index of the focal document within the document list) to locate all links out of the focal document. When it finds a non zero matrix value it adds a representation of a link to the local map for that document.

Because the link information is held in matrix form it is a simple process to present the user with not only the links out of a document, but also the links into a document. The filter searches through the line $y = N$ ($N$ is the index of the focal document in the document list) to locate all links into the document. The two search lines are illustrated in the following diagram (see figure 4.13).

The map filter displays links out of a document as solid black lines on the map and links into a document as dashed black lines. This can be seen in figure 4.14.
4.3.3. Dynamic map updating

The local map filter can display a map for any document that appears in its document list, but for simplicity only one map is displayed at any time. Consider the scenario where a user has opened the map filter for document A. They then view a second document B. The map is still showing the map for document A. The situation would be improved by making the map dynamically update its display to show the map for the new document.

The local map already monitors its communication channel for the 'IS.OPENING' message (used in the map construction) transmitted every time a document is opened. In addition, it can use this message to update its display. For example, when the user opens document B the map will receive an 'IS.OPENING' message, triggering it to show the map for document B.

A second type of dynamic updating can be achieved by using other messages as signals to re-display the current map. For example, suppose the map is currently displaying a map for document A, containing one link out from the document and three links into it. The user now highlights a paragraph within A and tells the system to search for all links out from it. The system locates six links (explicit and implicit), the user is now confused, since one part of the system shows A linked to six documents and another shows that there are only four.
How can this be? Before the user searched for links from A the local map was only showing links that were authored as buttons (the original four). The user then searched for all links from a paragraph and the system found six links of different types (ranging from buttons to generic links). Hence the map is now incomplete, it should be updated and completed.

Every link that is found as a result of such a user initiated search results in a 'DISPATCH' message being transmitted. The local map has been extended to use such messages as a re-display trigger allowing it to update its display every time the user locates new links. This enables the local map filter to always display the most accurate representation of the hypermedia network.

4.3.4. History within the local map filter

The local map filter maintains a list of maps seen by users in the current session. Users are provided with the ability to backtrack through the previous maps, by selecting documents with links back to the previous focal document, or by using the 'Back' button on the map interface (see figure 4.14).

In addition to the history maintained by the local map filter, the local map files themselves maintain a history of use. Every time the local map filter is started (usually at the beginning of the session) it records the current user, time and date. Users can display a complete list of every session that the map has been used in.

This built-in history can be used to allow users to choose whether or not to display links from a particular session. For example, if they see that some of the links were discovered in a session when they were carrying out a particular task, they can choose to see only those links on the local maps. Furthermore, if the hypermedia network is highly connected there is a risk that the local map for any one document will become cluttered and unusable. The built in map history is provided as a mechanism that allows the user to filter the complexity of the local map structure. If the maps do become too cluttered then the users can elect to show only the links discovered most recently etc.
4.3.5. Different levels of maps

The local map filter uses the hierarchical ini. file system (see section 3.3.5) to store all of its option settings and the name of the local map file (used for storing maps between sessions). This allows local maps to be generated at any of the three levels (system, application or user). Thus a single set of maps defined at the system level could be used for every user and for every application.

Normally, however, the application level would be chosen, allowing a single set of maps to be shared by every user of the specified application. Application level maps are advantageous as they only store maps for the relevant application, but will be constructed quickly due to use by multiple users.

4.4. Summary

The Microcosm message system, combined with filter technology, provide the necessary functionality to implement easily some of the traditional navigation tools previously discussed in chapter two. The only exception is the mimic system, which needed to be implemented as both a filter and a viewer (allowing mimics to be treated as documents by the rest of the system).

Filters are perfectly designed to both monitor and generate the control messages needed in the construction of Microcosm navigation tools. For example, the history filter and the mimic system both make good use of these abilities. Both allow users to re-transmit messages captured earlier in order to open and control specific document viewers.

The filter management structure allows users to choose which navigation tools to use and when to use them. Such flexibility will not affect the various tools as long as they operate in a self contained manner. For example, if the local map filter relied on the history filter to maintain its own private history, then it could not function without it.

The division of processes that lead to the viewer/filter distinction causes problems within the mimic system. Without this separation the mimic system could
have been implemented as two processes rather than three (i.e. a mimic generator and viewer). Furthermore, the length of the communication route between the viewer and filter (which is a function of the number of filters and the filter/viewer segregation, see section 3.3.1.5) can cause noticeable lags between users interacting with the system and its response.

The extendibility of the message system allows each part of divided tools, such as the mimic system, to communicate with each other without affecting the remainder of the system. At the same time new parts of the system could be designed that used the private mimic system messages but had no effect on its operation. Furthermore, the modularity of the mimic system allows either the front or back-ends to be changed without affecting the other.

All of the navigation tools discussed within this chapter are being employed by real users. The Microcosm system is currently being extensively used by various projects, both in Southampton and elsewhere. In particular, it has been exploited during user trials as part of a 1st year undergraduate course on 'Cell Motility' within the Department of Biology at the University of Southampton (Hutchings, 1993). As part of these trials the participants were asked to fill in questionnaires which covered every aspect of both the hypermedia material and the system interface itself.

The users' responses to the mimic system were extremely favourable. Their response to the history filter was slightly less favourable, although the majority of those questioned found it to be useful. The most overwhelming response concerned the local map filter, with eighty-three percent of the users saying that they found it very useful (Hutchings, 1993).

The history filter and the mimic system currently form part of the supported release of the Microcosm system. The local map filter, however, is not part of the release for a variety of commercial reasons, the largest of which is that it had not been fully debugged at the point when the release software was decided.
5. Implementing novel navigation tools using filters

Examples of the traditional tools, described in the previous chapter, can be seen in Microcosm and other hypermedia systems (e.g. Intermedia (Haan et al., 1992), Notecards (Halasz et al., 1987; Halasz, 1988) etc.). In contrast, the tools described within the following sections occur, as far as we know, only in Microcosm.

The tools in question are aimed at providing users with additional navigational support within an open hypermedia system. Furthermore, they attempt to solve some of the unique problems associated with Microcosm (as described in section 3.3). These include, the problems caused by the use of implicit links, the separation between filters and viewers and the segregated communication model.

The purpose of this chapter is to illustrate the types of navigation tools that are required within an open hypermedia system and to show how they might be implemented. Whilst the previous chapter dealt with the adaptation of traditional tools to the Microcosm system this chapter focuses on providing novel tools that can exist within, and even exploit, the dynamic constraints imposed by an open hypermedia system such as Microcosm. The work described in this chapter does not discuss the usability of the tools but instead focuses on their design and implementation. Detailed studies of the usability of such tools would be the concern of further research work.

5.1. Memory filter

Systems such as Guide (Brown, 1987), KMS (Akscyn et al., 1988) and Notecards (Halasz et al., 1987; Halasz, 1988) allow users to construct links that have one start anchor and one end anchor (one to one links). Navigation is simplified for users, on following a link they are taken directly from the start anchor to the end anchor.

If the transition is fast enough then users will suffer very little cognitive loss. This premise heavily influenced the implementation of the KMS system. It aims to take users from the start of a link to the end in less than one second. If users can
remember the information at the start of the link when they reach the end then the meaning of the link will be more apparent (for more detail see section 2.3.1).

In contrast, systems such as Microcosm (see chapter three) or Intermedia (Haan et al, 1992; Yankelovich et al, 1988b) allow users to construct links that have one start anchor and many end anchors (one to many links). When users follow such links they cannot be taken directly to an end anchor, as there are more than one to choose from.

Users must select one branch of the link from the list of available branches in order for the link traversal to be completed. The presentation of this list will interrupt the user's cognitive thought process. In addition, they are forced into making a selection from this list of available link branches, causing an even greater cognitive load. If they are presented with a long list, or take a long time to select the link branch, then there is a high likelihood that they will forget the contents of the start anchor before they reach the end anchor. On arriving at the end of the selected link they will struggle to understand its purpose as they have lost the context provided by the link's start anchor.

One to many links provide a richer linking environment, but have the drawback that users are likely to become disorientated when following them. Every attempt should be made to lessen the cognitive load placed on the user whilst they are travelling from the start to the end of such links. The aim of the design and implementation of the memory filter is to reduce this cognitive load.

When users follow one to many links in Microcosm they will be presented with a list of link branches from which to choose. If the link has only one end anchor then the list stage can be ignored resulting in users being taken directly to the end of the link. Each link is given a description when it is created, with each branch of a one to many link having a different description. These descriptions are displayed by a filter called 'Available links' (see figure 5.1). The user can follow a link by selecting an entry in the list and pressing the 'Follow Link...' button. The window (shown in figure 5.1) will remain on the screen until the user presses the 'Cancel' button.
Users are required to read, understand and mentally filter out link branches from the list before choosing one. No indication is given to users that they may have already seen some, or all, of the end anchors belonging to the link branches presented to them. One mechanism for reducing the cognitive load in this situation would be to overlay such information onto the existing list.

5.1.1. A model of short term memory

The memory filter contains the same basic functionality as the 'Available links' filter (shown in figure 5.1). However, it has been extended to incorporate a simple model of a user's short term memory (STM). A person's STM acts as a temporary store for things that may be later moved to their long term memory (LTM) (Gleitman, 1986).

Research has indicated that a person's STM can hold seven plus or minus two objects (Miller, 1956). Under normal conditions objects pass through the STM and into the LTM. However, it is possible for objects to pass out of the STM but not enter the LTM, whereupon they will be forgotten. This has been demonstrated experimentally in the following manner (Gleitman, 1986).

A set of subjects were given lists of three letter groups to remember. After a specified time period they were asked to recall these groups. When the time period was short, it was found that the people could repeat them easily. The experiment was repeated, but this time the people were asked to carry out an unrelated task in-between the memorising and recall of the three letter groups. This task consisted of counting backwards from an arbitrary three digit number in intervals of three. The
people were then asked to recall the letter groups, but could not do so. It is thought that the unrelated task (e.g. the counting) was displacing the three letter groups from the STM before they could be moved into the LTM (Gleitman, 1986).

The memory filter attempts to avoid showing users link branches to documents that they have already recently seen. In effect, it is using the state of the system (as represented by the list of documents seen by the user) to dictate which links they should, or should not, follow.

A similar device can be seen in the LINCTUS PB prototype (Briggs et al, 1991) where a short term memory is modelled by a list of the last five documents seen. These documents are then excluded from any information presented to users. Such exclusion prevents them seeing the same documents repeatedly (e.g. moving in circles within the information network). Further examples of using the system's state to dictate which links can be followed can be seen in the Storyspace system (Bolter & Joyce, 1987; StorySpace, 1991), TRELLIS (Furuta & Stotts, 1990; Stotts & Furuta, 1989; Stotts & Furuta, 1990) and finally the MacWeb system (Nanard & Nanard, 1991).

5.1.2. Mechanisms for presenting extra information

The 'Available links' filter in Microcosm presents users with a list of link branch descriptions when they follow a one to many link. The memory filter provides the same basic functionality, but presents users with additional information concerning the number of documents that they have seen since they last saw the end anchors associated with the links concerned. Such extra information could be presented by one of three mechanisms:

Inserting a representation of the elapsed time into each line of the list - Every link branch is represented by a textual description of its end anchor document type followed by the description of the document that it ends in. The number of documents seen since users last saw the link's end anchor document could be inserted into the start of each line. This would provide users with an accurate picture of which links had been seen and when. However, it would place an extra burden on their cognitive processes as they would now have to read and interpret this additional information.
Using position in the list to indicate chronological ordering - The link branch descriptions could be sorted into chronological order; the top of the list would contain descriptions of links to documents that have not been seen, whilst the bottom would contain the ones seen most recently. This sorting would provide users with the required additional information but would have two major disadvantages:

- Links do not arrive at the memory filter at the same time. The original ordering of the links represents the time that each link has taken to arrive (from the issue of the follow link command). For example, linkbases on remote machines or the use of image processing mechanisms will result in links arriving later than ones from the local machine's linkbases. Such information is implicitly portrayed by the ordering of the link descriptions in the memory filter.

- Because the link descriptions arrive at different intervals, they would need to be constantly sorted. Not only would this be computationally wasteful, but it would cause the users disorientation. The list of link descriptions would be constantly shuffled around before their eyes.

Using colour to denote time - The third option would be to use colour to denote the time elapsed since users had last seen the end anchors for the links concerned. This would require little additional computation, need no sorting, and utilise a metaphor from the user's natural world. Items in the real world change colour as they age, for example paper turns from white to yellow as time passes.

In fact the colour solution suffers none of the drawbacks of the sorted solution and does not require the users to read any additional information. As a result of this the colour method was implemented.

5.1.3. Use of colour

Four levels of colour were used (shades of grey were chosen) to represent the position of a link's end anchor document in the short term memory. This provided users with indication of the relative, but not absolute, number of documents that had been seen since the end anchor documents for each link had last been seen.
The natural colour changes that paper (and other objects) undergo as they age led to the decision to use shades of colour in the memory filter, as opposed to using distinct colours for each stage in the memory. Furthermore, such considerations as colour blindness led to the choice of the following shades: black, dark grey, medium grey and light grey.

These grey shade values are only implemented as a default and the users are able to change them to suit their own individual tastes. Any such colour changes will be stored in the layered ini. file system described in section 3.3.5. The following diagram illustrates the ordering and use of these colours (see figure 5.2).

The STM within the memory filter is quantified by the following two measurements:

- The amount of time that has elapsed since the end anchor document for a particular link branch had previously been seen.
- The number of documents that have been seen since the document containing the end anchor for a particular link branch had previously been seen.

The length of the short term memory in terms of these two measurements is set by the users. Furthermore they are able to choose to use just one of the above two measurements as their STM definition. For simplicity, we will examine a scenario where the user has elected to use just the latter mechanism. In addition, they have set the maximum length of the STM to be nine documents.

---

**Figure 5.2**: Default colours used by the short term memory filter and their meanings.
The last nine documents that the user has read will be placed into the STM in the order that they were opened. If a document has not been seen by the user or was seen more than nine documents ago it will not be in the STM. Links to such documents will be coloured black within the memory filter. This colour provides the highest contrast with its white background, making these links visually stand out.

The STM maintained by the memory filter is sub-divided into three bands (with each band representing a third of the short term memory), in this scenario the bands represent:

**Band1** - These are the three documents seen most recently (e.g. positions one to three). Links to these documents will be shown in light grey within the memory filter. This colour is the closest to the white background, reducing the visual appearance of these links.

**Band2** - These are the documents that occupy positions four to six in the STM. Links to these documents will be shown in medium grey within the memory filter. This colour is not as close to white as the band1 colour and so these links will be more visually apparent.

**Band3** - These are the documents that occupy positions seven to nine in the STM. Links to these documents will be shown in dark grey within the memory filter. This colour is the closest to black and will show up more than the colours used for band1 and band2.

As new documents are seen they will be placed in the STM at position one. The documents already present in the STM will be shuffled down towards position nine. The document at position nine will be removed from the memory.

The previous diagram (see figure 5.2) illustrates the ordering of the colours used to represent links to a document as it moves through the STM. For example, links to a document that has not been seen will be coloured in black, making them stand out in the memory filter. The user will then view the document placing it into the STM at position one. Any links to this document will be displayed in light grey. As the user views more documents the original document will move down through the STM towards position nine. As it does so links to it will increase in contrast being coloured in medium grey then dark grey. When the original document leaves the STM all links to it will be once more coloured in black.
5.1.4. The memory filter in use

Figure 5.3 illustrates the memory filter in use. Four representations of the memory filter are included to show the passage of a link's end anchor document through the STM. Figure 5.3(a) shows three links from the selected start anchor, none of the links has been followed. Figure 5.3(b) shows the same three links but the third has already been followed. The user has seen the end anchor document. Figure 5.3(c) shows that both the second and third links have been followed, the third being the furthest into the memory. Finally, figure 5.3(d) shows that the user has seen all three links, the most recently seen being the first in the list.

From a design point of view, the use of colours enhances the visual impact of links to documents not in the STM, and lessens it for links to documents that users have recently seen. As documents move through the memory links to them darken in colour, becoming more distinct against the white background.

The memory filter allows users to customise the length of the STM in terms of the two measures discussed earlier. In addition, they can flush the contents of the STM or turn it off completely. If the STM is turned off the memory filter will provide identical functionality to that of the Available links filter.

The memory filter was used extensively during user trials as part of a 1st year undergraduate course on 'Cell Motility' within the Department of Biology at the University of Southampton. However, it is not currently part of the supported
release of the Microcosm system. It was excluded from the release for commercial reasons.

5.2. Show links filter

The Microcosm system provides users with the ability to link arbitrary sections of documents together. These sections can be within any documents (more precisely any file) within the file system. It is logical to assume that as the number of documents linked together by Microcosm increases, so will the number of authored links. The creation of all links, explicitly, by an author soon becomes unfeasible. Microcosm provides a type of implicit link (called the generic link) as a solution to this problem. Implicit and generic links were discussed in more detail in sections 2.3.1.2 and 3.3.2.

Explicit links can be indicated to the user as buttons because their start and end anchors are defined when the link is created. However, a generic link does not have a defined start anchor when it is authored, it only has a defined end anchor. The unresolved nature of generic link start anchors makes them difficult to highlight to the user.

The theoretical solution to the problems involved in the highlighting of generic links would be to resolve them all, thus translating them from implicit to explicit links. In order to achieve such a translation the process would need to obtain the set of possible anchors from the link database. It would then need to resolve every anchor against every document, within the entire file system. By doing this it would obtain the complete set of explicit links that could be formed from the original set of implicit links. These explicit links could then be highlighted when necessary.

The above solution is a worst case scenario, as such it would be possible to implement but would not provide the users with the highlighted links in real time. The complexity of the above scenario can be drastically reduced by only resolving the generic links on a need to know basis. For example, only the generic links within the current document need be resolved (i.e. the document that the user is currently reading).
The show links filter allows users to select arbitrary areas of a document, then resolve all the generic links into a set of applicable explicit links. In effect it translates the area of text selected by the user into a sequence of messages understood by the linkbases (Heath, 1992). The first version of the show links filter was implemented as a filter, a separate entity from the linkbases themselves. This allowed users to choose whether or not they would use the functionality that it provided, by inserting it into or removing it from the filter chain.

After the analysis of a variety of existing link databases it was decided that the complexity of the generic link resolution problem could be reduced still further. This was achieved by only resolving links that had start anchors which consisted of single or double words. The analysis showed that this accounted for eighty-five percent of the total number of links (see figure 5.4). Resolving the generic links with longer start anchors would be possible, but the associated trade off in speed was deemed unsatisfactory for the increase in links returned.

![Figure 5.4: Percentage of links whose start anchors are less than three words in length.](image)

The first implementation of the show links filter monitored the communication system for a Microcosm 'SHOW.LINKS' (see figure 5.5) message. This was transmitted by a text document viewer when the user selected an area of text and chose to search for all links contained within it.
There are three main classes of proteins which make up the cytoskeleton - microfilaments, microtubules and intermediate filaments.

When the filter receives such a message it first sub-divides the selection into separate words and then it removes all stop words (such as 'and', 'or', 'a' etc.). The filter assumes that links will not be authored from these common words. The remaining words are then transmitted to the linkbases using the Microcosm 'FOLLOW.LINK' message. Each word is transmitted once on its own and then with its immediate successor. In addition, duplicate words are only sent to the linkbases once, thus reducing the number of link searches carried out by the linkbases.

Each linkbase must search through its entire link contents every time it receives a 'FOLLOW.LINK' message. It was not possible to optimise this process specifically for the show links filter as the linkbase cannot discern what purpose the 'FOLLOW.LINK' message served. For example, users selecting an anchor and searching directly for a link as opposed to selecting a large area of text and searching for links contained within it.

The show links filter has been used extensively by users of the Microcosm system and has become a useful tool for discovering the existence of generic links. Furthermore, it is now included within the supported release of the Microcosm system. However, the inefficient nature of the communication chain (see section 3.3.1.5) coupled with the inefficiencies associated with repeated linkbase searches made this implementation of the show links filter cumbersome. It was these problems that eventually prompted its re-design.

The second implementation of the show links functionality has been incorporated directly into the linkbases themselves. The linkbases are now responsible for receiving the 'SHOW.LINKS' message and for processing the selected text. This integration has two side effects. Firstly, the load on the communication system is removed, as all processing is now internal to the linkbase. Secondly, the linkbase searches can be optimised as the 'SHOW.LINKS' message can be differentiated from the 'FOLLOW.LINK' message.
The combination of the show links and linkbase filters is contradictory to the open modular nature of the Microcosm system. However, in this particular situation it was felt that the ability to resolve generic links in this manner should become part of the functionality provided by the linkbase.

5.3. Graph analysis tool

The network structure created by the documents and links in a hypermedia document set can be suitably represented by a graph. Documents would be represented by the graph nodes and the links by the graph edges. Furthermore, if the links have associated directions (i.e. a link is from document A to B and not B to A) then the network can be represented by a directed graph. Such graph based representations of hypermedia structures can be seen in (Conklin, 1987; Utting & Yankelovich, 1990; Rivlin et al, 1994).

As the number of documents within a hypermedia network increases, it is reasonable to assume that the number of links (explicit and implicit) will also increase, especially within an open hypermedia system. The increase of either causes problems for the layout of the graph. Graphs can quickly become cluttered and unusable (Conklin, 1987; Utting & Yankelovich, 1990). These problems were discussed in more detail in section 2.3.2.

The graph analysis tool (GAT) in Microcosm uses a different mechanism to represent the complete structure of the hypermedia network. Firstly, the documents are listed in an arbitrary order, this list forms both axes of a scatter graph. Secondly, each point on the scatter graph represents a link between the two respective documents. For example, a uni-directional link between documents A and B would be represented by a point with the co-ordinates \((A,B)\) on the scatter graph. Furthermore, a bi-directional link between the same documents would be represented by points with the co-ordinates \((A,B)\) for the forward direction, and with \((B,A)\) for the other. An example of such a scatter graph representation of links can be seen below (see figure 5.6). The value actually stored at the co-ordinate \((A,B)\) is the number of links from document A to document B.
The use of a scatter graph to represent the hypermedia network allows the GAT to avoid the problems inherent in the layout of complex directed graphs. For example, the positioning of the nodes so that the edges do not become an incomprehensible tangle. The GAT is primarily aimed at authors who wish to create hypermedia networks which are easy to navigate through. However, users who wish to examine the underlying network structure may also benefit from its functionality.

The GAT internally maintains the data to be plotted on the scatter graph as a matrix. This matrix is similar to the link matrix implemented within the local map filter (see section 4.3). However, the values held in the matrix have different meanings in each case. In the local map filter's link matrix the values held are used to denote the session that the links were discovered in. This allows users to select links from a particular session (see section 4.3.4). In contrast, the values in the GAT's link matrix represent the numbers of links (explicit and implicit) between the respective documents. The similarity between the two link matrices allows the GAT to be used to pre-construct sets of maps for the local map filter.

The following sections discuss specific areas of the design and implementation of the GAT.

5.3.1. Explicit links

Microcosm supports the authoring of explicit links, these are links whose start and end anchors are defined when they are created (see section 2.3.1.1). Each link is
stored as a single entry in a linkbase. The record of each entry contains details of both the start and end anchors.

Before the GAT can display the structure of the hypermedia network it must first generate the associated internal link matrix. Explicit links can be evaluated by examining the contents of each linkbase, since each link contains details of both anchors. The GAT searches each link in turn (during an off-line search process), setting the appropriate values within its link matrix. The matrix will contain a complete representation of all explicit links (within the hypermedia network) once the GAT has examined every link in the linkbases. Once such a link matrix has been generated the following structures can be identified:

**Loops** - Are formed when both the start and end anchors of a link are found in the same document (i.e. simple loop). This definition can be extended to include paths from a document to other documents and back again (i.e. complex loop). Loops can confuse users by unexpectedly returning them to the same document time and time again.

**Islands** - Are subsets of the total document set that may be internally connected, but have no connections to any member of the total set not in the subset (complex island). Furthermore, a single document that is not connected to any other member of the document set can be thought of as a simple island. Islands are difficult for users to get to and from, the contents of such structures will very often remain hidden from them. Furthermore, because the contents of the islands are not linked to the remainder of the documents users will have difficulty perceiving their relationship to the majority of the hypermedia information in the document set.

**Dead ends** - Are documents that contain one or more link end anchors but no link start anchors. Users can easily get to the dead end but cannot follow links away from it to any other document. In certain situations these structures can be useful. For example, an encyclopaedia could contain many such dead end documents.

**Start points** - Are the opposite of the dead end structure, such documents will contain a large number of link start anchors but no link end anchors. Therefore, users can easily leave these documents but will find it difficult to get back to them.
Highly linear sections - These are areas of the graph that contain groups of documents connected in such a manner that the only route through them is to follow the single set of links connecting them together. Such structures do not easily fit into the hypermedia ethos, ideally each document should be reachable by more than one path.

The above structure objects can be illustrated within a directed graph. The following figure shows an example hypermedia network, each of the structure objects listed above has been labelled (see figure 5.7)

![Figure 5.7: Structure objects within a hypermedia network.](image)

5.3.2. Implicit links

Up until this point we have only discussed the portrayal of explicit links within the GAT. However, the GAT would prove to be of much more use to both author and user if it could capture the complete network structure. In order to achieve this it must display the areas of the network created by the implicit links within the system (i.e. the generic links).

Generic links are held within the Microcosm linkbases, but only their end anchors are explicitly defined. A start anchor for a generic link is not resolved until a user attempts to follow it. The local map filter circumvents this problem by constructing its maps in a dynamic way, adding the generic links when they are
resolved by the user and the rest of the system. However, the GAT is used as a stand alone tool and so cannot use this mechanism. Therefore it adopts the following strategy:

- Firstly, it extracts a list of documents for a specific hypermedia network from the Document Management System (DMS). This reduces the complexity of the generic link resolution problem (see section 2.3.1.2), by reducing the number of documents to search through. The GAT only constructs a representation of the hypermedia network for this set of documents.

- Secondly, it constructs a list of generic links, extracting the relevant information from the linkbases connected to the specified hypermedia network.

- Finally, it resolves all the generic links against all of the documents listed, effectively translating the implicit generic links into explicit links.

By adding the resolved generic links to the structure previously generated from the explicit links the GAT is able to portray the complete hypermedia network. However, the inclusion of these links within the existing structure will change its appearance. The objects identified before the inclusion (e.g. the dead ends and islands) will not necessarily still exist. Such changes are illustrated in the following diagram (see figure 5.8)

![Figure 5.8: Changes to the structure due to the inclusion of generic links](image-url)
The above diagram (see figure 5.8) shows the following changes to the objects identified in the earlier diagram (see figure 5.7). The old structure is shown in grey, with the changes highlighted in black. The simple island in the previous figure has disappeared, the highly linear section has also disappeared and the complex island has been joined to the rest of the structure.

5.3.3. Locating structure types

The GAT provides a variety of functions specifically designed for the location of certain structures within the hypermedia network. Each function takes a link matrix as its input and produces a list of the specified structure objects. Each structure object can be represented by a list of one or more documents. Functions have been included, within the GAT, to search for simple loops, dead ends, start points and simple islands.

As the interface for each function is identical, there is no reason why extra functions should not be added to the GAT. In addition, the control interface provided by the GAT has been designed to allow for such extension of its functionality. Sets of new functions can be constructed and dynamically linked to the GAT via the Dynamic Linked Library (DLL) mechanism provided by the Microsoft Windows environment. This allows the functionality it provides to be extended without any need for the GAT itself to be recompiled.

Functions could be written that would locate the remainder of the structure objects discussed earlier. For example, complex loops, complex islands or highly linear sections. Furthermore, functions could be incorporated that produce the structural metrics discussed by Botafogo (Botafogo & Mosse, 1992), for example converted distance, relative out connectivity or relative in connectivity.

5.3.4. Scatter graph representation of hypermedia structures

The GAT provides its users with the ability to display a hypermedia network as a scatter graph. It also provides functions to locate specific structural objects within the hypermedia network (e.g. simple islands, dead ends etc.). These two separate areas of functionality overlap. When the scatter graph is examined the user can clearly see the structural objects located by the functions. For example, the following
figure (see figure 5.9) illustrates an equivalent hypermedia structure to that shown in figure 5.7, but this time it is represented by a scatter graph.

![Figure 5.9: Scatter graph of hypermedia structure from figure 5.7](image)

The structural objects discussed earlier can be clearly seen in the above diagram (see figure 5.9). The following list describes the scatter graph characteristics of each structural object in turn:

**Simple loop** - Can be located by looking for points on the line $x=y$. A point at coordinate $(A,B)$ represents a link from document $A$ to $B$. Therefore, a point at $(A,A)$ will be a link from the document $A$ to itself.

**Simple island** - Can be located by looking at the two lines $x=n$ and $y=n$ for any value of $n$. If no points are found in either line then the document represented by $n$ is a simple island.

**Dead end** - These are shown by one or more points in the line $y=n$ and no corresponding points in the line $x=n$, for any value $n$.

**Start point** - These are shown by one or more points in the line $x=n$ and no corresponding points in the line $y=n$, for any value $n$. 
However, not all structural objects are easy to see within the scatter graph. Objects such as complex loops and complex islands are difficult to discern because they involve looking at more than one document concurrently. In addition, objects such as highly linear sections are reliant on the ordering of the documents along the axes. As a result these objects can only be easily located using the function mechanism discussed earlier.

The scatter graph is stored internally within the GAT in the form of a matrix. This matrix is similar to the one implemented within the local map filter (see section 4.3). This similarity allows the GAT to export its matrix in a form usable by the local map filter, providing it with a complete set of local maps for any given hypermedia network.

The GAT's internal matrix can also be exported as a comma separated variable text file (a format widely supported by spreadsheet applications). Each line in the file represents a row of the matrix. Each value in this row is separated from the next by a comma. This export mechanism allows the scatter graph to be plotted by an external application, for example, a spreadsheet.

The use of scatter graphs to represent this type of information can be seen in the Knowledge apprentice (Bernstein, 1990). However, in this system the scatter graph portrays relationships between documents generated from full text retrieval processes. In contrast, the relationships shown by the GAT are generated by analysing authored links.

5.3.5. Ordering the documents along the axes

So far we have seen that certain structural objects, defined earlier, can be seen in the scatter graph representation of the hypermedia network. If we order the documents that are applied to the graph axes so that all documents of the same media type are grouped together then we discover two further structural features:

**Vertical bands** - If one particular media group cannot support start anchors then this will show up as a clear vertical band in the scatter graph.

**Horizontal bands** - If one particular media group cannot support end anchors then this will show up as a clear horizontal band in the scatter graph.
Examples of these structural features can be seen in the following diagram (see figure 5.10).

![Diagram of band structures in a scatter graph.](image)

**Figure 5.10 : Examples of band structures in a scatter graph.**

As it becomes possible to generate start and end anchors in more and more media types these bands will begin to disappear. For example, Microcosm can only support generic links from text based material. If the above scatter graph was taken from the generic links in any of the Microcosm hypermedia networks, then vertical bands would be seen in every media group except text. However, as technology improves and the generic link system is extended to other media, the other vertical bands will begin to fill up with points.

5.3.6. Consistency of the scatter graph.

The GAT constructs a link matrix from examining a set of linkbases and a set of documents (which together represent a complete hypermedia network). The link matrix that is produced as a result of this examination is effectively a *snapshot* of the hypermedia network at the instant the examination took place. Changes to either the links represented by the linkbases or the documents will result in inconsistencies between the scatter graph and the network that it represents.
5.3.7. Example scatter graphs.

This section contains two example scatter graphs, produced using the GAT and printed using the Microsoft Excel spreadsheet application. The documents contained in each example have been grouped together by media type. These groups have been labelled on the graphs themselves.

The first graph (see figure 5.11) illustrates the structure of the *Cell Biology* hypermedia network (Hall et al, 1989). The data concerned were originally authored on the Macintosh within the Stackmaker application (Hutchings et al, 1992). As a result it contains no generic links and all link start anchors are located in the text document group. The scatter graph includes examples of both start points and dead ends, two of the structural objects discussed earlier. Both structures are illustrated within the scatter graph (see figure 5.11).

![Figure 5.11: Scatter graph representing the Cell Biology hypermedia network.](image)

The second scatter graph (see figure 5.12) illustrates the structure of the *Yugoslavia* hypermedia network (Colson & Hall, 1992; Hall & Colson, 1992). This was created by members of the Department of History at the University of Southampton. The network is very sparsely linked, although it does contain generic
links and links that have start anchors in non text documents. Once again specific features have been highlighted for clarity.

The two example scatter graphs (see figures 5.11 and 5.12) illustrate a dramatic variation in authoring style. Figure 5.11 depicts a hypermedia network that contains only explicit links (shown by the lack of horizontal runs of points) and all links have been made from text documents. The network is highly connected with an even distribution of links across all text documents, excluding those towards the top end of the text group. These are reference documents and as such don't contain any links emanating from them. In contrast, figure 5.12 shows a sparsely connected network with links emanating from a small number of documents. However, the graph also shows the use of generic links and two links from documents containing images.

![Figure 5.12: Scatter graph representing the Yugoslavia hypermedia network.](image)

The GAT is a proof of concept tool aimed at both authors and users of hypermedia networks. As such it has never been used by any real users of the Microcosm system. Furthermore, it is not included within the supported release of Microcosm. It is included within this chapter in order to illustrate the problems inherent in the understanding, and representation of, complex hypermedia structures associated with open hypermedia systems such as Microcosm.
5.4. Active path guide

The majority of navigation tools provide users with advice that allows them to plan their journey one document at a time. For example, the history filter (see section 4.1) informs them of which documents they have already seen, allowing them to decide not to see them again. The local map filter (see section 4.3) illustrates the connections between the current document and others, allowing users to choose the next document to see from all those linked to the current document. Finally the memory filter shows them whether they have already seen the documents at the ends of the current links, again easing the choice of which document to see next.

The only tool that provides users with a complete pathway to follow is the mimic system (see 4.2). This provides users with static pre-authored points of view of a dynamic hypermedia network. This static nature will remain hidden unless one or more of the following scenarios occur:

- One of the documents that is part of the guided tour is removed from the hypermedia network. The mimic system could simply miss out the removed document, but the lack of continuity produced by this action may be apparent to the user.

- The content of a document within a guided tour is changed. The mimic will still present the document to the users, but they may be confused by the changes.

- Users following a guided tour become frustrated because they know that it is not showing them all documents that are relevant to the tour's subject.

What is needed is a navigation tool that provides its users with dynamic pathways through the dynamic hypermedia network. The active path guide (APG) is such a tool, it provides its users with a dynamic source of navigation advice based on the movements of previous system users. This advice will adapt to changes in the hypermedia network. For example, if new documents are added it will change to include them. Also, if documents are removed new advice will replace the now redundant advice.
The following sections discuss a variety of issues of the design and implementation of the APG in more detail.

5.4.1. Active path guide implementation

The APG constructs its advice by monitoring users as they move from document to document in the hypermedia network. The information gained from this monitoring is stored in a matrix similar to that implemented in the local map filter (see section 4.3) and the graph analysis tool described earlier in this chapter.

The values stored within the matrix, maintained by the APG, represent the number of users that have moved between the respective documents. For example, the value held at \((A,B)\) represents the number of users that have moved from document \(A\) to document \(B\). In contrast, the value held at \((A,B)\) in the local map filter's link matrix is used to represent the presence of a link from document \(A\) to document \(B\).

The matrix is used to store user generated associations between pairs of documents. As more users move from document \(A\) to document \(B\) the association held at co-ordinate \((A,B)\) in the matrix is increased (by a value of one for each user movement). Therefore, the matrix maintained by the APG is called the association matrix.

The APG uses the values in the association matrix to provide users with a list of the most popular routes from the current document. The popularity of these routes is represented by the strength of the associations held in the association matrix. The APG searches through the column \(x=N\) (where \(N\) is the index of the current document within the matrix). The values held in this column are used to construct a sorted list of documents for the users to see next. This procedure is illustrated in the following diagram (see figure 5.13).
If one of the documents within the hypermedia network has been removed then the values held in the column $y=N$ (where $N$ is the index of the removed document in the matrix) will decrease relative to the other values held in the matrix. The users would see the removed document slip down through the advice provided by the APG. The advice would be adapting to the changes in the hypermedia network.

If new documents are added to the hypermedia network then they will be added to the association matrix. However, as no users have seen them the advice to see them would not exist (represented by zero values in the association matrix). As users discover these new documents, by the use of other navigation tools (e.g. local map filter) the values in the lines $y=N$ and $x=N$ (where $N$ is the index of the new document in the matrix) will increase relative to the other values in the matrix. Once again, the advice provided by the APG will be adapting to the changes in the hypermedia network.

The actual details of the above APG implementation are not the key issue within this section. The design included within this, and the following sections is purely meant to illustrate the proof of concept. The important issue is the use of the state of the hypermedia network, as represented by the previous user's actions, as a source of navigation advice. As the users move around the hypermedia network the state of the system is changed and these changes effect the advice presented to the users. The APG is an attempt to provide the current user with a navigation source that is capable of adapting to the dynamic structure of the hypermedia network within an open hypermedia system. Further research would need to be carried out in order to clarify whether the advice generated by the APG is robust to changes in the

Figure 5.13: APG association matrix

<table>
<thead>
<tr>
<th>Association matrix</th>
<th>(5,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

The user will be advised to see one of the following documents:
3, 0, 4, 5, 2

Current document (N = 2)

Strongest advice

Weakest advice

The actual details of the above APG implementation are not the key issue within this section. The design included within this, and the following sections is purely meant to illustrate the proof of concept. The important issue is the use of the state of the hypermedia network, as represented by the previous user's actions, as a source of navigation advice. As the users move around the hypermedia network the state of the system is changed and these changes effect the advice presented to the users. The APG is an attempt to provide the current user with a navigation source that is capable of adapting to the dynamic structure of the hypermedia network within an open hypermedia system. Further research would need to be carried out in order to clarify whether the advice generated by the APG is robust to changes in the

Figure 5.13: APG association matrix

<table>
<thead>
<tr>
<th>Association matrix</th>
<th>(5,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

The user will be advised to see one of the following documents:
3, 0, 4, 5, 2

Current document (N = 2)

Strongest advice

Weakest advice

The actual details of the above APG implementation are not the key issue within this section. The design included within this, and the following sections is purely meant to illustrate the proof of concept. The important issue is the use of the state of the hypermedia network, as represented by the previous user's actions, as a source of navigation advice. As the users move around the hypermedia network the state of the system is changed and these changes effect the advice presented to the users. The APG is an attempt to provide the current user with a navigation source that is capable of adapting to the dynamic structure of the hypermedia network within an open hypermedia system. Further research would need to be carried out in order to clarify whether the advice generated by the APG is robust to changes in the
hypermedia network, and whether the advice is therefore of any use to the users of the system.

5.4.2. Instant versus delayed response

The APG constructs its advice from analysing the values held within its association matrix. These values are generated by monitoring the user's movements through the hypermedia network. This monitoring can be achieved via two mechanisms:

Active monitoring - The APG is placed in the filter chain (see section 3.3.1), allowing it to actively monitor messages produced by the system in response to users moving through the hypermedia network. The advice generated from the association matrix will change instantly to reflect their movements.

Passive monitoring - The APG is not in the filter chain, its place is taken by a simple monitoring filter. This filter records every message that it receives in a set of unique log files. One log file is generated for each user for each use of the hypermedia network. The advice generated by the APG is not affected by the user's movements until the log files are analysed at a later stage.

The APG uses the 'IS.OPENING' message to track the users as they move from document to document through the hypermedia network. In the first mechanism, the APG receives these messages directly from the filter chain. However, in the second mechanism it must process the log files produced by the monitoring filter before it can obtain these messages. The remaining information (held within the log files) is essentially superfluous (see next section).

Of the two mechanisms the passive monitoring is the more flexible. This flexibility is a result of the extra information that the monitoring filter stores in the log files that it generates. The following section discusses the use of this additional information.

5.4.3. Advice construction

The previous section discussed two processes for constructing the association matrix within the APG. Both provide sufficient information for a simple implementation of the APG (i.e. one which only uses the 'IS.OPENING' message). However,
associations generated by only considering the 'IS.OPENING' messages are susceptible to the following problems:

1. No information relating to the quality of the user's movements is recorded. For example, it could be argued that a movement from document A to B to C is better than a movement from document A to B then back to A before moving to C. The first route shows a positive navigation through the hypermedia network. In contrast, the second indicates that the user could have made a mistake moving to document B. Realising their mistake they moved back to document A then on to document C.

2. No information relating to the user's reasons for their movements is captured.

3. No information is captured about how the user has moved from document to document. Users can move through the hypermedia network via a variety of mechanisms (i.e. following links, using a navigation tool or directly opening a new document).

4. In certain circumstances the real movements of a user are not captured at all. For example, Microcosm allows a user to have multiple documents open on the desktop. If a user opens document A then document B, the APG records this event by increasing the value held at (A,B) in its association matrix. If the user now opens a third document C, the APG records this by increasing the value held at (B,C), because the user opened document C directly after opening document B.

Is this correct? The APG monitored three 'IS.OPENING' messages, one for each of the three documents. It updated the values held at (A,B) and (B,C) as this was the order that it received the 'IS.OPENING' messages. However, in reality the user may have opened document A then document B. On reading B they discovered that it contained nothing of relevance. They then went back to the already open document A and followed a link causing document C to open.

Given this actual list of events the APG should have increased the values held at (A,B) and (A,C), instead of (A,B) and (B,C), in its association matrix.
The last problem illustrates the lack of information provided by only analysing the 'IS.OPENING' messages. However, the Microcosm system produces a plethora of other messages, all of which are available to either of the advice construction mechanisms discussed earlier. These messages can be used in the following ways:

- The way in which users move from document to document (problem 3) can be obtained by examining other messages in the system. For example, following a link will result in the production of a 'FOLLOW.LINK' message.

- Examining more than one movement (i.e. whole pathways) will provide more information to the APG, allowing it to ascertain the quality of movement (i.e. solving problem 1). For example a movement from document A to B to C is of higher quality than A to B back to A then to C, as the user has retraced their steps during the movements.

The above two points illustrate ways in which the additional information transmitted through the Microcosm system can be used to enhance the advice provided by the APG. The serial nature of the Microcosm message system coupled with its co-operative message passing means that the APG will not always be able to obtain the information that it requires in order to advise the user. As a result of the problems inherent within the Microcosm communication model (see chapter three) the APG was left as a proof of concept tool. Therefore the mechanism used to combine the information extracted from the system, by the APG, is not discussed further in this thesis and is a separate area of research.

5.4.4. Levels of advice

The APG stores its association matrix in a file between sessions of use. Furthermore, it uses the layered ini. file system (see section 3.3.5) to store the name of this file together with other user configurable settings.

The use of these ini. files allows the APG to provide advice at one of three levels; user, application or system. The advice provided from the system level is not used as it contains too much redundant information, this problem is discussed in the local map filter section of section 4.3.
The APG uses both the user and application levels to store movements of users in previous sessions. In addition, it begins each session with an empty association matrix which it uses to store the current user's movements in the current session. The introduction of the empty association matrix provides the following three advice levels:

**Public advice** - Generated from an association matrix holding associations constructed by all users in all previous sessions for the hypermedia network of the current application. This is stored in a file whose name is held at the application level in the ini. file system.

**Private advice** - Generated from an association matrix holding associations constructed by the current user in previous sessions using the current hypermedia network. This is stored in a file whose name is held at the user level in the ini file system. This advice is combined with the public advice at the end of the session.

**Current advice** - Generated from movements by the current user in the current session for the current hypermedia network. This association matrix is not stored between sessions, instead it starts empty at the beginning of a session. As the user moves around, it is filled up with new associations generated from monitoring their movements. This advice is combined with the private advice at the end of the session.

The following section discusses the use of each of these three advice levels. In addition, it presents a mechanism for the combination of these levels into one source of advice for the user.

### 5.4.5. Advice combination

The APG presents users with advice constructed by combining the advice produced by the above three association matrices. Values from each matrix are multiplied by a weight for the respective matrix and then summed together. This process is illustrated by the following pseudo code (see figure 5.14).
The above algorithm shows that each association matrix has a corresponding weight. These weights can be used by the user to vary the influence of the advice contributed by each association matrix. For example, if the user does not wish to use the current advice then the weight for that matrix will be set to zero. Furthermore, if they wish the advice from the private association matrix to be stronger than that from the public association matrix the weights can be adjusted accordingly.

The overall associations obtained from the algorithm shown above (see figure 5.14) are then sorted into descending order. The document related to the value at the head of the list is the one that the APG most strongly advises the user to see. As the list is descended the strength of the advice to see the specific documents reduces. The following diagram illustrates the advice combination process in more detail (see figure 5.15).
The above diagram (see figure 5.15) illustrates the advice combination mechanism used by the APG. The user has asked for advice about which document to see after the current document C. The advice from the public association matrix suggests that they should see one of the following documents: D, A, E, B, C. The advice from the private association matrix suggests that they should see one of the following documents: E, D, B, A, C. Whereas the advice from the current association matrix suggests that they should not see document B (because of the -ve weight) as they have already seen it. The advice from the three matrices is then combined and sorted, it now suggests that they should see one of the following documents: D, E, A, C, B.

5.4.6. Forms of advice

The APG advises its users which document to see next based on the movements of previous users. It presents this advice in the form of a sorted list of documents, the document at the top of this list is the most popular document to see next. As the list is descended the popularity of the routes to the other documents decreases.
However, if we consider the information that the association matrices contain about pairs of documents then the following types of advice can also be provided:

**Information infilling** - A user has moved from document $A$ to document $B$ by following a link. They then ask the APG whether they could get from document $A$ to $B$ via a sequence of other documents. Examination of the association matrices will yield such advice.

**Path routing** - Consider the scenario where a user is at document $A$ and decides that they wish to get to another document $B$. The advice within the association matrices can provide the basis for the production of such popular paths linking $A$ and $B$.

**Alternative routes** - A user has moved from document $A$ to document $B$. The information within the association matrices is able to provide a list of alternative documents that the user could have seen prior to $B$ instead of $A$.

Each of the three advice types can be implemented by simple examination of the raw data contained within the association matrices. For example, the alternative routes advice will be generated by examining the association matrix values within the line $y=N$ (where $N$ is the index of the document the user is currently reading in the association matrix). The highest value along this line will be the most popular document to see before the current document.

### 5.4.7. Pre-authored advice

When the APG is first used its association matrices will be empty, it will be unable to provide users with advice. The associations will be constructed over time as users move through the hypermedia network. Herein lies a problem, users cannot request advice from the APG until the association matrices have been at least partially filled.

The problem of initial APG ignorance can be solved by allowing an author of a hypermedia network to provide associations in a similar manner to the way in which they create links. Furthermore, tools can be provided that will examine sources of information such as linkbases, generating such associations automatically.
Association matrices were constructed from log files recorded during user trials as part of a 1st year undergraduate course on 'Cell Motility' within the Department of Biology at the University of Southampton. The results of the APG advice combination algorithm taken from processing these association matrices can be seen at the end of chapter eight, where they are included as part of the Advisor agent test scenarios.

5.5. Summary

This chapter has discussed the results of research into the design and implementation of a set of novel navigation tools. These tools were primarily designed as proof of concept tools aimed at providing users with the extra navigation advice that they require when navigating through large open hypermedia networks. However, during the course of the research two of the tools (the memory and show links filter) were used with real users of the Microcosm system. Furthermore, one tool (the show links filter) has become a part of the supported release of Microcosm.

The separation of links within the Microcosm system aids the graph analysis tool in its generation of a representation of the hypermedia structure. The support for implicit links makes the authoring effort involved in writing large hypermedia document sets substantially less. However, it makes the generation of structure representations a more computationally expensive task.

Tools that attempt to send large quantities of messages to others will highlight the inefficiencies of the communication system (see section 3.3.1.5). For example, the show links filter was initially implemented as a separate filter but was later incorporated within the linkbase. This change was due to the length of time taken to send large volumes of messages between two filters.

The uni-directional nature of the filter communication chain causes problems for most of the navigation tools implemented so far. For example, the show links filter must be before the linkbases in order to communicate with them. The local map must be between the linkbases and the memory filter (see section 5.1) so that it can monitor the messages successfully.
The dynamic nature of the hypermedia document sets created with Microcosm need tools that are capable of providing dynamic navigation advice to users. The memory and show links filters provide such advice by utilising the dynamic nature of the link support offered by the Microcosm linkbases. Links can be added to the hypermedia document set and users will instantly benefit from this addition using these filters.

Microcosm allows the construction, and use, of new navigation tools without needing to change the system itself. Such a feature provides a very powerful research platform and has helped greatly in the development of the tools described in this chapter.

The contents of both this chapter and the preceding one have discussed the design and implementation of seven separate navigation tools. Each tool provides the users with a separate point of view based on its local knowledge. Should the user wish to obtain an overview of their current situation (as regards their position and projected route within the hypermedia network) they will have little choice but to query all the navigation tools currently running. Failure to do so could result in them missing the single crucial piece of advice that guides them in the right direction.

It would seem that the user could benefit from a tool that provides them with this overall navigation strategy. The tool would be given the responsibility of constructing the overall strategy from the disjoint advice extracted from the disparate navigation tools. Furthermore, the tool would need to be able to function within the dynamic constraints imposed by both the hypermedia network and the open hypermedia system. Agents are perfectly suited to such a dynamic semi-autonomous task. The following chapter briefly reviews the topic of agents, providing a discussion of other agent systems as a background to the further discussion on the merging of agent and hypermedia functionality.
6. Agents in the interface

agent n. a person or thing that exerts power or has the power to act; one entrusted with the business of another; a deputy or substitute. - agency n. instrumentality; a mode of exerting power; office or duties of an agent.

This chapter begins by discussing the increasing need for agent style applications, acting on behalf of the user, within the interfaces based on the desktop metaphor. This is followed by a literature overview from the area of agent based systems.

The chapter then continues to describe how agent based software could be applied to the area of hypermedia. An open hypermedia system (in this instance Microcosm) is examined and the case for viewing it as an agent community is put forward. Three major roles for agents within a hypermedia system are discussed. Finally the chapter concludes with a summary of its key points.

6.1. Why do we need agents?

Computer based assistants, or agents, are coming within reach of most computer systems. With them comes the promise of intelligent helpers that can find new information and perform boring, repetitive tasks. In addition, such agents could provide advice about specific aspects of their own computer environment.

When a user sits in front of a modern computer they often interact with it using an interface based on the metaphor of a real world desktop. They operate the computer, dictating what tasks it performs and when. If they wish to search through a database for a specific item they must construct a query, present it to the application and await the results. Consider the scenario where the database is remote and contains material that is continuously updated (e.g. a news wire service). Each time the user wishes to query the database they must repeat the process outlined above.

The problem is that the desktop interface, as it stands, is user driven; the user is the main source of activation. Palaniappan in (Palaniappan et al, 1992) discusses a shift from the current user centred sifting and sorting behaviour (discussed above)
to a specifying and delegating behaviour. This shift will bring the biggest gain in productivity to people who deal with large volumes of dynamic data by being able to perform repeated tasks.

How would we instigate such a shift in behaviour? Laurel (Laurel, 1991) views the interface as a stage. The user and the applications within the computer are simply actors acting out a dynamic play. The interaction between the user and the applications shapes the course of the play. At each stage during the dialogue actions by one or the other turn possible events, through probable events, into definite actions. Palaniappan (Palaniappan et al, 1992) shares a similar concept of the current interface, viewing the applications as operators, as it is these which operate on the user's data allowing them to interact with the system. They both argue that the current interfaces should be extended to encompass agent type elements. Such extensions would provide the user with the ability to move towards the predicted shift in behaviour.

The following areas of user interaction can benefit from such an agent based approach to the user interface:

**Automation** - An agent could be created that would periodically sift through an information source (e.g. a news feed). The agent would be provided with a set of topics to look for. If anything appeared within the bounds of its search parameters it would report its findings. This would have the same net effect as the user sorting through the information source, but without using up their time, providing the user's requirements are particularly well defined.

**Monitoring** - An agent could be created that would constantly monitor a dynamic object, reporting back any changes to the user. For instance, the user could set up an agent that continuously monitors the amount of free disk space. The agent could either constantly report free disk space, or more likely, it could report when the free disk space dropped below some pre-set level. The agent would take the responsibility of processing the menial task away from the user.

**Future execution** - The user could tell an agent to execute its task at some pre-defined point in the future. For example, if the user knows that a database will be updated one month in the future they can assign an agent the task of returning the results of a query after the database has been updated. The
user is delegating the responsibility of the database query to the agent to execute in the future.

**Services** - Agents will eventually be able to offer additional services to the user. For example, consider a user who is attempting to keep track of the changes to a jointly authored document set. This requires them to query the file system at set intervals. Furthermore, they must remember the previous state of the system in order to be able to detect any changes. By shifting these tasks to an agent the burden of both tasks is removed from the user. The agent will be charged with the responsibility of continually monitoring the set of files, reporting any changes to the user.

To summarise, agents allow the user to shift the burden of repetitive work to the computer. They also allow the user to allocate future tasks in the present and will eventually provide the user with new services.

### 6.2. Implementation strategies

There are four main mechanisms that can be employed in the implementation of agent based systems. All of these mechanisms incorporate a degree of *intelligence* allowing the agents to react in a reasonable way to any unforeseen circumstances that may occur. The following four sections deal with each mechanism in more detail.

#### 6.2.1. Artificial intelligence

The majority of agents implemented so far have some form of intelligence built into them. This section discusses three possible methods by which such intelligent behaviour can become part of an agent system.

6.2.1.1. Learning

Four main types of learning are generally recognised within the field of artificial intelligence. Cohen in (Cohen, 1982) states these as being:

**Rote learning** - Memorising new knowledge so that it is available when it is needed at a later point in time.
Learning by being told - The system is given amorphous, general purpose knowledge, or advice, which it must process into a form that can be used in its current environment.

Learning from example - The system is provided with a set of specific examples which it must process to form more general higher level rules that can be applied to its current environment.

Learning by analogy - The system is given some form of knowledge base for a related task and must then produce analogies between this knowledge and the current environment.

The majority of agent systems are a mixture of the learning from example and being told methods. They are given advice (in the form of high level instructions) on the assignment from which they must formulate the specifics. In addition the agents would provide more functionality to the user if they could adapt to the user's needs and methods of working. In this instance it is being given examples of their working practice and must generalise them to fit the current situation. One such system is the COKES system (Kaye & Karam, 1987), which is described in more detail later in this chapter. The main problem with this approach is the large amount of time needed in order to construct a comprehensive enough knowledge base, allowing the agent to cope with the majority of situations.

6.2.1.2. Intelligent knowledge based systems

A primary mechanism used for providing an agent with intelligence is to base its behaviour on rules. A rule base together with corresponding actions (used when a particular set of rules are satisfied) is provided for the agent. Object-Lens (Lai et al, 1988) is one such agent system, the user completes a specified form and hence creates rules for the system to use.

The primary advantage of the rule based approach is the extra flexibility that it gives the system compared with a standard algorithmic solution. The disadvantage is the time needed for the user to create the system's rule base.
6.2.1.3. Divide and conquer theory

Minsky in his "Society of the mind" (Minsky, 1986) suggests that the human mind is a mass of communicating agents, each operating in parallel. Each agent concentrates on a particular aspect of the problem being tackled. These agents are hierarchically organised and can be collected together to form a more powerful unit, capable of completing tasks too complex for any single agent. This theory has been employed to great effect in two agent systems, Neural agents (Adams & Nabi, 1989) and the Playground system (Fenton & Beck, 1989).

Although the idea behind the system is very simple it is technically very difficult to implement. The inter-agent co-operation and the selection of which agents to run in parallel are difficult to ascertain.

6.2.2. Monitoring agents

These agents are, as the name suggests, strictly limited to scenarios where they can monitor a set of objects within their environment. This could be the system time for an alarm signal, or disk space to warn when it has become too low. Examples of such systems can be seen in the UNIX scheduling tools cron or at or Sun's calendar manager (Sun Microsystem Inc., 1990).

Cron allows users to specify both the time and frequency at which certain commands will be performed. The system then monitors the system clock, performing these tasks at their allocated times. In addition the calendar manager, from Sun, acts as an automated agenda/reminder facility, sending the user copies of calendar entries at required times and dates.

The Envoy framework (Palaniappan et al, 1992) also falls into this category, allowing users to specify missions and frequencies with which they must be carried out. This system is discussed in more detail later in this chapter.

6.2.3. Application based approach

This mechanism produces agents that are capable of a great variation in tasks. However, they do not generally have the complexity of the AI based approach. In this scenario the agents are integrated into specific applications. As a result they
have access to increased volumes of knowledge about the tasks at hand. Examples of such specialised agents can be seen in the wizards embedded into the new range of Microsoft software products (Microsoft, 1992; Microsoft, 1993) (e.g. Excel 4.0 and Visual C++).

Such agents can be used within many different scenarios in their respective domains. However, most end users would be unable to tailor these applications and agents to perform the tasks they require because of the complexity involved.

6.2.4. Desktop based approach

This scenario sees agents being integrated into the environment at the desktop and operating system level. Such integration will allow users of applications that are unaware of the agents to access the additional functionality they provide. The NewWave environment (Hewlett Packard, 1989) uses such a technique to provide agents within the Microsoft Windows interface. Users can specify tasks by example, automatically producing scripts that the agents will follow. The scripts record the actions of the user and the agent then recreates these actions by interpreting the scripts each time the task needs to be executed. The Envoy framework (Palaniappan et al, 1992) also attempts to provide a similar level of system wide agent functionality.

The major benefit of such an approach is that the user does not need to learn new techniques in order to automate their tasks. They still interact through the familiar applications. However, this type of agent is susceptible to changes within its environment. For example, if the configuration of the user's machine changes (e.g. files are moved within the file system) then there is a high likelihood that the agent will fail in its allotted task.

6.3. Agent systems

A number of agent based systems already exist, ranging from fully fledged commercial products (e.g. NewWave agents) to research and development systems (e.g. Apple Guides). This section discusses seven examples of such agent based architectures, attempting to categorise them within the classes discussed in the previous sections. These systems are:
• Apple Guides
• COKES: co-operative knowledge based assistants
• Envoy framework
• ITX system
• Neural agents
• NewWave agents
• Object lens
• Playground system

These examples cover a wide range of tasks and are implemented within a variety of systems. Furthermore, they cover systems that range from research platforms to fully commercial systems.

6.3.1. Apple Guides

The principle objectives of the Apple Guides experiments (Laurel, 1990; Laurel et al, 1990b; Laurel, 1991; Oren et al, 1990) were to help achieve media integration by providing a media neutral navigation aid and to reduce the cognitive load placed on users as they navigate through the hypermedia system. Furthermore, the developers of the guide agents attempted to achieve the following three aims:

• The integration of searching and browsing activities under a single interface umbrella.

• Provision of a simple, effective and engaging means for users to access the hypermedia database.

• The development of alternatives to the spatial metaphors used in the organisation of information.

The authors found that the use of interface agents reduced the cognitive load for users of the hypermedia system. These agents produce dynamic tours derived from a combination of information about the document set, their knowledge about a
user's movements and their own pre-programmed points of view. The latter are calculated by key-wording every document from a limited vocabulary and programming the guides to respond to certain subsets of the same keyword vocabulary.

The guide agents took the form of ten representations of prototypical characters from the period in American history featured in the hypermedia database (i.e. USA history between 1800 and 1850). They were provided as an additional navigation aid to the other aids already in the system (including timelines, animated maps, article indexes and manually created guided tours). The agents were aimed at helping to solve the two classic navigation problems, both of which increase the cognitive load to the user. An increase in cognitive load decreases the usefulness of the system, as the user has to expend more effort on managing their navigation through the hypermedia network and so will not absorb so much of its content. The agents were designed to:

1. Reduce the possible routes from any particular anchor. This would eliminate the resulting search sub-problem in which the user must determine the optimal next choice from the route list.

2. Reduce the effect of disorientation due to a rapid succession of jumps between documents.

The guide agents construct an $n$ dimensional vector by examining the keywords assigned to the user's current document and its own point of view keywords. This vector is then used to promote certain documents from the complete document set. It is this reduced sub-set that is provided to the user as a form of navigation advice. As the user moves around, the vector and hence the advice is continuously updated, providing new documents at each stage. The experiments showed that even characters with very minimal dramatic traits can successfully suggest human like qualities to the users.

The knowledge processed by the guide agents can be divided into two areas:

**Nature** - Learning that has been constructed throughout past events. In this case the process of key-wording the document set and providing the agent with its point of view.
Nurture - Learning constructed from examining current events. In this case the process of the guide adapting its advice to the movements of the user and its perception of their requirements.

The Apples Guide agents would seem to fall under the AI implementation category. They are provided with a simple rule base in the form of their point of view keywords. In addition, the manner in which they interpret these rules is implicit in the mechanism they use for constructing the advice vectors.

6.3.2. COKES: co-operative knowledge based assistants

The COKES system (Kaye & Karam, 1987) contains a network of co-operating knowledge-based assistants and servers. These knowledge based agents contain both factual and procedural knowledge and are capable of making use of existing conventional office technology. The authors of the system felt that the current technological tools used in the office were usually designed to support low-level tasks and activities. The integration of these into a high-level task is left to the user. For example, a message system supports the sending and receiving of messages but does not do anything to relate those messages to their function within the organisation.

The agents can support concurrent multiple tasks with facilities for interruption and resumption of those tasks. They co-operate in solving problems by passing messages between themselves. The agents are divided into two classes, these are:

- **Assistants** - Devoted to individual users within the office. They hold knowledge bases dedicated to the representation of their respective users.

- **Servers** - Dedicated to specialised areas of organisational knowledge or functionality.

The authors felt that their agent system should be capable of fulfilling the following four objectives:

1. It should be capable of applying general policy and procedures to new situations, with the possibility for users to specify variations.
2. It should be capable of reasoning and answering questions about the organisation and explaining its own actions.

3. Assistants should co-operate to solve problems independently of their users whenever possible.

4. Whenever possible (when high level activities involve structured use of existing office tools) the necessary access to these tools should be automated as part of the high level support.

All objects that exist within the office are represented within the system as frames. For example, a personnel object may possess such attributes as classification, position, office location, electronic mail address and personnel identification number. In addition, the system supports the concept of class frames; these are used to represent groups of similar objects. For example, all software projects could be represented by a class frame and a particular software project would be represented by a particular instance of that class frame. The instance would inherit all the properties of the generic software project class frame.

Every user of the system is provided with a knowledge based assistant devoted to their use. Each assistant, though identical in basic structure to the others, has a knowledge base appropriate to the needs, authority and responsibilities of its user. In addition, the system also includes a collection of specialised servers, each containing knowledge of general interest to the office population. The architecture of the servers is identical to that of the assistants. Whenever possible servers are constructed that provide interfaces to existing conventional office technology such as a database. Such an overall structure allows users to take advantage of the existing technology through a consistent interface.

The general principle that information should be stored where control over its evolution is located is supported by the system. For example, information about a particular project will be stored in the project manager's personal assistant.
Public knowledge, located on any assistant or server, is available to other assistants through message passing. The overall structure of the system can be seen above in figure 6.1. Finally, the authors of the system foresee a future where individual workstations would be routinely equipped with knowledge-based systems devoted to the needs of their primary user.

The system falls into the AI category, exhibiting significant similarities to Minsky's "Divide and Conquer" approach. In addition, the assistants contain knowledge bases and inference engines placing them into the AI learning category.

6.3.3. Envoy framework

This system (Palaniappan et al, 1992) attempts to provide a system wide environment into which existing applications can be slotted. The applications will hence gain the extra functionality provided by the Envoy system.

The framework is based on the delegation metaphor which the authors argue offers significant benefits over other models. Users specify their requirements (or missions), leaving the envoys to work with the application and hence produce results which are then presented via a suite of informer applications.

The envoys can interact directly with the front-ends of certain applications (or operatives) to provide the user with interactive reports. This has the added benefit
that users can manipulate the mission results using the native applications. The different areas that make up the framework can be seen in the figure 6.2.

Figure 6.2: The Envoy framework reproduced from (Palaniappan et al, 1992)

The unshaded region of the diagram indicates the components of the system that the user interacts with. These are:

**Operative front-end** - Users interact with these in the same way that they would with standard applications. In addition the envoy can use these to interact with the user.

**Mission summary** - This can be displayed at any time by the user. It shows the progress of the current envoy mission or missions. Furthermore, it shows the completed past missions and the future missions still awaiting scheduling.

**Informers** - These communicate the results of any completed envoy missions to the user. The user specifies which informer mechanism to use for each mission when that mission is set up. The informers support the idea that they should find the user, not the other way around. For instance, they could inform the user via E-mail, FAX or by using an operative front end.

The shaded region of the diagram indicates the components that interact with the underlying system, these are:

**Operative back-end** - These enable the operative front-ends to interact with the rest of the system.
Envoys - These are the user's agents. In order to provide effective support they must have access to all resources (both local and remote). The envoys interact with all parts of the system and provide the main functionality.

Bureau chief - The central administrator of the system. As such, only one exists on each local area network. It is responsible for scheduling and tracking the progress of the envoys.

Only certain types of applications are suitable for conversion into operatives. For example, text or graphics editors are not suited as they require too much interaction with the user. One of the guidelines that the designers of the envoy system stressed was that envoys must save time for the users. They must not make extra work for the users by requiring them to learn new complex tasks and concepts before they could be used. In addition, they should fit seamlessly into the user's computing environment taking advantage of any existing applications.

The authors believe that if a system similar to their envoy/operative system is integrated into standard operating environments, then users will be able to benefit from a host of agent-aware applications. They will no longer have to be information seekers, instead the information will find them.

This system falls into the desktop based approach category, providing an agent based framework into which agents and agent-aware applications can be slotted. Any further classification would depend upon the implementation of both the aforementioned process groups.

6.3.4. ITX system

This system (Lee et al, 1993) attempts to provide a framework, similar to that provided by the Envoy framework system, in which distributed co-operative tasks, or agents, can operate. The authors envisage that the user will have a specific task that they wish to carry out (for example, video conferencing or computer supported co-operative work). The agent will first attempt to allocate the required resources for the mission. If this is not possible then the agent will either inform the user, wait for the resources to become available, or attempt to change the resources required to complete the task.
The area that the authors of the ITX system concentrated on was that of teleconferencing. Each participant needs both audio and video connectivity. Each user that requests a conference instructs their agent to contact the other parties via their agents. The agents then attempt to allocate the required resources or negotiate a suitable compromise. In addition, agents must continuously monitor the system to make intelligent adjustments throughout the duration of the calls. For example, if one user's video fails it may force them to converse with all the other participants by audio only. Such a scenario would require the agent to re-negotiate its hardware requirements.

The system framework integrates feedback control with transaction processing techniques allowing it to provide reliable interactions between the agents and the shared resources. The distributed agents maintain co-ordination and consistency using a system of shared data objects. The system uses a fixed point criterion to ensure that it is always in a stable state and fulfils the requirements of all of its users. This works by iterating through an observation and action loop until the system requirements have not changed for two whole iterations (a mechanism similar to that by which people barter for goods).

The agents can be categorised in both the AI and the application based classes. The former class because they contain rules for:

- The initial resource requirements.
- The mechanism by which they negotiate in cases where a compromise must be reached.

The latter class because an agent must be aware of the specific requirements of the applications in question.

6.3.5. Neural agents

Many theories have been advanced to allow the modelling of complex heuristic behaviour. These range from Minsky's "Society of the Mind" (Minsky, 1986) to neural network theory. Many theories share a common architecture consisting of many highly connected simple elements that together exhibit a highly sophisticated macro behaviour.
The Neural agent system (Adams & Nabi, 1989) provides a framework that combines the high level interaction of collaborative rule based agents with the topological connectivity of neural networks. It implements decision making networks based on agent objects. These are connected like neurons within a neural network but are in fact highly specialised local experts.

The authors of the Neural agent system quote Minsky's definition of agent and agency as follows:

**Agent** - Any part of a process of the mind that by itself is simple enough to understand, even though the interactions among groups of such agents may produce phenomena that are much harder to understand.

**Agency** - Any assembly of parts considered in terms of what it can accomplish as a unit, without regard to what each of its parts does by itself.

The authors then continue to stress that their agents have no limit to their individual complexity and that they do consider an individual agent's functions when considering an agency. The stipulation that an individual agent can be as complex as need be is also a key element in the other systems examined in this chapter.

The agents within the Neural agent system communicate with each other by means of a client-facilitator-consultant (CFC) model. This was designed after enumerating the roles humans play in their collaborative efforts. The authors discovered that most roles could be described in terms of their model. The three roles are defined as follows:

**Client** - object or agent that receives advice or other services from an agent.

**Consultant** - agent that provides advice or other services to a client.

**Facilitator** - agent that enlists advice and/or services of other consultant agents in order to provide advice and/or services to its client.

All advice that is exchanged between agents is packaged with a confidence value, that is an indication of how confident the consultant is that it is correct. Furthermore, the client weights it depending on how reliable it believes the consultant to be. The Neural agent system is constructed from three types of object. These are:
Rules - objects that test a condition and execute an action if that condition is true. They can monitor other objects ranging from aspects of the system to parts of the agent to which they belong.

Agents - an aggregation of rules with a common host and shared memory local to the shared rules. Agents utilise their rules to respond to interesting aspects of their surroundings.

Collaborating agents - are a specialisation of the agents that can form collaboration topologies according to the previously mentioned CFC model.

An example of the Neural agent type theory can be seen in the Petworld system (Coderre, 1989). Three types of object inhabit the Petworld simulation, pets, trees and rocks. Pets eat fruit from the trees and collect rocks to build nests. Pets also fight each other, experiencing both fear and hunger, reacting accordingly. The Neural agent system provides the underlying rule based framework that allows the Petworld simulation to exist.

One of the overriding beliefs held by the authors of the Neural agents system is that agents demand productive environments. Furthermore, they require seamless integration into software systems before they can be widely used as a programming paradigm.

The three components (see above) within the system can be used to construct complex topologies which can alter their connections, and hence the topology, to suit the circumstances of the current environment. The agents within the Neural agent framework fall into the AI category as they make heavy use of rules, either within themselves, or in the rule objects.

6.3.6. NewWave agents

The NewWave system (Hewlett Packard, 1989) exists as a layer on top of the Microsoft Windows environment. It provides the user with an object oriented model for documents, tools and tasks. Document objects are associated with the applications that are able to manipulate them. Activating document objects activates its associated application. Tools represent applications that do not really have a document type to manipulate, for example calculators, clocks and printers. In
addition the system introduces an agent object which is associated with documents called agent tasks.

Agent tasks are best suited to the automation of complex, system wide activities. They can be performed instantly (by dropping them on the agent icon) or scheduled for later execution by linking them to time and date events. Users create them by creating an agent task object. On to this they record the sequence of events that they wish the object to represent, and finally they compile the task so that it can be performed by the agent. The un-compiled tasks exist in an English like language allowing users to edit and manipulate them.

When an agent task is scheduled for future execution the user delegates the responsibility of that task to the agent scheduler. The scheduler can either execute a particular task at one specified time and date, or repeatedly over a set period of time, or at set intervals. Once a task has been registered with the scheduler, the user can re-schedule it or even remove it completely. The agent scheduler can display a list of currently scheduled tasks (in a similar manner to the mission summary discussed earlier within the Envoy framework). Lastly the system can generate a log of all agent activity, allowing the user to monitor exactly what actions it has performed.

Agent tasks can be made relatively robust. For example, if a task involves a document that has been moved since it was recorded then the agent can be told to prompt the user for its new location. The agents within the NewWave environment can be categorised under the desktop based approach as they can operate over all applications within the desktop environment.

6.3.7. Object lens

The Object Lens system (Lai et al, 1988) allows users to modify a series of templates for various semi-structured objects (representing people, tasks, products, messages and many other types of information). These templates provide the users with a mechanism that facilitates the creation of their own co-operative work applications. Furthermore, they are able to create semi-autonomous agents that can automatically process information (derived from the template based objects created by the users) in different ways at different times.
The authors claim that the combination of these primitives provides a single consistent interface that integrates the facilities required for object oriented databases, hypertext, electronic messaging and rule based intelligent agents.

The agents provide a natural mechanism for partitioning the tasks performed automatically by the system. Agents are triggered by events such as the arrival of new mail, new objects appearing in a particular folder, the arrival of a specified time or an explicit selection by the user. The authors of the system view the agents as *semi-autonomous* for the following reasons:

**Autonomous** - Once created, they can take actions without the explicit attention of a human.

**Semi-autonomous** - They are always able to be controlled by the user and they often refer objects to users for their attention instead of taking actions themselves.

When an agent is triggered it applies a set of rules to a collection of objects. For example, an agent could be triggered by the arrival of new mail. When such an event occurs it could invoke the rules in its rule base to sort the mail on behalf of the user.

Since agents and the rules themselves are objects, users can look at and modify them with the same template based user interfaces that they use for all the other objects within the Object Lens system.

The agents form a part of the larger Object lens system which is a computer environment for the Xerox 1100 series of workstations. They can be categorised under the AI class as they contain knowledge bases in the form of rules and associated actions.

**6.3.8. Playground system**

This system (Fenton & Beck, 1989) shares a similar foundation to that of the Neural agent system taking its impetus from Minsky's "Society of the mind" work. It aims to solve some of the problems usually inherent in programming languages aimed at children (e.g. primitive control and data structures, indirect user interfaces and artificial syntax). It is a child oriented programming language, using objects to
structure data, providing a modular control structure together with a direct manipulation interface and finally an English style syntax.

The system presents the user with an interface similar to that of an object based drawing package. The objects represent creatures (each having sensors, effectors and processing elements) and are contained within a window called a playfield. It is this playfield that mediates the interactions between the creature objects.

An object can control how it moves and how it appears to the user. Furthermore, it can sense the presence of other objects in the playfield (locating nearby objects, objects of a certain type or objects that overlap etc.).

If users select an object the system will allow them to edit its agent rules. These describe cause and effect relationships that apply to the simulation. All rules run in parallel, when an appropriate set of circumstances occurs the agent rule triggers starting a designated sequence of operations. However, if an agent rule does not specify any conditions it will run continuously (e.g. the agent rules that dictate the shape of the objects).

Agent rules are atomic independently executing entities that are sensitive to a particular set of stimuli. These rules can be excited or inhibited by other agents, achieving a powerful control of the agent. Agent rules can also contain state information that persists over time.

The authors of the system believe that such semi-autonomous agent systems can be implemented using two different methods. These are:

**Pulling** - An agent tests for conditions explicitly hoping to detect appropriate configurations in others that would enable it to continue.

**Pushing** - An agent that generates an event must notify all other agents that are interested.

Each method has its advantages and disadvantages. Pulling can waste time checking for conditions that have not changed. Pushing requires overheads to track the dependency relationships among objects. The Playground system supports a mixed mode strategy. Each agent can test for the events and situations under which it would be run. When a rule is posted, the condition part is examined. If appropriate the rule will be registered so that it only runs when the appropriate
events take place (e.g. user interface events). In addition, one agent may trigger another agent explicitly, an operation similar to sending a message in a conventional object based system.

This system can be classed as an AI style application as the agents contain rules and associated actions. However, it differs from a standard rule based system in that the activation of one rule can directly affect the activation of the others.

6.4. Open hypermedia system: an agent based view

If we consider a large dynamic open distributed hypermedia system then the advantages of agents can be clearly seen. The following sections discuss three ways in which agent based technology can be employed within such hypermedia systems.

6.4.1. Maintenance

Hypermedia systems provide the ability to create an umbrella of links over a collection of documents. Open hypermedia systems provide the same functionality but present the user with a dynamic set of both links and documents. The dynamic nature of the data (both links and documents) can cause many problems. For example, links can be rendered useless if the documents they connect are removed or even slightly changed. The majority of open hypermedia systems gloss over such editing problems.

One possible solution could be provided by an agent whose sole task is to check links against the document set. It could search for links that no longer make sense (e.g. either the source or destination anchors have changed). In the case where the link anchor has been moved within the document, it can be fixed by the agent. If more serious problems occur, such as the document being deleted, then the agent could request help from the user. However, if the agent is provided with knowledge of document movements within the file system, these more serious problems could also be solved solely by the agent.
6.4.2. Monitoring and searching

The document set and links contained within the open hypermedia application can be viewed as a database in their own right. Given that the contents of both will be highly dynamic then it seems reasonable to employ the agents as information locators/monitors. For example, users could set up agents that have the sole purpose of monitoring the documents, looking for certain keywords or contents. If the required information is located then the agent could direct the users to the new documents or create links to them. Furthermore, agents could be constructed that monitor the links themselves. If a link is created (e.g. by another user) connecting an interesting document to some other information then the agent can inform its user.

Monitoring agents would provide the greatest benefit to groups of collaborative users editing a shared corpus of hypermedia information. In addition, they would prove useful to users who repeatedly need to search for the same information within a dynamic hypermedia document set.

Initially agents would be limited to looking for text based information (as proven search techniques only really exist in this area). However, there is no reason why they should not be extended to look for information in other media types as soon as search techniques become available.

6.4.3. Navigation

Many tools have been provided to help users navigate around hypermedia document sets (see chapter two). An advanced example of such a navigation device can be seen in the Apple Guides experiments (Laurel et al, 1990b). The Guides agents provide dynamic tours that are generated by examining the document set and combining this with their built in points of view. Oren (Oren, 1987) summed up the role of such agents in the following paragraphs:

'I suggest that an agent in a hypertext database need not have a deep understanding of the documents themselves, so long as it has a model of the conventions of hypertext links, and the database follows these conventions uniformly. A parallel would be the human reference librarian who does not comprehend the material in articles being sought, but does understand the conventions of card catalogues,
abstract collections, citation indexes and bibliographic references. Because these relations can be made explicit in hypertext they can be utilised without, for instance, having any deep comprehension of the meaning of any article title. Such a hypertext agent would always be delving ahead of the user, calling up documents suggested by link patterns, rating them for relevance to the current topic and readying them for display. The user could either accept these suggestions or continue with a query strategy.

The navigational problem can be sub-divided into three areas. A navigational agent within a hypermedia system should be able to fulfil the following:

1. Reduce the number of possible destinations for a user to choose from when too many are offered. As the size of the hypermedia data space increases so, theoretically, does the number of links contained within it. The number of choices available from any given selection quickly becomes unmanageable for the user. The main function of the navigational agent is to reduce the number of choices by promoting choices in the next document list.

2. Expand the number of destinations for a user to choose from if none are apparent. This would be achieved by the agent querying the various sources of navigation information already in the system. Furthermore, the agents could enhance the number of destinations by searching the documents and links for certain key information.

3. Gather together and intelligently combine existing sources of information (e.g. navigation) already in the system, presenting the user with one combined source of information instead of many. A discussion on the implementation of such an agent can be seen in chapter eight.

The latter point provides a possible mechanism for achieving the former two points. For example, if the user reaches a state where they cannot find any more information, the agent could automatically query any sources of information within the system. The results from these queries could then be presented to the user allowing them to continue with the exploration of the document set.

Alternatively, should the user reach a point where they are overwhelmed by possible destination documents, the agent could query other sources of information allowing the destinations to be ranked in order of importance. This ranking would
have the effect of lessening the number of choices offered to the user. A description of an implementation of such an agent (within the Microcosm system) can be seen in chapter eight.

6.4.4. Microcosm as a community of agents

Microcosm exists as a set of processes that are able to communicate with each other by means of a custom communication system (see chapters three and seven). Each process carries out its own specialised task, the results from one agent being transported to the next using a system of messages. It could, therefore, be argued that the Microcosm system is actually a co-operating community of agents working together, breaking down the complicated tasks presented by a hypermedia system into simple tasks in the manner suggested by Minsky (Minsky, 1986). Processes within the Microcosm system display the following forms of agency:

- When document viewers are started they automatically query any linkbases within the system to locate any buttons for their current document. The viewer is acting on the behalf of the user, removing the need for them to search for buttons themselves.

- The history filter monitors the communication system looking for 'IS.OPENING' messages generated when any document is opened. It maintains a list of the documents that the user has opened. Furthermore, it also looks for the 'CLOSE.FILE' messages that are generated whenever the user closes a document. The history filter can tell the user which documents they have seen and which are still available on the desktop. The user has delegated the responsibilities associated with these two tasks to the history filter.

- The show links filter allows users to search for links from within large selections of text. It takes the selection made by the user and divides it into single words and word pairs (see section 5.2). These are then sent to the linkbase which searches for links from the subselections given to it. The alternative solution would be for the user to select each word in their selection and search for links individually. The show links filter has been given this task, thus removing the burden from the user.
• The mimic system allows the user to replay tours (see section 4.2). Throughout the duration of these tours it is able to take over the responsibility of opening, closing, locking and unlocking document viewers from the user. The author of a tour delegates the responsibility of controlling the document viewers to the mimic system for the future.

• The local map filter (see section 4.3) monitors the message system, allowing it to generate its maps without placing any additional burden on the user. Furthermore, it can take over the responsibility of changing its display every time it detects new links or notices that the document the user is interested in has changed.

• The memory filter (see section 5.1) records which documents have been seen by the user. It uses this information to group and sort the routes from any particular link anchor that the user may choose. It shows the users which routes will lead to documents that they have already seen and approximately how long ago they may have seen them. It removes the burden of having to remember which documents they may be about to see again, making their choice of which route to follow easier.

• The active path guide (see section 5.4) monitors which documents are seen, and in what order, by all users of the system. It uses this information to provide a ranked list of routes that the user may follow from any particular document. These routes are ranked by their popularity amongst the previous users of the system.

More generally, the viewer class of process is responsible for taking information in a specific format and displaying it to the user. They also provide users with a front-end to any form of link query supported by the filter class of process. They translate user actions into information understood by the filter agents. For example, if a user wishes to follow a link on a word in a text document they:

• Open the text document in a text viewer.

• Select the word to act as the key when the search for the link is executed.
• Select the 'Follow Link' menu option, informing the viewer that it should take the user's selection and pass it on to a filter agent that can search for links.

The filter agents take hypermedia queries directly from the user, or indirectly via a viewer. These are interpreted and executed on their local knowledge (e.g. links in a linkbase). The results are returned to the user directly, or indirectly via another filter agent. In addition, they are able to store user actions for later processing, playback and interpretation (e.g. the history and mimic system).

6.5. Summary

Increased computing power is bringing computer based assistants, or agents, within the reach of most systems. It is envisaged that they will bring a shift in user activity from searching for, and sifting, data towards a specifying and delegating behaviour.

Agents are able to provide three modes of interaction for the user; automation, monitoring and future execution. In addition they will be able to offer users new services. The four main mechanisms that can be employed in the implementation of agent systems are artificial intelligence, monitoring, application based approaches and desktop based approaches.

A range of agent type systems already exist, from fully commercial products (e.g. NewWave) to research projects such as the Apple Guides experiments. The majority of these systems employ mixtures of the four mechanisms listed previously in their design and implementation.

One of the overriding beliefs held by the majority of the agent system authors is that agents demand productive environments. Furthermore, they require seamless integration into software systems before they will be widely used as a programming paradigm.

Agents can be employed within open hypermedia systems. Furthermore, processes within the Microcosm system show clear signs of agency. They can fulfil a variety of roles serving as information maintainers and monitors, or as aids to the
user's navigation. It is the latter role that is examined in more detail within chapter eight.

It would seem that, after a brief inspection, Microcosm is perfectly designed to allow the inclusion of an agent type process. Such an agent could for example be responsible for arbitrating between the user and the plethora of navigation tools that they need to navigate within a complex open hypermedia network. An agent in this type of role would be similar to the concept outlined in the quote from Oren (Oren, 1987) included within section 6.4.3. Microcosm's modular nature allows the agent to be introduced with the minimum of changes to the remainder of the system. Furthermore, the agent would seem to have full access to all the information it requires via the standard Microcosm communication system.

However, the Segregated Communication Model (SCM), described in chapter 3 incorporated within Microcosm has several major disadvantages which will be highlighted should such an agent process be implemented. The following chapter discusses these faults and introduces two new communication models that will allow the above style of agent process to be implemented.
7. A direct communication model

This chapter opens with a description of a direct communication model (DCM) for the Microcosm hypermedia system. This is presented as an alternative to the segregated communication model (SCM) described earlier in chapter three. The differences between the two models, together with their effect on the operation and structure of the Microcosm system are described with respect to the implementation of the navigation tools detailed in chapters four and five, and also with respect to the implementation of an agent as discussed in the previous chapter.

The chapter then continues to describe a prototype hypermedia system constructed by changing the communication mechanism used by a subset of the Microcosm processes. Examples are provided of changes in both:

- Areas of user perception of the prototype
- Interactions between its various constituent processes

The third section in the chapter details a hybrid communication model (HCM) constructed by merging the direct and segregated communication models. The chapter then includes a section discussing a variation in the SCM (developed in parallel to the work contained within this thesis) that contains an active filter management system (AFMS) (Hill, 1994). The chapter is completed by a section summarising the major differences between the three major communication models.

7.1. Direct communication

The SCM on which the original Microcosm system was based (described in chapter three) contains the following features:

- Viewer/filter segregation - the processes are divided into two classes. Each class has its own controlling process; the Filter Management System (FMS) for the filters and the Document Control System (DCS) for the viewers.
• No direct communication between different process classes - all communication between processes of different classes must go through both the DCS and the FMS.

• Uni-directional communication chain - the filters communicate with each other via a one way communication chain. This causes additional communication overheads and filter ordering problems (see chapter three).

• An asymmetrical architecture - the filter and viewer parts of the model contain different architectures. The DCS can selectively route messages to any of the viewers, in contrast the FMS must send all messages to every filter in sequence.

• A client/server architecture - the DCS and FMS act as servers for the viewer and filter clients respectively.

• Fixed message routing topology - the topology is dictated by the implementation of the DCS and FMS message routers. Filters and viewers have no control over the routing of their messages.

• Messages are routed independent of their content - all messages between any two processes travel along the same route.

• Messages are passed by co-operative processes - messages passing along the filter chain rely on the filters actively passing on messages that they receive. It is possible to produce a filter that does not do this and thus stop all communication within the system.

The following sections discuss a direct communication model that does not contain any of the above features. The advantages gained by their removal are also discussed.

7.1.1. Equivalence for all

The DCM (Wilkins et al, 1993) makes no distinction between viewers and filters. It treats both classes as a single group of processes. Furthermore, the model contains no dedicated routing processes. Instead, the messages are controlled by a combination of the communication system and the processes that generated them.
The removal of the routers and the class segregation allows any process to communicate directly with any other, and has many side effects. The length of the message pathways between processes is always one, irrespective of the processes involved. In contrast, messages within the SCM travel along routes of different lengths dictated by the type and position of the sending and receiving processes.

In the DCM, if a receiver does not exist then the message will not be sent at all. This contrasts with the SCM where no receiver often results in longer message pathways (see section 3.3.1). Furthermore, in the DCM a process is guaranteed the message it sends will be delivered to a receiving process if one exists. If no such process exists then the sender will be informed of this. The segregated model has no mechanism for determining whether a process exists to receive a particular message or whether the message will be successfully delivered. As a result messages often travel through the entire communication system when they need never have been sent.

7.1.2. Symmetrical peer-to-peer architecture

The SCM acts as a client-server network, with the DCS and the FMS behaving as servers for the viewers and filters respectively. This is shown below in figure 7.1.

![Figure 7.1: Microcosm SCM architecture.](image)

In contrast, the DCM acts as a peer-to-peer network, with every process existing at the same level. This is shown in figure 7.2.
Figures 7.1 and 7.2 show the real communication channels and the topology enforced by any message routers for both models. For example a message travelling from process A (A viewer in the SCM) to B (a filter in the SCM) in the two models would cause the following messages to be generated (see figures 7.3(a) and (b)):

Figure 7.3, above, shows the messages generated for the equivalent path through the two models. The route shown in 7.3(a) is a direct result of the routing topology produced by the DCS and FMS. This route is outside the control of the processes sending and receiving the messages. In contrast, the DCM allows the processes such control, producing significantly reduced message route lengths.

Figure 7.3(a) also highlights the effect of the uni-directional chain. If F1 talks to B the chain will work. But if B attempts to talk to F1 it cannot as the communication is against the direction of the chain. The chain makes the model asymmetrical, as no such feature exists on the viewer half of the model. The DCM provides a perfectly symmetrical peer-to-peer architecture. All processes exist as members of the one class and at the same level.
7.1.3. Different communication mechanisms

The DCM contains the concepts of topics and process names (e.g. Linkbase, Available links). A topic is an abstract label used by the communication module to route messages to groups of processes. For example a linkbase will register 'FOLLOW_LINK' as a topic. Any messages sent about this topic will be automatically routed to the linkbase. Process names and topics serve the same purpose as far as the system is concerned, but they differ at a conceptual level. The process names were introduced to allow the DCM to more accurately match the existing SCM in order to facilitate the porting of its functionality.

In order for a process to register with the communication module it must provide a registration message. This contains its name and a list of topics which can be used by the module to distribute messages during conversations. The process will receive a unique identifier when the module has completed its registration. There is no reason why a process cannot register more than once in a session. Each registration will be given a new identifier. To the rest of the system each registration of the process will appear as a separate entity.

The process name and topics which are contained within the registration message need not be unique as each registration is given a unique identifier. For example when two bitmap document viewers register they both use the same process name and topics, but each will receive a unique identifier. This identifier is then used by the rest of the system to identify the process.

A process can communicate with other processes, or conceivably with itself via four different routing methods, these are:

1. **General broadcast** - one process sends a message to all the other processes.

2. **Name oriented** - one process sends a message to all other processes which are registered under the given process *name*.

3. **Topic oriented** - one process sends a message to all other processes which are registered under the given *topic*.

4. **Process to process** - one process sends a message to another process using one of its unique identifiers.
At the lowest conceptual level the DCM is represented by figure 7.4(a). The radial lines on the diagram represent the actual bi-directional channels between the processes and the communication module. Figures 7.4(b) to (e) show the different types of communication which are supported by the DCM. The diagrams show that the topology of the communication model is controlled by the type of communication executed combined with the processes that are communicating.

Every message has data (which can be a tagged ASCII format for compatibility with the message processing within the SCM), an information message and an identifier for the process that sent it. The information message contains the identifiers and process names of the receiver and sender processes. If the
communication is topic oriented then the information message will also contain the topic name. The data is represented by a pointer to a block of shared memory which can contain any information required by the sending and receiving processes.

7.1.4. Centralised process information

The DCM can be used to maintain the state of a process. A process can un-register a topic while it is performing the required function (e.g. processing a link query). On completion it can re-register interest in the topic, thus allowing any new queries to be processed. For example, in the SCM the DCS has to maintain a table of all the viewers currently open and which of those are unlocked (i.e. no bookmark set by the user). In the DCM the onus is on the viewers themselves to remember their current locked state. This state is available to the rest of the system through the dynamic registration of the 'CAN_REUSE' topic. By using this mechanism to reflect the general state of a process, information about the system as a whole is now available to every process. The DCM provides an API that allows processes to query it about which processes are registered for a particular topic etc.

The processes are able to access information (contained within the DCM) about the rest of the system. This allows them to have complete control over the message routing. This has the advantage that the topology of the system is completely dynamic, changing from message to message. However, it does require the processes to become more aware of the system leading to an increase in their complexity.

7.1.5. Message logging and error handling

In addition to the routing mechanisms described in section 7.1.3 and access to the centralised process information discussed in section 7.1.4, the DCM provides functionality to allow a process to register to receive messages in the following cases:

- Every message sent about a particular topic.
- Every message sent to a particular process name.
- Every message ever sent.
This functionality facilitates monitoring of the system by one or more of the processes. In addition a process can register to receive messages that are sent in the following cases:

- To processes which are not currently running.
- About a topic which does not exist.

This allows the easy inclusion of processes to handle error cases. For example if the text document viewer attempts to retrieve links from a linkbase and none are registered then a third process could start one up.

### 7.1.6. Scalability of the communication model

The majority of the communication overhead in the SCM is due to the messages travelling along the filter chain. This is not due to the generation of a large number of messages, but to the serial nature of the chain mechanism. Filters must pass on all messages to the next filter in the chain.

Consider a multi-tasking environment in which viewer processes can be accurately synchronised. Suppose the user causes five viewers to open at the same instant. Upon initialisation those five viewers may send messages to all linkbase filters present (e.g. requesting the locations of any text buttons). All of those messages must be funnelled through the channel between the DCS and the FMS. They must then be bounced through all the filters to the linkbases themselves. The linkbases process the five messages, producing perhaps hundreds of messages which must be returned to the five viewers. The messages produced in parallel must be serially sent down the filter chain, then between the FMS and the DCS. They are only distributed at the last stage when the DCS sends the messages to each of the viewers. The net effect of this is to slow down the whole system. Such slowing down is a direct result of the number of messages generated within the system. The following formulae shown below (and previously in section 3.3.1.5) can be used to calculate the number of messages generated (in the SCM) for the events outlined above. Formula (7.1a) is the number of messages generated when the \( v \) viewers send off link requests to the linkbases. Formula (7.1b) is the number of messages generated when the linkbases at consecutive positions \( p, p+1, p+2 \) in the filter chain.
each respond with \( k \) links. Formula (7.1c) is the total number of messages generated in the above situation.

\[
\begin{align*}
(a) & \quad v(2f+3) \\
(b) & \quad \sum_{q=0}^{p+1} v_k^{-2(f-(p+q))+3} \\
(c) & \quad \text{total} = (a) + (b)
\end{align*}
\]

\( f = \) number of filters in filter chain
\( v = \) number of viewers
\( v_k = \) number of links returned per linkbase
\( p = \) position of linkbase 1 in filter chain
\( L = \) number of linkbases

---

**Equation 7.1**: Calculating no. messages generated in the SCM.

The following formulae represent the calculations shown above but for the DCM. Formula (7.2a) represents the same calculation as (7.1a), formula (7.2c) represents the same calculation as (7.1c). As the new model contains no chain, hence no ordering, formulae (7.1b) can be represented by the simplified formula (7.2b).

\[
\begin{align*}
(a) & \quad vL \\
(b) & \quad vLk \\
(c) & \quad \text{total} = (a) + (b)
\end{align*}
\]

\( f = \) number of filters in filter chain
\( v = \) number of viewers
\( k = \) number of links returned per linkbase
\( p = \) position of linkbase 1 in filter chain
\( L = \) number of linkbases

---

**Equation 7.2**: Calculating no. messages generated in the DCM.

The above sets of formulae can be applied to equivalent instances of both models. For example suppose we have ten filters (including three linkbases) and five viewers in both systems. For simplification we can assume that each viewer will open the same document and each linkbase will contain the same set of links. The graph below (figure 7.5) shows the number of messages produced against the position of the first linkbase in the chain (in all cases \( k=100 \) and \( L=3 \)).
The above graph shows that the number of messages generated (hence the load on the system) in the SCM is dependent on the number of processes (more accurately the number of filters) between the last linkbase and the end of the filter chain. Furthermore, it shows that the number of messages is also dependent on the positions of the sending and receiving processes. In contrast the number of messages generated in the DCM is constant showing that the number of messages generated is independent of both the number of processes and their positions. A more detailed discussion on the effect of such an additional load on the communication system was given in section 3.3.1.5.

7.2. Direct communication prototype

A DCM Microcosm prototype has been constructed by adapting a subset of the current Microcosm system. The prototype consists of an Available links process, DCS, Linkbase and a Text Viewer process. These are the minimum set of processes needed in order to provide a simple hypertext/hypermedia system within the Microcosm design. The processes provide the following functions in the new model:
• **Available links process** - provides the user with the ability to choose a destination from a set of destinations when following a link within the hypermedia network.

• **DCS** - provides the ability to open up document viewers if needed for direct access to any document which Microcosm is aware of. Unlike the DCS in the SCM the new DCS contains no communication or state management functionality. It exists at the same level as any of the other processes within the system.

• **Linkbase** - provides the link storage and retrieval mechanism for any links stored in the system.

• **Text viewer process** - provides the user with the ability to read text documents and follow links from within them.

Each process registers with the DCM when it starts up. It registers a process name and the topics on which it wishes to receive conversations. For example the text viewer registers with the process name TEXT and the topic 'CAN_REUSE'. Viewers within the Microcosm system can be locked by the user, indicating that they do not want the system to use the viewer to display a new document (they are in effect setting a bookmark at the current document). The text viewer dynamically un-registers the 'CAN_REUSE' topic upon being locked by the user, and re-registers it upon being unlocked. Thus, the DCS is able to locate viewers which are not locked (able to display a new document) as only these will be registered with the 'CAN_REUSE' topic.

7.2.1. Cleaner process interaction

Cleaner process interaction is illustrated by the manner in which the Available links process now works. In the SCM, the Available links process has no knowledge of the number of linkbases currently executing. It has to rely on each linkbase sending control messages along the filter chain. The Available links process maintains a counter which is incremented every time it receives a 'PROCESSING' control message and decremented every time it receives a 'COMPLETED' control message. In addition the DCS itself sends an extra 'COMPLETED' control message. When the counter reaches -1 it knows that all the linkbases have finished producing links.
This is a very complex protocol which requires special case coding within the DCS. In the new model a much simpler protocol was implemented. Every module which wishes to produce links, registers under the same topic (for example, 'LINK_PRODUCER'). The Available links process is able to ask the DCM about the current number of link producing processes within the system. Each link producer can send a single control message to denote when it has finished processing (for example, 'COMPLETED'). The Available links process can count these and thus decide when all of the link producers have finished. This is a much simpler protocol which requires no special case code.

The length of the message pathways are considerably reduced in the DCM. For example, a viewer requests the positions of buttons within a document when it is first displayed. The viewer can send the request to all of the linkbases, which have registered the 'FIND_BUTTONS' topic. Each linkbase in turn, responds directly to the viewer using the process-to-process communication facility. This method is much more direct than the current filter chain topology, as only the originating process is involved in the reception of the message.

7.2.2. New user interaction with the system

There are many different ways of interacting with information within GUIs. The most common metaphor used is that of the desktop. Here, the user is presented with documents which are manipulated by applications. Various methods are provided within the windowing environment to allow the user to do this:

- Executing an application and using commands within it to locate and open a document.
- Executing a command with the document as a parameter.
- Interacting with the file system directly. Users can associate particular files (documents) with certain applications. By activating the file, the application is started and the document loaded.
- Dragging a document from one application and dropping it on another.

The Microcosm SCM only supports the first of these interactions. The user must first start Microcosm and then use it as an interface to the documents. This approach is
well suited for naive users, as they only need to learn about the system itself. It is, however, restrictive in that it does not allow more proficient users to interact with Microcosm in the same manner as other applications. The inclusion of these interaction styles will allow Microcosm to become more of a background process than is currently the case. These styles of interaction do not rely on the user interacting directly with Microcosm itself.

The DCM no longer requires the central control of a single process as message routing happens dynamically. It is therefore possible for the remaining three methods of interaction to be incorporated into the system.

For example, the user can start the text viewer and then select the document to be displayed using the text viewer's menu. Or, associations can be made, so that Microcosm processes can be started by browsing the file system. The act of selecting a file will cause the associated application to start and process it. For example, if the link file is associated with the linkbase process, users can choose to use the links by selecting the relevant file. New links will appear in any of the already open documents.

The new system is more flexible, as it provides different interaction styles catering for different types of users.

7.2.3. Reduced communication times

The DCM allows direct communication between any two processes. This greatly reduces the number of messages generated for any given scenario (see section 7.1.6). Furthermore, it also reduces communication times. However, in a multi-process system, it is conceivable that a process will be unable to receive a message because it is busy. The DCM must be able to accommodate this situation, providing a method for re-transmitting the message. This can be achieved by queuing the messages that are to be sent.

Messages are removed from the head of the queue and resent at set time intervals. If this delivery fails then the message is placed on the tail of the queue. This mechanism ensures that messages are not lost, but it does not ensure that they will be received rapidly.
The above approach does not allow for the synchronisation of processes, or the delivery of messages in a time-critical situation. In both cases, the messages must be delivered as quickly as possible. One solution is to allow the message senders to flag the message as important. In this case the DCM does its utmost to ensure that the message is sent as soon as possible. If delivery is unsuccessful, the message will be placed at the head of the queue instead of the tail as normal, ensuring the message will be given priority. A suitable time-out period will ensure the system does not deadlock.

If this does not provide the required decrease in delivery time, more drastic steps can be taken. These rely on knowledge of the underlying implementation of event processing within the GUI. The dangers with using such an extreme approach are that it would lead to a tighter coupling between the communication module and the processes (possibly leading to degradation in communication performance) and make the system more difficult to port to other GUIs and system architectures.

The following table shows the message pathway lengths in both the SCM and the DCM. The length of message pathways directly relates to the time taken to deliver a message from process A to B (see section 3.3.1.5). The four scenarios discussed in section 3.3.1 are considered (i.e. viewer to filter, filter to viewer, filter to filter and viewer to viewer). Each column shows the best and worse cases for both successful and unsuccessful message delivery. There are three different configurations considered:

1. The SCM with two filters and two viewers

2. A generalisation of the SCM, with n filters and m viewers.

3. The DCM, with any number of viewers and filters.
The results show that in the SCM the lengths of the message pathways are independent of the number of viewers, but in most cases proportional to the number of filters (see formula 7.1). From this we can conclude that the main overhead in communication within the SCM is due to the filter chain. In the DCM, however, all messages are delivered directly to the process, and so all message pathways are the same length. This provides a delivery time that is independent of the number of processes.

If there is no process to receive a message, then nothing is actually sent. In the prototype implementation; the DCM simply returns an error to the sending process. This is radically different to the SCM, where the message pathway is, in most cases, longer. This can be seen by looking at the table 7.1.

### 7.2.4. Problems with the prototype

In the SCM, the DCS and FMS contain code to both control processes and handle communication between them. However, communication within the DCM is now handled by the communication module. The control processes have been removed from the new system, and as a result the user has to explicitly start any processes they require. This approach is more flexible, but requires the user to be more aware of the functionality of the different processes available within the system. If the process control features of the SCM are required, then an additional process which
mimics those features within the DCS and Filter manger is required: a Process Manager.

This Process Manager will have to be started transparently whenever the user starts a Microcosm process. By taking advantage of message trapping built into the communication module, it can detect when messages are sent to non-existent processes and so start them up. For example, the user may wish to compute links without the relevant process running. The Process Manager would detect a message being sent to the 'COMPUTE_LINKS' topic, determine the name of the process, start it, and forward the message. The user would see the compute links process start up and the results of their text query appear.

Unlike the control processes in the SCM the Process Manager would exist at the same level as the other processes within the system. It would use the facilities provided by the communication module to detect when to start up new processes.

In the DCM there is no concept of explicit process ordering. In contrast, the SCM provides ordering as a side effect of the uni-directional chain mechanism. Whether such an inherent ordering is needed is still open to debate. In addition, the centralised user interface (situated in the DCS and FMS) that allows a user to shut down or start-up processes has been removed in the DCM. Once again, it is not immediately obvious whether such an interface is essential to the operation of the system.

It is not possible, within the SCM, to determine whether or not there is a receiving process for a message. The DCM allows a process to register the topics it will listen to (receive), but not ones it will talk about (send). The system can only detect a message being sent to a non-existent process at the time of sending. If processes register the topics that they wish to send then the system can detect the error as soon as they start. The Process Manager can then start the missing process or warn the user at the earliest opportunity.

The use of abstract topic identifiers de-couples the content of the message from the destination and provides a further level of abstraction. For example, all processes that understand the concept of link following register for the topic 'FOLLOW_LINK'. The routing of messages has become dynamic, as the message pathways are computed whenever a message is sent.
Because of the new dynamic topology, new protocols need to be designed to allow process interaction. For example, the protocol used within the SCM when a link is followed has already been described in section 7.2.1, along with a possible new protocol. Due to the dynamic nature of the DCM, other protocols must be designed to provide new services within the system. For example, an abstract channel can be defined through which information about link services can be exchanged. Any process interested in such information need only register for that topic (say 'LINKS'). A linkbase might notify other processes that it is starting or shutting down. A text viewer registered for this topic, is able to monitor this abstract channel and react accordingly. It might remove from the document all buttons from a particular linkbase when that linkbase closes down.

7.3. A hybrid model

The introduction of a radically different communication model to the Microcosm system avoids the current message bottle-necks leading to greater system speed, improved interprocess security and less communication redundancy. In addition, the new model enables a far greater degree of flexibility for user interaction. However, the DCM does set new challenges which must be met.

It also introduces certain problems, for example, the lack of ordering or process management. Therefore a hybrid communication model (HCM) is proposed that will be based heavily on the SCM but will allow any process that wishes to use the DCM to do so. In this way the SCM provides the ordering and the process control (via the FMS and DCS) and flexibility and speed is provided by the DCM. The following diagram (see figure 7.6) illustrates how the two models can exist alongside each other, forming a HCM.
In the HCM the DCS still controls the viewers and the FMS the filters. Normal communication is still routed through the SCM, adhering to the topology enforced by the routers (DCS and FMS). However, any process is now able to register with the DCM allowing it to take advantage of direct communication etc.

For example, filter $B$ can block messages to $A$ as long as they travel through the SCM. But $B$ and $A$ can hold a bi-directional conversation using the DCM. The HCM provides the best features of both the direct and the segregated models.

Processes in the HCM connect to both communication modules. The implementor of the process decides which communication system to use for each type of message sent. For example, the mimic system has been adapted so that it sends all of its private messages directly between the mimic engine and the mimic viewer via the DCM. However, it still sends and receives all other messages (that involve other parts of the Microcosm system) via the SCM.

The HCM exists as an extension to the SCM. Any process that wishes to make use of the facilities provided by the DCM links to the dynamic link library that contains its functionality. The other processes do not need to be altered from their original SCM implementations. Therefore, in order to integrate an agent into the
HCM all that needs to be changed are the processes that will directly communicate with the agent and vice versa.

7.4. The active filter management system

During the period of research that resulted in this thesis a separate research project to investigate the distribution of the Microcosm system took place. This resulted in the development of a variation in the original SCM called the active filter management system (AFMS) (Hill, 1994).

The AFMS takes the original chain of filters from the SCM and divides it into a number of smaller chains. Each action (e.g. FOLLOW.LINK and IS.OPENING) is represented by a unique sub-chain. Filters register which actions they wish to receive and are placed into the appropriate sub-chains. The system has the following advantages:

- Reduced communications overhead - All messages are only sent to the filters that have registered to receive them.
- Reduced complexity in the filter ordering problem - Ordering of filters is only a problem within the sub-chains. The overall ordering problem inherent in the SCM has been eliminated.

However the AFMS still has the following disadvantages:

- Asymmetrical architecture - The viewers can be individually addressed by the DCS but the FMS can still only address the filters through the communication chains.
- Segregated architecture - the AFMS still distinguishes between the viewers and the filters.
- No direct communication - All communication between viewers and filters must still go through the DCS/FMS bottleneck.
- Uni-directional communication chains - Filters communicate with each other via a set of ordered one-way communication chains. Whilst the ordering problem has been reduced by the division of the SCM chain it still exists.
Furthermore, the ordering creates a higher communication overhead than the DCM.

- Client/server architecture - the DCS and the FMS still act as servers for the viewers and filters respectively.

Furthermore, the AFMS does not provide the richness in communication types that are present within the DCM (more specifically the general broadcast and the process to process mechanisms). It does not provide the processes with access to information about the other processes that are currently running. In addition, the facilities for error handling and message logging are also lacking.

The AFMS only changes the filter and FMS side of the SCM system. As a result the problems inherent in the SCM concerning the viewers are still apparent. For example, the DCM allows users to utilise any of the mechanisms provided by the Windows environment for starting viewers (e.g. drag and drop and direct interaction with the file system via the file manager), whereas the AFMS only allows interaction via the DCS.

More fundamentally the AFMS cannot exist as a background process (unlike the DCM). This is due to the need to execute the DCS, FMS and necessary filters before the user can use any of the document viewers. In contrast, users of the DCM would be able to start up their document viewer without needing the rest of the system to be executed. The hypermedia functionality is started seamlessly when they attempt to follow a link etc.

7.5. Summary

This chapter has shown that the faults in the original SCM Microcosm can be removed by introducing a completely new DCM. This new model maintains the functionality provided by the previous model but in addition provides direct communication between any processes together with system based hooks for error detection and event logging.

The DCM provides the system with a peer-to-peer symmetrical architecture. Furthermore, the two features restricting the scalability of the SCM (i.e. the DCS/FMS bottleneck and the uni-directional filter chain) have been removed. All
processes belong to the same class and there is no longer any distinction between viewers and filters.

The DCM provides processes with four mechanisms for transmitting messages to others. Processes now have full control over the destination of the messages they send. In addition, the DCM is able to inform the sending process whether or not their message has been received. The fixed topology of the SCM has been replaced by a dynamic topology that changes with each message sent.

It would be very difficult to implement an agent style process within the SCM (or its AFMS variant) for the following reasons:

- There is no direct communication between any one process and all the other processes in the system.
- The client server nature of the system makes it difficult for the client processes to control their peers.
- No uniform mechanism exists to allow processes to be uniquely identified.

In contrast the DCM encompasses solutions to all of the above SCM deficiencies allowing such an agent to be implemented. However, the DCM does not solve all the problems presented by the SCM. Furthermore, it creates some new problems, for example the lack of process management together with the removal of any process sequencing.

The hybrid model presented at the end of this chapter provides a compromise between the direct and segregated models. It retains the process management and process sequencing from the SCM but adds the benefits of the DCM, allowing processes to choose which communication model they want to use for each message sent. It is this hybrid model that forms the basis of the Advisor agent discussed in the following chapter (see chapter eight).

In chapters four and five the implementation of a collection of seven navigation tools was discussed each acting as separate entities. Chapter six discusses agent functionality as a mechanism for autonomously combining advice from these navigation tools (and others). This advice would become a single navigation strategy for presentation to the user. This chapter has shown that the implementation of such an agent would be impossible within the SCM version of
Microcosm. Furthermore, it has presented two new communication models (DCM and HCM) that would allow the agent style process to be implemented within Microcosm.

The following chapter combines the three separate streams of this thesis (e.g. navigation tools, agents and communication models). It discusses the design and implementation of an agent that uses the facilities provided by the HCM to construct an overall navigation strategy from the disjoint advice extracted from the disparate navigation tools within the Microcosm system.
8. The advisor agent

This chapter begins by briefly describing the structure of the existing navigation tools in the Microcosm system. It then discusses in detail several problems inherent in this structure. Although these problems are described in terms of Microcosm they apply equally to any system presenting navigation advice via a set of disparate tools.

The chapter proposes a solution to these problems. This takes the form of an agent (implemented within the HCM version of Microcosm) dedicated to the task of providing overall navigation advice within an open hypermedia system (in this case Microcosm). This section is then followed by a detailed description of the implementation of such an agent.

The chapter continues with a section discussing the performance and functionality of the implemented agent. Finally, the chapter concludes with a summary of its key points.

8.1. Navigation tools within Microcosm

Microcosm is constructed from a set of processes that communicate via a variety of communication mechanisms (see chapters three and seven). The navigation tools have all (with the exception of parts of the mimic system) been implemented as filters. This implementation has several advantages. For example, the tools can access the messages travelling through the system allowing the monitoring of both the state of the system and the user. Furthermore, they can be designed and implemented in a modular manner.

The modular nature of Microcosm allows users to tailor the system to meet their exact requirements. In addition, they are able to decide which filters to run and when to run them. The filters can be thought of as discrete components with standardised interfaces. This drastically simplifies their design and reduces the complexity of their implementation.
With the introduction of the Direct Communication Model (DCM) any process (more specifically any navigation tool) within Microcosm can talk to any other process. Furthermore, they are all able to directly receive messages sent from the receiving processes, broadcast messages to all other processes and generally exhibit more control over their communication (see section 7.1.3).

8.2. Problems with the existing navigation tool strategy

The modular nature of the Microcosm system (although advantageous to the user) enforces the construction of very modular navigation tools. Such compartmentalised construction can cause users several problems as they make use of these tools. These problems are:

- The modular nature allows users to choose which navigation tools to use, and when to use them. They are able to plug in or unplug the tools at any point. As a direct consequence of this adaptability each tool must exist as a self contained unit (as it cannot guarantee what combination of other tools the user has chosen to use). Thus, each tool must maintain its own private local knowledge concerning the state of the system and/or a model of the current user.

- As the tools are acting as separate autonomous processes there is a high likelihood that they will duplicate certain aspects of each other's knowledge. For example, the history filter (see section 4.1) and the memory filter (see section 5.1) both contain information pertaining to the documents that the user has already seen within the current session.

- Each tool will provide users with a point of view based on its local knowledge. Thus in order for a user to gain an unbiased overview of their navigation situation they must query all of the available navigation tools. Once they have queried all of the tools they must judge what advice is of use to them and what is not. They must then combine this information to form the required overall navigation advice.

- Since the tools all operate on their own local knowledge there is no guarantee that their advice will be consistent with advice from other tools.
Not only must the user select the relevant advice from the plethora provided, but they must also recognise and resolve any contradictions that may be contained in that advice.

- Since tools each maintain their own points of view, scenarios can arise where users are provided with essentially the same advice from different navigation tools. For example, the memory filter, mimic system, local map filter and active path guide are all capable of presenting users with information concerning the document they should see next. Such duplication further complicates the user's analysis of the advice presented to them.

The above problems all stem from the modular nature of the filter architecture. Before we can propose a solution to these problems, we must examine what types of advice each part of the system can provide. The following section includes a strict definition of the term *advice* and presents four categories for its classification.

### 8.3. Advice within the Microcosm system

For the purposes of this chapter the term *advice* is defined as:

**Advice** - a list containing one or more entries. Each entry contains a document identifier\(^{10}\) and a confidence value. Furthermore, the advice should be sorted in decreasing order of confidence value.

With the above definition, an inspection of the Microcosm system yields the following four classes of advice:

**Navigation centred advice** - Advice directly derived from the navigation of the user. This would be obtained from sources such as the history filter, the mimic system or the local map filter.

**Filter centred advice** - Advice not directly relating to the user's navigation, but that is contained within a filter. For example, this advice could be extracted

\(^{10}\)Each document within Microcosm has a unique document identifier. These identifiers are maintained by the Document Management System (DMS).
from filters such as the linkbase (Heath, 1992), the computed linker (a full text retrieval filter) (Li, 1993) or a knowledge based filter (Soper & Pasha, 1994; Soper & Pasha, 1994b).

**Viewer centred advice** - Advice similar to the filter centred advice but extracted from the viewers, such as advice concerning the content of their current documents and/or the links from those documents.

**System centred advice** - Advice generated by querying services provided by the central parts of Microcosm. For example, advice could be extracted from services such as the document management system (a database containing information about the members of the current document set), or the registry (a hierarchical storage facility provided by Microcosm, replacing the layered ini. files discussed in section 3.3.5).

With the above categories in mind the next section describes a mechanism in the form of an agent which is proposed as a solution to the problems in the previous sections. This agent would act on behalf of the user, aiding their navigation through the current document set.

### 8.4. The advisor agent: a proposed solution

After examination of the five problems discussed earlier in section 8.2, what is needed is a process that arbitrates between the user and the advice sources provided by Microcosm. To be more exact, an agent that is able to interact with these advice sources, presenting its information to the user, would provide potential solutions to all of the user centred problems.

This agent would query the advice sources, collect the replies from each one, combine them and present the user with an overall view of the advice. The user would delegate the responsibilities listed below to the agent:

- Determining which advice sources are currently available
- Querying all the available advice sources.
- Combining the replies from these advice sources.
• Resolving contradictions in the advice.
• Eliminating any advice duplication.

The agent would be implemented as a filter, allowing it to exist as a peer within the filter system (not as a server). Such an implementation would allow it to monitor the existing messages within Microcosm without needing to change any of the other processes. Furthermore, users would be able to unplug, or plug in, the agent without affecting the functionality of the remaining system components. In addition, they would still be able to query the advice sources directly (e.g. the history filter). The agent would exist purely as an extra advice source. This design is similar to the implementation of the Apple guides experiments (see section 6.3.1).

The agent would need to be implemented within the Hybrid Communication Model (HCM). It would use the Segregated Communication Model (SCM) to monitor the rest of Microcosm and the user's effect on it. However, it would use the Direct Communication Model (DCM) to converse with the advice sources. This would allow it to query advice sources and receive their replies directly. The advice sources can be in any process on the user's machine, they need not be either a filter or a viewer, or even a part of the Microcosm system (see chapter seven).

Implementing such an agent within the SCM would be impossible as it could not be positioned within the filter chain so that it could communicate with all of the remaining system processes. Furthermore, the SCM does not provide facilities for determining what processes are currently operating. This information will be required by the agent in order for it to decide which advice sources are available for querying.

Lastly, the agent must be able to maintain its integrity if unplugged from the system. As it is implemented as a filter, users are at liberty to unplug it, or plug it in, part of the way through a session. Thus, the advice querying mechanism must be capable of conversing with the sources on the fly. In addition, once users have initialised the agent it should be capable of being suspended and reactivated. Furthermore, it must be robust to changes in the system, for example different sets of filters being executed. Finally, it should be implemented in such a manner as to allow it to cope with the addition of new advice sources.
8.5. Implementation of the advisor agent

This section discusses the implementation of the Advisor agent with regard to the aims and restrictions outlined previously. Furthermore, it describes features that are not immediately apparent given the issues raised in the above section. The implementation of the agent can be subdivided into four sections:

**Advice generation** - This discusses exactly what advice sources exist within Microcosm, what advice they offer and how they can be adapted to present this advice to the agent.

**Advice collection** - This covers the implementation of a generic protocol between the advice sources and the agent, the constraints introduced by Microcosm's dynamic nature and the reliability of advice sources.

**Advice amalgamation** - This describes the control interface (presented to the user) for controlling the collection of advice. In addition, it outlines the actual mechanism used to amalgamate the advice collected from the different advice sources.

**Advice presentation** - This considers the modes of interaction that the agent offers to users. It also describes a system that allows the advice to be presented by a variety of different advice display mechanisms.

The following sections discuss each of these four areas, in greater detail.

8.5.1. Advice generation

This section describes which parts of Microcosm can be used as advice sources. In addition, it details what advice is provided by each source and what changes are needed to each source to allow it to converse with the advisor agent.

8.5.1.1. What advice sources are available to the agent?

The Advisor agent has been implemented in the hybrid communication model, allowing it to query any process within the system that registers to accept such
queries. The advice provided by these processes can be sub-divided into four broad categories (see section 8.3).

The differences in advice origin can be hidden from the agent by implementing a generic communication protocol. Such a mechanism would allow the same piece of code (within the agent) to query all sources of advice, irrespective of their actual location within the system. In addition, if the replies also adhere to a generic protocol then the same piece of code (within the agent) could be used to receive the advice. The use of generic protocols would also make the system simple to extend, thus allowing the easy inclusion of additional advice sources.

Figure 8.1 illustrates the locations of the different advice sources. It should be noted that no distinction has been made between the clients (the filters and the viewers) and the servers (the DCS and the FMS). This is due to the ability of the direct communication model to allow them all to communicate on an equal status (i.e. peer-to-peer). The system centred advice is shown in the centre of the diagram, as all processes (irrespective of their class) can access it. This access is via a system of dynamically linked libraries that any process is able to link to, thus giving a process access to the advice contained within the libraries (or maintained by the libraries).
8.5.1.2. What types of advice are offered by each source?

Tables 8.1, 8.2 and 8.3 detail the advice provided by various processes within three of the four categories of advice source listed in section 8.3. The viewer centred advice has been omitted as a table because each viewer can only provide advice based on its own current document. Thus the following three advice source categories have been concentrated on:

<table>
<thead>
<tr>
<th>Advice source</th>
<th>Advice offered by source</th>
</tr>
</thead>
<tbody>
<tr>
<td>History filter</td>
<td>Which document not to see next, as they have already been seen. As documents descend the history list the strength of advice will decrease.</td>
</tr>
<tr>
<td>Local map filter</td>
<td>Which document to see next, based on the number of links to other documents. Thus the most connected document will be recommended allowing users to move around more easily.</td>
</tr>
<tr>
<td>Mimic system</td>
<td>Which document to see next based on which mimic is currently being viewed. If no mimic is being viewed then the mimic system will not provide any advice.</td>
</tr>
<tr>
<td>Memory filter</td>
<td>Which document to see next, based on its current set of links, the documents seen most recently (i.e. high in its memory) will be placed lowest in the advice.</td>
</tr>
<tr>
<td>Active path guide</td>
<td>Which documents to see based on the past routes of other users. The advice will be ranked by the popularity of the route from the current document to the next document.</td>
</tr>
</tbody>
</table>

Table 8.1: Types of advice generated from navigation centred sources
<table>
<thead>
<tr>
<th>Advice source</th>
<th>Advice offered by source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computed linker</td>
<td>Which document to see next (text only) based on the similarity between the current document and all other text documents in the document set. The most similar document will be placed highest in the advice.</td>
</tr>
<tr>
<td>Linkbase</td>
<td>Which document to see next, based on the links between documents. Links of different scope can be scored accordingly when producing the advice.</td>
</tr>
<tr>
<td>Knowledge base</td>
<td>Which document to see next based on the rule base contained within the filter. The resolution of the rules could of course be controlled by actions external to the filter.</td>
</tr>
</tbody>
</table>

Table 8.2: Types of advice generated from filter centred sources

<table>
<thead>
<tr>
<th>Advice source</th>
<th>Advice offered by source</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMS</td>
<td>Which documents to see next based on the information contained in the DMS. For example, it could advise the user to see documents by the same author.</td>
</tr>
<tr>
<td>Registry</td>
<td>Which documents to see next based on the information contained within the registry.</td>
</tr>
</tbody>
</table>

Table 8.3: Types of advice generated from system centred sources

8.5.1.3. Transforming Microcosm processes into advice sources

As the above tables clearly show, there is an abundance of different advice sources within the Microcosm system. However, before the advisor agent can take advantage of these sources they must all fulfil the following requirements.

Firstly, they must all be able to make use of the DCM. This requires that each advice source be represented by a window object. The window object will provide a message queue which is used by the DCM to act as the message receiver. All the filters and viewers already have associated window objects, but the system centred sources are contained within libraries and do not have directly related window objects.

One solution would be to allow the advisor agent to link directly to these system centred advice libraries. This would indeed allow it to access the advice they contain, but would mean that the agent needs to understand two acquisition
mechanisms (i.e. messages via the DCM, and function calls within the system centred advice libraries). The function call mechanism is not easy to extend dynamically and the duplication of acquisition mechanisms is unsatisfactory.

A more elegant solution is to introduce a window object into each of the system centred advice source libraries. This object is created when the libraries are initialised, allowing the library to communicate with the other processes in the system via the DCM. Thus, all advice sources would communicate with the agent using a common mechanism.

Even with the communication mechanism in place, the advice sources need to be changed before they can process advice requests from the agent. This requires them to be able to receive the request, construct advice (derived from their local knowledge) and return it to the agent. In addition, they need to be adapted to send and receive the house keeping protocol messages used by the agent. For example, when an advice source is initialised it must register with the DCM and then broadcast a message informing other processes of its presence.

Finally, the advice sources must broadcast a message when they terminate. In certain scenarios an advice source may be terminated without broadcasting such a message. This situation is covered by the protocol used by the advisor agent and is discussed in greater detail in the following section.

8.5.2. Advice collection

This section discusses the advice collection mechanism within the advisor agent. It outlines the generic protocols and APIs that exist between the agent and the advice sources. In addition, it discusses the problems caused by the dynamic nature of Microcosm.

8.5.2.1. Generic protocols for advice sources

Generic protocols are used to exchange information between the advisor agent and the advice sources. This means that as long as an advice source adheres to the protocol it can be introduced into the system without changing the agent, and vice versa. It also means that advice sources can be constructed from code templates where the protocol handling code will be within the template. The author then needs to fill in spaces within this generic code with more specific code to allow the
advice source to produce advice. Furthermore, the agent code can be constructed from a second template, allowing a new agent to be written without the author having to start from scratch.

The protocol has two aspects (or directions). The first handles information travelling from the agent to the advice sources and the second concerns information travelling in the opposite direction. Both are discussed below.

Agent to advice source - This information takes the form of an advice request, broadcast via the DCM to every process registered to receive it. The request contains the query document (i.e. the document about which advice is sought), the number of entries required in the reply and a request identifier. In addition to this explicit information, the request also includes the return address of the advisor agent (for a direct reply from the advice source) and the topic advice request.

Advice source to agent - This information takes the form of an advice reply, sent directly from the advice source to the agent (using the address extracted from the advice request). The reply contains the number of entries in the advice, the advice itself (each entry containing a document identifier and a confidence value) and a copy of the request identifier. In addition to this explicit information, the reply also includes the address of the advice source and the topic advice reply.

The use of the request identifier, together with a description of the confidence values, can be seen in the following sections.

8.5.2.2. Dynamic nature of system

The dynamic nature of Microcosm affects the advisor agent in two ways. Firstly, users can unplug, or plug in, advice sources dynamically. Secondly, they can unplug, or plug in, the advisor agent itself. Both problems can be overcome by the introduction of a second protocol. This deals with the dynamic nature of the system in the following ways:

On the fly advice source location - Before the agent issues an advice request it queries the DCM to determine exactly what advice sources are operational at that time. This is achieved by querying the system state information held within the DCM (see section 7.1.4). The agent then broadcasts (via the DCM)
an advice request to all active advice sources, with the DCM taking care of
the message routing. The agent inserts a request identifier into the request.
These are generated from a counter which starts at zero and increments with
each request generated. When an advice source receives a request it
responds immediately, telling the agent that it is processing the request. The
agent then waits for a set time period and if all the advice sources reply it
combines their advice and informs the user. If not, then it will combine the
advice received so far and inform the user of its possible incompleteness. If
the agent receives a reply with the wrong request identifier (for example a
late reply) it is ignored.

**Advice source house-keeping** - Every advice source broadcasts a house-keeping
message to the advisor as it is executed. This allows the agent to know
that a new advice source has become available. In addition, the advice
sources broadcast a second message when they are closed down. Thus, the
agent is informed of changes in the system (concerning advice sources) by
the advice sources themselves. When the agent is started it broadcasts a
request for the advice sources to identify themselves. This allows it to be
plugged in part of the way through a session, since as soon as it is initialised
it locates the current advice sources.

These two mechanisms, form a double edged protection against possible
contradictions between the agent's list of advice sources and the actual advice
sources within the system. For example, if an advice source is terminated without
sending the house keeping message then the agent will be protected by the former
mechanism. Furthermore, the agent's responsibility for constantly monitoring the
advice sources is removed by the latter mechanism.

8.5.2.3. Maintaining advice source information

The agent needs to store information specific to each advice source that it uses. For
example, it stores the names of the advice sources, whether they are being used or
not, and the reliability values associated with these advice sources.

The information is stored within the layered ini. files discussed in section 3.3.5.
This allows the user to start the agent whenever they wish. When started, it can
simply recreate the state of its local knowledge from the previous session. The user
only has to set up the information once and from then on it becomes the responsibility of the agent to maintain it.

8.5.3. Advice amalgamation

This section describes the interface that the agent provides for users to tailor the advice source combination and query mechanisms. Furthermore, it discusses the advice source combination mechanism in more detail.

8.5.3.1. The agent control interface

This interface allows users to control certain facets of the advice collection and combination mechanisms. Users can tell the agent to ignore advice from specific advice sources by turning the advice source off. In reality the advice source is not turned off but the advisor agent will not send advice requests to it.

Each advice source has an associated reliability value (or weight). These control how advice from each of the sources is combined. The interface allows users to change the reliability value for each of the currently active advice sources. Figure 8.2 illustrates the agent control interface.

![Agent control interface diagram](image-url)

Figure 8.2 : The agent control interface.
In addition to the functionality discussed above, the agent control interface also allows users to save the current state of the advice sources (e.g. whether they are on or off, together with their reliability values). Furthermore, the interface provides users with the ability to make the agent issue an advice location message. This has the effect of removing the current list of advice sources, broadcasting a location message and recreating the list using the replies from the advice sources. In addition, the list will also update dynamically as advice sources are started or shut down.

The interface allows users to set the type of interaction that the agent will use when communicating its advice to them. There are two styles of interaction, autonomous or semi-autonomous. They correspond to the radio buttons indicated in figure 8.2, and are discussed in more detail in section 8.5.4.1.

Finally, the interface has been designed to allow its seamless extension, enabling it to handle the introduction of new advice sources. The advice sources are listed within a list window, thus allowing new advice sources to be added without the need for the agent interface to be modified.

8.5.3.2. Advice combination mechanism

Each advice source provides advice based on its own local knowledge, as discussed earlier. When viewed globally, the advice is likely to contain both contradictions and inconsistencies. The advisor agent uses the advice combination mechanism to combine advice from multiple advice sources. Furthermore, it allows the agent to remove any contradictions and resolve any inconsistencies contained within the advice.

The mechanism includes two layers of weighting factors allowing advice from a single advice source to be ranked (locally) and the advice from any one advice source to be changed relative to all the other advice sources (globally). These layers are discussed below:

Confidence values (local) - These reflect the advice source's confidence in the associated information. For example, the documents at the top of its advice will have the highest confidence values, indicating that the advice source is confident that these documents are the most relevant. The confidence value can range from one hundred (most confident) to zero (no confidence at all).
Documents that have zero confidence are removed from the advice list. If such pruning was not carried out then the advice list would contain a large volume of useless information. The confidence values are generated by code that is specific to each advice source (a brief description of the basis for the production of these values for each advice source can be seen in tables 8.1, 8.2 and 8.3).

**Reliability values (global)** - These reflect the agent's view of the reliability of the associated advice source. For example, if the reliability value for a particular advice source is high then the agent will treat the advice from that advice source as being very reliable (causing it to score highly relative to the other advice sources). In addition, the reliability value is used to denote whether the advice source is providing positive or negative advice. An example of an advice source that produces negative advice would be the history filter, as it advises which documents not to see. The values are actually set by the user (via the agent control interface), therefore they reflect the user's view of the reliability of the various advice sources. The values are represented by floating point numbers allowing the user finer control over the combination of the advice from various advice sources.

The section of pseudo code in figure 8.3 illustrates the actual advice combination mechanism, the two layers of weights can be clearly seen.

```plaintext
Get list of currently active advice sources
Send off an advice query to each advice source
For each advice reply
   Retrieve global reliability value for the advice source
   For each entry in the reply
      Local confidence value * global reliability value
   Construct master list with advice from every advice source
If (document X in >1 advice reply) then
   Sum the values for document X
If (document X scores <= 0) then
   Remove from global list
Sort global advice list
```

**Figure 8.3**: The advice combination algorithm.

The following figure (see figure 8.4) illustrates a worked example of advice combination given the algorithm outlined above. The letters A to F represent documents that are known to the advice sources, as such they are all possible valid destinations.
The above example illustrates the advice combination mechanism. The advice sources are all navigation filters, providing advice with different reliability values. For example, Source 1 could be the active path guide and Source 3 could be the mimic system. Source 2, however, could be a history filter providing advice that is treated negatively (as shown by the negative reliability value).

Source 1 is recommending documents (A, B, C, D), these are the most popular routes taken by the previous users. Source 2 is recommending the user not to see documents (A, E, F), as the user has seen these documents before. Source 3 is recommending documents (B, D, C), as these are the documents which come next in the current mimic. When the agent receives the advice from each source it executes the advice combination algorithm shown earlier in figure 8.3.

The user has elected to make the mimic system the dominant source (i.e. the highest advice reliability). As a result the final advice is to follow the mimic, seeing document B next. If the user had not already seen document A then the advice would be to see documents (B, D, A, C). The dominant advice would still be to follow the mimic, but in addition the user would be shown that document A is also of interest.
8.5.4. Advice presentation

This section discusses the various modes of interaction, offered to the user, by the agent. The abstraction of the presentation mechanism from the agent is then described. This includes sections discussing an API that allows the two to communicate, and discusses the dynamic nature of the advice presenter/agent relationship. The section is concluded by a description of three different advice presenters.

8.5.4.1. Modes of interaction

The advisor agent presents users with two modes of interaction. This provides a greater degree of versatility, allowing users to choose the interaction mode that suits them best. Furthermore, they are able to change the mode of interaction at any time. The two modes are:

**Semi-autonomous** - The user explicitly queries the agent for advice when they require it. It is autonomous because once queried the agent operates without user interaction, collecting and combining the advice from the available advice sources. However, it is only semi-autonomous as the user still has to ask the agent for advice.

**Autonomous** - The agent provides advice to users without any direct user interaction. It uses a trigger from within the system to initiate the advice collection/combination process. For example, every document that is opened within the system causes an 'IS. OPENING' (see section 4.1) message to be sent through the communication system. If the agent uses this message as a trigger, then it will provide advice every time the user opens a document.

The mode of interaction is separated from the method by which the advice is actually presented to the user. The following sections discuss the issue of advice presentation in more detail.

8.5.4.2. Abstraction of advice presentation

Abstracting the advice presentation mechanism from the agent allows different advice presenters to be offered to users (all driven by the agent). Furthermore, the flexibility provided by such abstraction follows the philosophy of adaptability.
adopted by Microcosm itself. Each user is able to choose which presenter they wish to use and when they wish to use it.

A communication protocol has to be introduced between the agent and the advice presenter before the advice presenter can be abstracted from the agent. Each advice presenter will adhere to this protocol, allowing different presenters to interface to the same piece of code within the agent. In addition, once such a protocol has been instigated new presenters can be designed and implemented without the need to change the agent itself.

8.5.4.3. Dynamic nature of advice presenter and agent relationship

Each advice presenter is implemented as a dynamically loadable library, allowing users to change the advice presenter at any time (including whilst the agent is operating). Each library has the same standard interface, allowing it to interact with the agent without the agent having to be changed for each presenter.

When users wish to change the advice presenter, the agent unloads the library containing the current presenter and loads the library containing the new one. The user then continues, using the new advice presenter instead of the old one. The following section discusses several such advice presenters in more detail.

8.5.4.4. Presentation of the combined advice to the user

The dynamic nature of the advice presenter and agent relationship allows users to choose which advice presenter they want to use and when they wish to use it. They are able to unplug, or plug in, the advice presenter of their choice. The following three advice presenters are provided as examples of the possible range of such devices.

8.5.4.4.1. Memory filter

The memory filter advice presenter (MFAP) uses the standard memory filter (see section 5.1) as a display mechanism for advice generated by the advisor agent. The advice is presented through the same process as a list of end anchors emanating from a link's start anchor, or a list of statistically similar documents generated from the full text retrieval system (Li, 1993). The MFAP library takes advice from the agent (in one form) and translates it to a form understood by the memory filter. It then sends this advice back into the SCM.
This advice presenter has two advantages. Firstly it re-uses an existing part of the Microcosm system and secondly it provides users with the advice in a familiar environment. The advice appears within the memory filter in the same form as a list of links produced from a linkbase. The pieces of advice are sorted by their strength, with the entries at the top of the list being the strongest.

Figure 8.5 illustrates two routes that advice, from the advice presenter to the memory filter, can take. Route (a) takes the advice directly from the advice presenter back into the segregated communication model. Route (b) takes the advice directly to the memory filter, missing out the SCM entirely.

Route (a) requires that the advice presenter registers with the filter management system as a filter. Once registered it will receive all messages travelling through the SCM. Furthermore, it will be able to send messages to other filters within the system (e.g. the memory filter). This method is inefficient as it requires the advice presenter to process a large volume of irrelevant material. In addition, the use of the SCM means that the advice sent by the presenter may never reach the memory filter (see section 5.1).

Route (b) requires that both the advice presenter and the memory filter register with the direct communication model. The disadvantage of this route is that the memory filter needs to be adapted to allow it to accept advice through the DCM. However, the advantage is that advice sent from the presenter is guaranteed to arrive at the memory filter. Furthermore, using the DCM instead of the SCM reduces the overall number of messages travelling through the communication system (see section 7.1.6).
8.5.4.4.2. Bar graph list advice presenter

The MFAP allows the agent to present its advice through an existing part of the Microcosm system. This has the disadvantage that the advice presenter has no control over the way in which the advice is presented. All it can do is translate the advice provided by the agent into a form understood by the memory filter, and send that advice to it. The MFAP has no chance to embellish the advice in order to enhance its information content, since it is not specifically designed for advice presentation.

The second advice presenter actually presents the advice to the user itself. The advice is presented to the user in the form of a list (similar to that used in the memory filter). The advice in the memory filter used the ordering of the list to denote the strength of the advice so that the advice at the head of the list was the strongest.

With the MFAP users have no concept of the relative strengths of individual pieces of advice compared with others. For example, advice entries with strengths (100, 90, 80) would appear the same as advice entries with strengths (100, 10, 5). This is clearly undesirable, as the user will perceive the advice as being the same. In reality only the advice at the head of the second example is useful, the tail of the list is too weak in comparison.

Figure 8.6 illustrates the bar graph list advice presenter (BGLAP). It is basically the same as the MFAP, but has relative strength bars added to each piece of advice in the list. The advice entries in the example have the following strengths (100, 80, 5). The bars indicate the strength of a piece of advice relative to the top piece of advice. The top piece of advice will always be one hundred percent (as it is shown relative to itself). In this example the second and third pieces of advice will be eighty percent and five percent of the top piece of advice.
Figure 8.6: The bar graph list advice presenter

Users can easily see that the first two pieces of advice are valid, and that the third piece of advice is not as confident as the other two. The bar graphs provide the advice presenter with a mechanism that allows it to supply users with additional information.

8.5.4.3. Circular plot advice presenter

The circular plot advice presenter (CPAP) provides users with the same functionality as the BGLAP (i.e. it shows the strength of one piece of advice relative to others). However, it uses a spatial method to present this information (instead of a graphical mechanism). The following diagram (see figure 8.7) illustrates the CPAP.
Each point on the interface represents a piece of advice. The concentric circles allow users to see pieces of advice with equal strengths (each will be on the same circle). The nearer a point is to the origin of the circles, the stronger it is. For example, the points on the inner circle represent the strongest advice and the points on the outer circle the weakest advice.

The currently selected advice is shown by changing the colour of its representative point and by displaying its title in the area towards the top of the interface. Users can follow advice by selecting it, then pressing the view button towards the bottom of the interface. This will cause the document represented by the selected piece of advice to be opened by Microcosm. If the agent is operating in autonomous mode then the opening of the document will cause the advice presented to be updated.

The three advice presenters (illustrated above) are just three from a wide range of possible devices that could be implemented. They display a range in advice presentation from a bar graph method to the spatial method employed in the circular plot diagram. Furthermore, they show that existing parts of Microcosm can be used as vehicles for advice presentation. In addition they show that the advice presentation mechanism can be abstracted from the advice generation and the advice content. Further research would be needed to ascertain which advice presenter is preferred by users in which conditions.
8.6. An overview of the relationships between the advisor agent and Microcosm

This section discusses the relationships that exist between the Advisor agent and the rest of the Microcosm system. Firstly, the agent is implemented as a Microcosm filter, allowing it access to the messages that flow through the SCM. This is the only dedicated connection that the agent has with Microcosm.

All other connections are via the DCM. As stated previously the DCM is capable of hiding whether a process is part of Microcosm or not. Therefore, the agent cannot be sure that any advice source that it communicates with is really a part of Microcosm. Figure 8.8 illustrates the relationships between the agent and the other parts of the advice system (most of which are parts of Microcosm). The diagram is sub-divided into four areas. These show the distinction between front-end/back-end and Microcosm/Advisor agent functionality.

![Figure 8.8: Relationships between the Advisor agent and Microcosm.](image)

Figure 8.8 clearly shows two types of advice source, the first having a user interface (front-end) and the second only being directly accessible via the agent/advice source protocol (API 2).

The connection between the agent and the advice presenter is via the DCM and is not really a part of the Microcosm communication system. The connection is
dedicated to carrying advice to the presenter and no other part of the system is aware of its existence.

The only direct connection that the advisor agent has with the user is via the agent control interface. However, as the agent is implemented as a filter, users can use the filter management system as an indirect interface, allowing them to control whether the agent is operating or not.

The front-end of an advice source is connected to the back-end by the use of an advice source specific API. This is shown in the figure 8.8 as API 1, each advice source will have its own implementation of this API and no other process will have access to it.

8.7. Case studies of the advisor agent

The following sections contain two case studies of the advisor agent in use. Each case study contains two scenarios. The first scenario highlights a problem caused by using a specific set of advice sources. The second scenario illustrates how advice from other parts of the system can be used to remove the problem.

The following notation will be used within the two case studies to show the types of documents involved at each stage:

- (A) - Animation document
- (I) - Image document
- (M) - Mimic document
- (T) - Text document
- (V) - Video document

The advice lists included in the following case studies have been abbreviated. All documents that had a confidence value below a set truncation value have been removed. This does not change the advice but simply makes the case studies more compact.
8.7.1. Case study one

This case study involves a user attempting to discover more information concerning a particular topic within the cell biology document set. Both scenarios start with the user at the 'Introduction to cell biology' (T) document. This document acts as an overview to the remainder of the document set. The truncation value for this case study is forty, therefore every piece of advice that scores less than this value has been removed from the advice lists.

8.7.1.1. Scenario A

The advisor agent is using the active path guide (APG) as the only advice source. The APG is using local knowledge constructed from actual log files recorded during a set of experiments involving the use of the Cell Biology document set by undergraduates as part of a 1st year course on Cell Motility within the Biology department at the University of Southampton (Hall et al, 1989; Hutchings, 1993). The scenario describes advice about movements within a real hypermedia network derived from the analysis of data gathered from real users of the system.

(1) The user reads the 'Introduction to cell biology' (T) and the advisor agent is then queried for advice. The response of the agent is to produce the following list of advice:

100 Tissue cell movement (T)
87  Microfilaments (T) {Document viewed next}

(2) The user selects the second document as it contains information about the current topic. The agent is queried and responds with the following list of advice:

99  Actomyosin (V) {Document viewed next}
74  Actomyosin (T)
41  Microtubules (T)

(3) The user selects the first document as it is the most strongly advised. The agent is queried, it responds with the following list of advice:

99  Actomyosin (T) {Document viewed next}

(4) The next highest scoring document had a score of nine, thus the APG is showing that there is a strong connection between the two Actomyosin documents.
The user selects the document, the agent is queried and responds with the following advice:

99 Actin/Myosin motor (A) {Document viewed next}
47 Tubulin assembly (T)
44 Actomyosin in non-muscle cells (T)

(5) The user selects the first document as it contains more information relating to the previous two documents. The APG has now shown that the three Actomyosin documents contain connected information. The agent is queried and responds with the following advice:

99 Actomyosin (T) {Document viewed next}
81 Tissue cell movement (T)
81 Muscle contraction (V)

This is the end of this scenario. If the user chooses to see the first document and queries the agent he/she would be advised to see the document recommended at stage four. The user would be caught in an advice loop since the APG takes no account of the documents that the user has already seen.

8.7.1.2. Scenario B

This scenario is essentially the same as the previous one, but the advisor is now using the history filter as an additional advice source. The user is advised to follow the same path, as in the previous scenario, until they get to stage (5). The APG is still using its local knowledge created from analysing real data from real users of the system. This time the history filter will have an influence on the advice provided by the APG.

(1) In the previous scenario, the agent got caught in an advice loop (its only advice source was the APG). This time it provides the following advice:

81 Tissue cell movement (T) {Document viewed next}
81 Muscle contraction (V)

(2) It can be seen that the 'Actomyosin' (T) document has been removed from the head of the advice list. The history filter has provided advice telling the agent to remove the document from the advice as the user has already seen it. The user chooses the first document, queries the agent and receives the following advice:

99 Actin binding proteins (T) {Document viewed next}
88 Ruffling (V)
55 Tubulin assembly (T)
(3) The user selects the first document and continues on their journey.

The use of the APG as the only advice source led to the user being caught in an advice loop. The agent advised the user to see document B from document A and vice versa. The advice provided by the APG takes no account of which documents the user has already seen.

The introduction of the history filter as a second advice source allowed the user to move further through the document set. The history filter will provide the agent with negative advice about the documents that the user has seen already. By combining the advice provided by both advice sources the agent is able to suggest documents to see without suggesting documents that the user has already seen.

8.7.2. Case study two

This case study involves a user following a mimic about Microfilaments, within the cell biology document set. Both scenarios start with the user at the 'A walk through on Microfilaments' (M). The truncation value for this case study is fifty, so every piece of advice that scores less than this value has been removed from the advice lists.

8.7.2.1. Scenario A

The advisor is using the history filter and the APG (with the same real local knowledge as for case study one). The advisor is not, however, using advice from the mimic system. The user's task is to simply follow the mimic (indicated previously) and see how the advice from the advisor differs from the mimic route.

(1) The user opens the mimic document and queries the advisor agent. The agent responds with the following advice:

100 Introduction to cell biology (T) (Document viewed next)
50 Microfilaments (T)

(2) It is interesting to note that the above advice is to see the first then second document in the mimic (even without the mimic advice source). The user moves to the first document in the mimic, queries the agent and receives the following list of advice:

100 Tissue cell movement (T)
(3) The user views the second document in the list as this is the next document on the mimic. The agent is queried and responds with the following advice:

- 99  Actomyosin (V)
- 74  Actomyosin (T)

(4) It seems that the agent is not advising the user to see the next document in the mimic. However, the next document ('Intermediate filaments' (T)) does appear in the advice list, but only scores twenty-four. This value is not high enough to allow it to appear in the above list. The user views the next mimic document 'Intermediate filaments' (T), queries the agent and receives the following advice:

- 100 Microtubules (T)  (Document viewed next)
- 50  Actomyosin (T)

(5) The user views the first document in the list as this is the next document in the mimic. The agent is queried and responds with the following advice:

- 99  Tubulin assembly (T)
- 83  Arrangement of protofilaments (I)

(6) This time the agent has not included the next document in the mimic in the advice. The user still follows the mimic to see 'Assembly of F - actin' (T), queries the agent and receives the following advice:

- 100 Tubulin assembly (T)

The user has now seen the entire mimic and the scenario is finished. The advisor agent used the APG and the history filter to provide advice to the user. Even though the mimic system was not used as an advice source, by the agent, the documents within the mimic still appeared within the advice lists. This is due to the APG recording previous users following the mimic.

8.7.2.2. Scenario B

This scenario is almost identical to the previous one, except this time the advisor agent is using the mimic system as an advice source (in addition to the APG and the history filter). Once again the user's task is to follow the mimic and take note of the advice offered by the agent.

(1) The agent offers the same advice as the previous scenario until we get to stage (2). The user is looking at 'Introduction to cell biology' (T), which is the first
document in the mimic. The agent is queried and provides the following set of advice:

187  Microfilaments (T) {Document viewed next}
100  Tissue cell movement (T)
 87  Microtubules (T)
 75  Intermediate filaments (T)

(2) The advice now has the next mimic document at the top, followed by a document that is not part of the mimic and then two more documents from the mimic. The user selects the first document, queries the agent and receives the following advice:

123  Intermediate filaments (T) {Document viewed next}
107  Microtubules (T)
 99  Actomyosin (V)
 74  Actomyosin (T)

(3) The user chooses the first document as this is the next one on the mimic. The agent is queried and responds with the following advice:

200  Microtubules (T) {Document viewed next}
 50  Actomyosin (T)
 50  Assembly of F - actin (T)

(4) The user chooses the first document as this is the next document within the mimic. The agent is queried and responds with the following advice:

100  Assembly of F - actin (T) {Document viewed next}
 99  Tubulin assembly (T)
 83  Arrangement of protofilaments (I)

(5) The agent has now included the 'Assembly of F - actin' (T) in its advice. This is due to the advice contribution from the mimic system. The user views the first document and the scenario is complete (the mimic is over).

Without the contribution of the mimic system the advisor agent often advised the user to see documents that were not the next documents in the mimic. This is because the agent had no idea that the user was in fact following a mimic. As far as it was concerned the user was simply opening documents that had no authored inter-connections.

When the mimic system was included as an advice source it could be clearly seen that the documents within the mimic were promoted within the advice. In one instance a document that had not previously been in the advice list was now included (stage (6) scenario A vs. stage (5) scenario B). Furthermore, even though
the mimic system had promoted its own documents, other documents were still included within the advice.

8.8. Summary

Navigation tools within Microcosm have all been implemented as filters (with the exception of parts of the mimic system). This allows all of them to access the messages travelling through the segregated communication model (SCM), enabling their monitoring of the system and the user.

The implementation strategy employed in the Microcosm navigation tools causes several problems for its users. For example, they have to query multiple information sources to acquire their navigation advice. These sources can provide contradictory and inconsistent advice. Furthermore, it is left to the user to resolve these advice disparities.

The advice sources in Microcosm can be sub-divided into four groups: navigation, filter, viewer and system centred. The Advisor agent is capable of querying all of these sources, allowing it to arbitrate between them and the users. In addition, the user delegates the responsibilities involved with advice combination to the agent.

The Advisor agent and the advice sources communicate via the direct communication model (DCM) (as the SCM is inadequate for this purpose). However, the DCM is does not provide the process management or ordering that the SCM provides. As a result the Advisor agent is implemented within the hybrid communication model (HCM).

Advice could theoretically be gathered from processes that are not considered to be part of the Microcosm system. For example resource discovery tools that use external databases or information stored within the global computer network (the Internet). However, the processes that provide interfaces to such information sources would need to be adapted to use the DCM. They would also need to be capable of processing the messages sent from the advisor agent and generating the messages needed to reply. Furthermore, the Microcosm system would need to be made aware of documents that the user is advised to see by these external advice
sources. If these two criteria can be fulfilled then the advisor agent can be seamlessly extended to allow advice from sources external to the Microcosm system to be used.

The advice presentation mechanism has been abstracted from the advisor agent, allowing the users to tailor the presentation of advice. The relationship between the agent and the advice presenters is dynamic. This allows the users to choose which advice presenter to run and when to run it. The capability for such customisation is in keeping with that provided by Microcosm itself.

All parts of the Advisor agent system; the advice sources, the agent itself and the advice presenters communicate with each other via a suite of generic protocols. This allows any part of the system to be extended or replaced without the need for the other parts to be changed.

Finally, the Advisor agent has been implemented as an additional navigation tool. With the agent in place the users are provided with an additional source of navigational information. However, when it is removed they can still use the existing navigation tools and the functionality which they provide.
9. Future work

The previous chapters described several specific areas within the design and development of the Microcosm hypermedia system. More specifically, the areas concerned with the implementation of a variety of different communication models and a set of modular navigation tools. Furthermore, the novel use of agents within a hypermedia system has been discussed together with a description of the implementation of such an agent within the Microcosm system. With all of these areas in mind the following sections describe possible lines of future research that the work contained within the previous chapters has highlighted.

9.1. Further development of the DCM

The content of chapter seven shows that the Direct Communication Model (DCM) has several clear advantages over the Segregated Communication Model (SCM) (Wilkins et al, 1993). For example, it displays a greatly reduced communication overhead and a shift from a client server to a peer to peer architecture. However, it also has several distinct disadvantages. For example, the control of a message destination has become the responsibility of the sending process as opposed to the responsibility of the communication system, resulting in a need for the processes to be more intelligent. Furthermore, the DCM provides no process control mechanism and no process sequencing.

The Hybrid Communication Model (HCM) provides solutions to the problems caused by the DCM. However, it is still a compromise and several areas which are described in the following sections should be examined in more detail. The results of such an examination would provide the DCM with the additional functionality required by users that is currently provided by the SCM within the HCM.

9.1.1. Process sequencing

The SCM provides process sequencing (for the filter processes) as a side effect of the communication architecture that exists between the FMS (Filter Management System) and the filters (see chapters three and seven). This compulsory sequencing
has a dramatic effect on the communication overhead within the SCM. Furthermore, the sequencing is only needed in a minority of cases (e.g. in the generation of synonyms, where the synonym generator must be placed before the link databases in the sequence). The majority of scenarios can be implemented without the need for such sequencing and it is questionable whether the benefits of sequencing are outweighed by the disadvantages it produces in the communication system.

Several solutions exist for the implementation of scenarios that need sequencing in a system that does not directly support it. For example, in situations where all parts of the system concerned with the scenario can be altered the following can be implemented. Suppose process A is sending a message about topic X to process B. Now suppose we have another process C that wishes to get the messages with topic X before process B gets them. One solution would be to change the topic of the messages sent by process A to Y, then modify process C to accept these changed messages. When C has finished with them they can be sent on to process B with the original topic X. This scenario is illustrated below in figure 9.1.

![Figure 9.1: The simulation of process sequencing](image)

The above mechanism requires minor implementation changes to processes A and C (the new process). Process B (the existing receiving process) needs no alterations. In the above scenario, the changes needed would be simple to implement and the solution would seem to be a viable one. However, if the situation was more complex or more processes and message types were involved then the solution would soon become difficult to implement. The solution would only be practical in complex situations if the necessary changes to the processes and messages were managed automatically by the system.

A second solution could be the introduction of read only or read/write messages. It is only the latter type of messages that require sequencing as the former cannot be
changed by the processes. As a result they will not require sequencing as no process can change the messages before another process sees them. Obviously the communication model would need to support sequenced sending of messages in addition to parallel (un-sequenced) message sending. However, since the majority of the messages would be read only the communication model would still be significantly more efficient than the SCM.

A communication module, such as the DCM, that does not directly support process sequencing has several distinct advantages. For example, all processes act in parallel, processes are not held up by the need to wait for the slowest processes in front of them in the sequence. These advantages would display the greatest performance benefits in distributed situations, for example, where a user has several linkbases which are operating on remote machines. In the parallel case the link requests would reach the remote machines simultaneously and each linkbase can then process them in parallel. The results from each linkbase would then be transmitted back to the user's machine as each linkbase reaches the end of its processing. The user would receive all the links in the time it takes for the slowest linkbase to respond. In contrast, in a sequenced situation, the user would have to wait for the total time it takes for all linkbases to respond before receiving their links.

9.1.2. Process control

In the SCM process control is provided by the filter management system (FMS) and the document control system (DCS). These two processes act as servers for the filters and viewers respectively. However, in the DCM the FMS has been removed completely and the DCS has been reduced to a peer of the viewer and filter processes with no control over their communication. As a result the DCM does not have any process control facilities and all processes exist at an equal level.

The peer to peer structure of the DCM allows Microcosm to exist as a background process within the user's computing environment. For example, where the user had to start the whole of Microcosm through the DCS before they could use it, they can now start the parts of the system independently as they need them. In most situations, users need not explicitly start any of the Microcosm processes themselves, instead they will be started as an indirect result of their actions. For example, suppose the user wished to follow links from their favourite word
processor. They simply highlight the word they are interested in and select follow link from its menu. The linkbase processes (and any other ancillary processes such as the available links process) will be started up without requiring any participation from the user.

The above scenario is impossible to implement under the SCM, but within the DCM all that is needed is a process that is run by the word processor when the user selects the follow link option. It is this process that becomes responsible for starting up the linkbases etc. allowing the user to follow the links. Such a process would exist at the same level as the processes it in turn starts up. Unlike the SCM, any process within the DCM can start other processes. Once the processes are started it is their responsibility to connect to the DCM and through it to the rest of the system.

If users still wish to use the DCM version of Microcosm in the same manner as the SCM version, then simplified versions of the DCS and FMS can be constructed. These would fulfil the same roles as their counterparts in the SCM but would be less complex as they would no longer be responsible for any communications control. Furthermore, they would no longer be required to maintain complex system information (such as the re-use state of any of the document viewers) since such information would be implicitly maintained by the communication system itself or by the processes concerned.

The DCM currently still maintains the distinction between viewer and filter processes. Such process segregation was maintained in order to ease the design of the DCM prototype discussed in chapter seven. However, there is no real need for the distinction to be maintained. Removing the distinction and treating all processes as equal would allow the following:

- The linkbases (a filter in the SCM) could be seen as a viewer that allows users to view link databases.
- The mimic system (see section 4.2) could be implemented as two processes instead of three (i.e. mimic generator and mimic viewing/control process).
- The map and history processes (currently filters) could be used as viewing processes for their respective document types, while still maintaining their abilities to control the other processes, which they need to function fully.
The DCM provides facilities for any process to monitor the communication system, allowing for logging of events. The use of such facilities could lead to the system being able to adapt itself to a profile of the user. Such a profile can be constructed by the system by monitoring the messages produced as a result of the user's actions.

The DCM provides facilities for processes to detect communication errors, for example, messages being sent to processes that are not currently running. Such error detection opens the way for the construction of processes that exist at the same level as all the other processes but that are designed to cope with the errors detected.

The logging and error detection facilities provided by the DCM will lead to a more adaptive Microcosm system that can, in theory, remove some of the burdens produced by the user needing to know more about the functionality of the current DCM system components.

A full implementation of such an adaptive system would allow users to make use of their everyday tools within the computer environment, but at the same time allow them to seamlessly access the extra functionality provided by Microcosm. The system would provide them with hypermedia functionality at a level similar to that of the clipboard found within all modern GUIs.

9.1.3. Centralised system information

The DCM requires a greater degree of complexity within its processes than the SCM in order to take full advantage of the novel facilities that it provides, such as the ability to control the routes of messages and the ability to know more about the rest of the system via the centralised system information provided by the DCM.

Processes are able to query the DCM for information concerning the other processes currently running, what topics are available to hold conversations about and how many processes are registered for each topic etc. This wealth of information is available to all processes through the API provided by the DCM.

However, since the DCM was originally designed and constructed Microcosm has been extended to provide a registry facility. This replaces the layered ini. file system, used by the processes within the SCM, with a hierarchical storage facility
that is capable of maintaining arbitrary data during and across sessions. The DCM
maintains its data internally, meaning that processes wishing to access it must use
its API.

The presence of the registry essentially leads to a duplication of functionality
within the Microcosm DCM system. It would seem sensible to move the centralised
system information from the DCM to the registry. This would simplify the DCM
API (leaving communication functionality but removing the information access
functions) and standardise both the information storage and access within
Microcosm as a whole.

9.1.4. Inter-process communication protocols

The DCM discussed in chapter seven allows for cleaner interactions between its
processes. For example, the control messages needed in the dialogue between the
link producers and the available links process were eliminated. This was achieved
by making the available links process more aware of the rest of the system (i.e. the
number of link producers currently running). Such a solution would have been
impossible within the SCM as no single area of the system actually knows what
processes are currently running and more fundamentally the functionality that each
process can provide.

The DCM provides every process with access to its central system information
storage mechanism. This has two benefits:

- It allows the processes to become more intelligent, reducing the number of
  messages within the system.

- It allows the construction of protocols that do not rely on the contents of the
  messages which are exchanged between the processes.

Furthermore, the messages within the DCM contain information pertaining to the
sender of the message. This allows the receiving process to know which process sent
the message and to act accordingly. For example, viewer processes can colour their
buttons depending on which link producer sent them the associated messages. This
would also be possible within the SCM but would require special coding in the link
producers.
The DCM provides four mechanisms for processes to communicate with one another. These are:

- Process to process
- Name oriented
- Topic oriented
- General broadcast

The processes must become more intelligent if they wish to use all four communication mechanisms. In addition, new protocols need to be developed that make the most efficient use of the DCM and the new facilities that it provides.

9.1.5. Seamless distributed heterogeneous architecture

The peer to peer nature of the DCM significantly simplifies its construction. The complexity within the DCS and FMS has been removed, but is replaced with a greater overall complexity within the processes themselves. Such changes effectively make the re-implementation of Microcosm within different environments much easier.

The DCM communication module code is the only code that needs to be platform specific (within the context of the process communication). Once this code has been re-implemented the rest of the system will be virtually identical on all platforms. The DCM communications module will hide the platform specific inter-process communication requirements of each platform.

Each machine in the heterogeneous network will have its own communications module running on it. These modules will communicate with each other by means of a standardised communication mechanism, such as sockets (Comer, 1991; Renaud, 1993). This inter-machine communication method does not have to be the same as that implemented at the intra-machine level. Furthermore, only the DCM module needs to know about the inter-machine protocol employed in the system.

All DCMs will exist at an equal level, each will have knowledge about its own processes and will be able to poll other DCMs to collect information about their
processes. Such a scenario will allow any process on any machine to effectively communicate with any other.

Due to the absence of process sequencing, all communication will be in parallel and so will not be restricted by the slowest machine/process pairing or the communication route to any great extent. Each DCM will have the responsibility of maintaining information about its own set of local processes and any remote processes that are currently being used.

9.2. Agents for monitoring and maintenance

Chapter six discussed the use of agents for three tasks within an open hypermedia system. These are:

- As a navigation aid
- For monitoring the large volumes of dynamic information associated with a hypermedia network
- For maintaining the structural integrity of a hypermedia network

Chapter eight dealt with the implementation of an agent as a navigation tool. The following sections discuss the remaining two uses in more detail.

9.2.1. Agents for monitoring hypermedia network content

A hypermedia network represents the content of a plethora of documents and links. This content will be highly dynamic when considered as part of an open hypermedia system. It seems reasonable to employ agents as information monitors and locators within the hypermedia network. For example, users could construct agents that are given the sole task of monitoring documents, both inside and outside of the current hypermedia network. These agents will automatically construct links to new documents and therefore seamlessly include them into the corpus of material already represented by the existing hypermedia network.

Agents can also be designed that monitor the existing content of the hypermedia network in order to construct new links between related sections of material already
within the existing corpus. In both cases the user will effectively be delegating some of the responsibilities associated with authoring links to the agents.

Open hypermedia systems such as Microcosm are ideal environments for the development of such monitoring agents. All documents within Microcosm are kept in their native format and, more importantly, the links are stored in an easy to interpret ASCII format. Both characteristics provide the agents with easy access to the major components of the hypermedia network structure.

Monitoring agents will provide the greatest benefits in situations where the hypermedia network is created by a group of authors (e.g. within a CSCW environment (Wiil & Leggett, 1993; Ellis et al, 1991; Haake & Wilson, 1992; Streitz et al, 1992; Trigg et al, 1986)) or where the hypermedia network contains large volumes of dynamically changing information (e.g. up to date information related to stocks and shares).

In the former situation the agent will work on behalf of the user, locating new information contributed by other members of the co-operative work group. When information is located the agent could inform its user or seamlessly create links to the new documents. The agent in the latter scenario will perform a similar task but would automatically include the latest information into the current hypermedia network instead of requiring other authors to create it.

Initially agents will only be able to monitor information that contains text or numerical data, since many proven search techniques exist in these areas allowing agents to be easily constructed. However, there is no reason why agents of the future should not be set the task of monitoring the full spectrum of media types available to the hypermedia author. As search techniques for image, digital video and sound become available new agents can be designed which allow users to automatically locate and integrate documents from the respective media groups.

9.2.2. Agents for maintaining the structural integrity of hypermedia networks

Open hypermedia by its nature is highly dynamic, the content of both the documents and the links which connect them are likely to change dramatically as the hypermedia network is developed and used. However, hypermedia networks
are fragile since links can be rendered useless by the editing or removal of documents to which they relate. The majority of open hypermedia systems gloss over these editing problems, assuming that a document's content will not change once it has been included in the hypermedia network. However, in order for hypermedia to be widely accepted such editing problems must be solved (Davis et al, 1992).

Open hypermedia systems such as Microcosm store link information separately from the documents that they connect. Such separation allows either the document content or the links to become out of step with the other. Maintaining link and document consistency becomes difficult for small hypermedia networks, as the network's size and scale increase the task soon becomes impossible. Within the Microcosm project alone three possible solutions to such maintenance issues have been discussed. These are:

**Link aware editors** - These will allow the user to edit the document whilst automatically updating any related links.

**Linking to document versions** - Links are made to specific versions of a document. Link consistency is only guaranteed between the versions of the document that existed when the link was created (Haake, 1992; Osterbyte, 1992).

**Embedding anchor context** - Context information for each anchor of a link is embedded within the link itself.

All three solutions cause problems. For example, by their nature open hypermedia systems such as Microcosm draw some of their power from the ability to allow users to link any type of document supported by the tools within their environment. Such variety means that for a complete solution using link aware editors a comprehensive suite to cover all document formats must be developed. Every new media type will require a new link aware editor in order to maintain a consistent system.

As documents grow in their complexity the task of constructing link aware editors also grows in complexity. For example, the task of constructing a link aware editor for a document type such as RTF (Rich Text Format) is phenomenally
complex. Furthermore, in a heterogeneous environment editors will be required for each platform for each required document type.

Versioning control systems such as SCCS or RCS exist on all major hardware platforms. Extending a system such as Microcosm to be aware of versioning could be achieved by changing only a small subset of its constitutive processes. An author would link two documents together as normal, but in reality the link would only exist for the respective current versions of those documents. There would be no guarantee that the integrity of the link would be maintained across earlier or later versions of the linked documents.

The reconciliation of links in one version of a document against all other versions of the same document would be possible. However, to achieve such reconciliation over all versions of all documents and all links would obviously be computationally expensive. Furthermore, it would be an arduous and time consuming task for the user to perform.

The theory behind embedding anchor context within the link is that when the link is followed or searched for the system will be able to decide if its integrity has been maintained. Furthermore, the context can be used to regain this integrity should it be lost. For example, suppose a link existed from the word 'contrast' within a document. At some point after the link had been created the document was edited and ten lines of text before the original link anchor, for the link in question, had been removed. The user opens the edited document and the system discovers that the text held in the link as its start anchor differs from the text in the document at the anchor's original position. A simple solution would be to search for the link anchor text elsewhere in the document. However, if only the word 'contrast' is considered in the search there is no guarantee that all other occurrences of the word in the document will have the same context. A solution is to embed several of the words from either side of the link anchor in the link. When the above situation occurs it is more likely that the new occurrence of the link anchor resulting from the context sensitive search will be correct.

It would seem that a combination of the latter two solutions will provide a workable solution to the problems of link integrity. However, for large and complex hypermedia networks such on-line fixing of links would seem not to be beneficial to the users. There is a danger that the users will be left waiting for the system to fix a large number of broken links every time they open a document. Such intervention
by the system will add further to the cognitive load associated with the traversal of hypermedia links.

An agent that could work autonomously, on behalf of the user, fixing links broken by document editing would seem to be a solution to the problem of hypermedia network integrity. The agent could work on the documents and links, checking their integrity whilst the user was using the system, or even whilst the system was inactive (e.g. overnight). When the agent found links that had been broken it could use the embedded context to fix them. If this was not possible, for example if the context sections have also been removed from the document in question, then the agent could report to the user (e.g. via e-mail), allowing them to repair the link manually.

If the system contains a versioning mechanism then the agent can be given the arduous task of resolving a link against newer and older versions of its anchor documents. This will broaden the scope of the link and allow it to be followed from more versions of its anchor documents. Once again, if the resolution is impossible then the agent can flag the user's attention at a convenient time and draw them to the problem.

It would seem that the combination of the agent paradigm with the realms of hypermedia system research will offer great potential in the areas of monitoring and maintaining the associated hypermedia networks. In addition, the area of navigation can also benefit from such agent integration as discussed in chapter eight.

9.3. The advisor agent

The content of section 8.1 describes problems with the current implementation of the navigation tools within Microcosm. The modular nature of the system means that the navigation tools are implemented as discrete autonomous entities. Each tool provides users with advice derived from its own local knowledge. As a result of this fragmented implementation users can be provided with information which at times can be both inconsistent and contradictory. Furthermore, users must query all tools to gain the overall picture of navigation advice that is offered. It is the users
themselves that are given the burden of combining the advice from the navigation tools.

Section 8.3 continues by suggesting that the processes within the system can be changed to provide advice to a central agent process, the advisor agent. It then becomes the job of this agent to combine advice from the separate navigation tools. The users are finally provided with a complete picture of the navigation advice from the agent itself.

The advisor agent mechanism can be divided into four areas (see section 8.5.1.2) each of which is described in detail within chapter eight (including implementations for use with the Microcosm system). These are:

- Advice generation
- Advice collection
- Advice amalgamation
- Advice presentation

The advice generation mechanism is specific to the source of the advice. For example, the mechanism employed within the local map filter is completely different to that of the history filter. New sources of advice within the Microcosm system could be located, each requiring a new advice generation mechanism uniquely designed to fulfil its particular needs. At present, the only type of advice the advice sources can produce is advice about which document to see next. However, there is no reason why they should not be extended to include new types of advice as highlighted in section 5.4.6 (the active path guide).

The advice collection mechanism is generic to all advice sources, allowing new sources to be created, or old ones changed internally, without any need to change the agent itself. The mechanism is openly designed and will be able to support new advice generation types if and when they are included within the sources.

The advice amalgamation mechanism is very primitive. The weights used to combine the advice from each source are effectively constant (until changed by the user). Furthermore, no mechanism exists for either the agent or the advice sources to change the weights autonomously. It is conceivable that some form of learning
mechanism (such as neural networks) could be employed within the advice amalgamation mechanism. This would allow the agent to adapt to the users, learning about their requirements and improving the accuracy of its advice.

Section 8.5.4 presents three advice presentation mechanisms, discussing the abstraction of the advice presentation mechanism from the advisor agent. Such abstraction allows new advice presentation modules to be investigated without the need to change the design of the advisor agent itself.

The use of an agent to aid users as they navigate through hypermedia networks opens up a large area of research. The implementation discussed in chapter eight is simply one possibility out of a multitude of such designs. For example, in addition to the above areas, there still remains the question of how users will perceive advice presented to them from an agent of this kind. Will they treat such advice in the same manner as advice from existing ordinary navigation tools (such as the history filter)? Maybe the autonomous nature of the agent will cause them to ask for explanations of how it arrived at its current advice. Such questions can only be answered by further research into this area, involving trials of such an agent on users of a system like Microcosm.
10. Conclusions

This thesis outlines the four main types of navigation tool and describes how they are used during the exploration of hypermedia networks. The design and development of the Microcosm open hypermedia system is then described, followed by the discussion of a variety of navigation tools designed to help the user as they move around their hypermedia network.

Agents are proposed as a solution to the problems caused by the fragmented implementation of the Microcosm navigation tools. However, an agent can not be implemented within the standard SCM version of Microcosm. This prompted the implementation of the alternative communication models (the DCM and HCM). These models are used in the implementation of an agent designed to integrate the navigation information provided by the parts of the Microcosm system.

The following sections discuss the conclusions that were arrived at during the research detailed in the preceding chapters.

10.1. The need for navigation tools

Many works have been published on the disorientation of users within hypermedia networks (Conklin, 1987; Bernstein, 1988; Marchionini & Schneiderman, 1988; Utting & Yankelovich, 1990; Hutchings, 1993). A popular hypothesis for the cause of this disorientation is that users suffer from a lack of spatial and temporal context (see section 2.1). It is this lack of context that results in them being *lost in hyperspace* (Van Dam, 1988). Furthermore, in an effort to make hypermedia more usable, its creators have turned to navigation tools, such as overviews, guided tours and histories, to provide users with the much needed contextual information. These tools provide the context information in the following ways:

**Histories** - provide users with temporal context, by showing them which parts of the hypermedia network they have already seen.

**Links** - allow authors to show users relationships between areas of the hypermedia network, providing them with a limited spatial context.
Overviews - provide users with a scalable sense of context, depending on the scope of the overview in question.

Tours - provide users with both temporal and spatial contexts, by linearising sections of the hypermedia space.

The Microcosm system (discussed throughout this thesis) provides users with a comprehensive set of navigation tools (e.g. local map, history, guided tour etc.) and yet users still seem to get disorientated (Hutchings, 1993). Possible reasons for this continued disorientation (such as the contradictory and fragmented advice provided by the existing navigation tools) are discussed in chapter eight and in section 10.2.

It can be argued that the introduction of tools such as full text retrieval (Frisse, 1987; Frisse, 1988; Li et al, 1992; Salton, 1989; Salton & Buckley, 1989) will alleviate the problem of users being disorientated. Such a system places the user just one search term away from their goal at all times. However this is not always the case. For instance, what happens when the user cannot create the correct search term? They are left jumping around the hypermedia system and will have no hope of gaining either the temporal or spatial contexts that they need. Furthermore, the beauty and elegance of the hypermedia philosophy is that users can follow links from one piece of related material to the next, learning as they travel. Therefore, it would seem obvious that even in systems that provide full text retrieval there is still a need for navigation tools.

10.2. Integration of navigation tools

Chapters four and five discussed the implementation of a set of navigation tools for use within the Microcosm system. The modular nature of Microcosm dictates that the navigation tools should be implemented as separate autonomous processes. Each tool, due to this autonomy, will have its own point of view derived from its local knowledge. Such a disparate approach has the following drawbacks:

- Users can be presented with information that contains inconsistencies and contradictions.

• Users must query all navigation tools in order to gain a complete overview of their current spatial and temporal context.

• It is left to the user to merge information gathered from each navigation tool before they can interpret their current situation.

However, within the Microcosm system it is not simply advice provided by the navigation tools that can aid the user in their navigation around the hypermedia network (as shown in section 8.5.1.1). Furthermore, users could benefit from advice produced by parts of the system that do not even have a user interface.

With all the above in mind, the purpose of the advisor agent (described in chapter eight) is to integrate the advice produced by the navigation tools and other parts of Microcosm. The agent provides a mechanism by which parts of the system that the user would not normally interact with can contribute to the overall picture of their current situation.

The implementation of the advisor agent allows it to autonomously integrate advice generated by any part of the system (from navigation tools to the document management system). This effectively integrates the advice generators into a single advice source. The user can query this single source without needing to deal with the inconsistencies and contradictions associated with the existing disparate navigation systems.

The advisor agent is constructed in such a manner as to allow the user to still use the existing systems as normal, even when the agent is not running. Furthermore, the agent is capable of adapting to changes in the Microcosm system (e.g. filters being removed) and will only use advice sources that are currently available to it. This allows the user to still dictate which parts of the system to execute and when to execute them, whilst at the same time being able to utilise the additional functionality offered by the advisor agent.

The mechanism of advice collection and amalgamation opens the way for new advice sources to be developed that do not need a user interface. The agent becomes responsible for all interactions between the user and the new sources. The agent is perceived by the system as being at an equal level to its other components. This allows the system to behave as normal when the agent is not running. In addition,
the DCM provides the agent with a mechanism that allows it to collect and use information from processes operating outside of the Microcosm system.

The advisor agent provides a platform on which the effect of the presentation to the user of autonomously unified advice drawn from many areas within a hypermedia system can be researched. For example, it could be argued that a side effect of such unification would be to implicitly reduce the complexity and confusion caused by the plethora of different interfaces currently provided within Microcosm. Furthermore, such simplification might allow users to navigate through the hypermedia network without becoming so disorientated.

10.3. Navigation advice

The concept of navigation advice was introduced during the discussion of the advisor agent implementation in chapter eight. It is used to describe information collected from different parts of the Microcosm system by the advisor agent. This advice is used by the agent as a source for the advice that it will present to the user.

Navigation advice comes mainly from the navigation tools within Microcosm. However these are not the only sources of such advice. Section 8.5.1 describes advice generated by other filters (such as the linkbases and the full text retrieval filter). The generation of navigation advice is source specific, but the introduction of a generic protocol between the advice source and the agent allows the agent to gather advice from any source, independently of its generation mechanism. This ability to treat all advice sources as equal allows new advice sources to be introduced without the need to alter the agent itself.

Furthermore, the advice sources can be independently extended to produce different types of advice. Once again, through the use of the generic protocol, these new types of advice can be transmitted to the advisor agent without affecting the rest of the system. The advice generation and collection mechanisms implemented within the advisor agent system provide a flexible and extensible framework to study the use of an agent style navigation aid within an open hypermedia system.

The advisor agent itself can be thought of as a black box. The advice combination mechanism implemented within it can be changed without affecting
the rest of the system (i.e. advice generation, collection and presentation). Furthermore, the abstraction of the advice presentation from the advisor agent allows different advice presentation strategies to be experimented with, without the need to alter the remainder of the advisor agent.

The modular nature of the advisor agent system parallels that of its parent system (i.e. Microcosm). The modular nature of Microcosm allows it to be used as a research platform for the area of open hypermedia. Furthermore, the modular nature of the advisor agent's design will allow it to act as a similar platform in the research area of agent use within the realms of open hypermedia navigation tools.

10.4. Communication models within Microcosm

Chapters three and seven discuss the inefficiencies of the Segregated Communication Model (SCM) version of Microcosm. These inefficiencies can be summarised as:

- High communication overheads caused by the chain filter system.
- Non-scalable architecture.
- No direct communication routes between all parts of the system
- Large minimum system (i.e. DCS, FMS, one viewer and necessary filters).
- Very much a foreground system, with the whole system needing to be executed before the user can take advantage of the functionality that it offers.
- Processes can only be started through the use of the DCS and FMS.

In contrast the Direct Communication Model (DCM) version of Microcosm does not suffer from any of the above inefficiencies. Furthermore, it provides areas of additional functionality that are of use to all processes, including agent type processes. These are:

- The ability for agent processes to control the processes within Microcosm, either through the normal functionality provided by the underlying
operating system, or by being able to hold direct bi-directional conversations with them.

- System wide information that the agents can utilise allowing it to know more about the Microcosm system currently being used.
- A higher general level of intelligence within the Microcosm processes, due to their ability to control where their messages are sent.
- Agents can exist at the same level as all other processes within the Microcosm system.
- An API for the agent to monitor all communication within the DCM, without the need to specially adapt the existing Microcosm processes.
- An API for the agent to detect communication errors within the system, allowing for autonomous system and agent adaptation to handle the errors.

The advisor agent (discussed in chapter eight) was implemented within the Hybrid Communications Model (HCM) version of Microcosm discussed in chapter seven. Such an implementation would have been impossible within the SCM due to the constraints it places upon processes. The HCM provided the functionality of the DCM whilst maintaining the underlying system support of the SCM. The DCM functionality was used extensively by the advisor agent, for example, for communicating with other Microcosm processes which it treated as advice generators (see section 8.5.1).

These advice generators are simply processes that already exist within Microcosm, but that have been extended in two ways:

- To take advantage of the new facilities provided by the DCM.
- To allow them to autonomously generate advice for the agent, on behalf of the user.

As a result of these extensions the processes became more aware of the surrounding environment and were able to communicate directly with both the agent and other areas of the system.
The DCM provides an environment that is more suitable for research into the areas associated with the integration of agent processes with the functionality provided by a hypermedia system. This suitability is mostly due to the DCM bringing Microcosm nearer to the level of an operating system service. In addition, the DCM provides agents with support for the low level integration they need in order to control other processes autonomously on the user's behalf. Furthermore, the facilities provided by the DCM (such as communication monitoring and error detection) allow agents to adapt themselves to the requirements of the user and the constraints placed upon them by the user's environment.

10.5. The use of agents in hypermedia systems

Open hypermedia is by its very nature highly dynamic. This dynamism is displayed both in the document content and the links that connect them. An open hypermedia network can encompass large volumes of information, represented by both the document content and the links. It is these features that make such networks suitable environments for agent type processes. Agents are designed to ease the burdens imposed on users by large volumes of dynamic information. Therefore it would seem that hypermedia and agent technology are ideally suited to each other.

It has been argued that hypermedia functionality should be provided by the operating system (OS) (Pearl, 1989; Davis et al, 1992; Lewis, 1993). Such an implementation would ease some of the problems suffered by today's hypermedia systems, for example:

- The tracking of files as they are moved around within the file system.
- Integration into all processes within the user's computing environment, in the same manner as the clipboard functionality that can be found behind the cut, copy and paste menu options on many processes in a graphical user interface.
- All applications would become link aware by default.

Furthermore, the OS would provide additional central APIs for processes to take advantage of the hypermedia functionality it now offers. Agents are another type of
functionality that will gain enormously by being implemented at an OS level (Palaniappan et al, 1992), for example:

- Agents will be able to be integrated into every process.

- All processes will be capable of being agent aware.

- Agents would be more robust as they could take advantage of the OS level functionality, allowing them to track files etc.

In addition, the OS would provide APIs that allow the agents to access and control other processes and vice versa. It would seem that not only are hypermedia and agents of use to each other, but that they would both greatly benefit from being implemented at the OS level. Such an implementation will open the way for a new generation of computing environments based on a new set of OSs.

However, these new OSs do not seem to have been implemented at the present time. One only has to look for signs in the latest OS environments developed for commercial release, to see that they offer little hope in the realms of either hypermedia or agents. Chicago, Microsoft's latest offering (Jackson, 1994; Tisdall, 1994), or even X11R6, the latest project from X Consortium Inc. (Heller, 1994), do not provide the functionality discussed above at any level. Once more, it will be left to the application developers to fully construct and include any functionality in either area.

Of the three platforms covered by the Microcosm system the X-Windows version has the greatest potential to support the four type of agent tasks discussed in section 6.1. This potential is provided by the underlying UNIX operating system in the following ways:

- Automation - Processes such as \textit{filter} can be set up so that it will automatically sort the users mail into different folders based on their content.

- Monitoring - UNIX supports many types of daemons that constantly monitor system parameters such as disk space, communication sockets and so on.

- Future execution - UNIX provides processes such as \textit{cron} and \textit{at} for repetitive and one-off command execution respectively.
However, UNIX lacks the services sought by the agent designers of many systems. Furthermore, UNIX does not stipulate that process designers should provide any sort of generic APIs allowing any process to control or query other processes in the system.

Some support of a kind is being offered in the form of cross application communication systems (CACS), allowing new types of process interoperability. Examples of such CACS can be found in System7 for the Apple Macintosh platform and within the latest versions of Microsoft Windows and Microsoft WindowsNT for the PC platform. System7 and Microsoft Windows provide apple events (Apple, 1991) and OLE2 (Microsoft, 1993b; Microsoft, 1993c) respectively. These systems follow on from work started by both DDE (Dynamic Data Exchange (Petzold, 1992)) and OLE (Object Linking and Embedding (Klemond, 1992)). They define standards which must be met by the application developers in order to describe their products as being fully system aware.

The primary aim of both is to shift developers away from producing the monolithic applications that currently fill our file systems. Instead, it is argued by the proponents of the above systems that the developers will produce small, simple applications that gain their power from working together. For example, newer will the user be forced into buying a spell-checker, thesaurus and grammar-checker every time they purchase a word-processor. Instead they will be free to buy them as separate packages from different vendors, safe in the knowledge that these programs will work together with their word-processor. The reason why they will all work with each other lies in the standards enforced by the CACS (e.g. OLE2 or apple events).

However, this new level of application interoperability will also allow designers of agent and/or hypermedia style applications to access some of the functionality they need in order to integrate their processes into the user's existing environment. Whilst this is by no means a fully comprehensive answer, it will at least provide a partial solution that will suffice until the OS developers offer the necessary functionality.

Only when both the agent and hypermedia functionality is provided at the OS level will it become commercially viable for every application developer to take advantage of the additional benefits offered to them and integrate such functionality
into their applications. Until such a time neither the agent or the hypermedia paradigms will become commonplace in every user's computing environment.
Bibliography


Knowledge Based Environments for Teaching and Learning, Rapallo, Italy, May 1991.


Engelbart D. G. & English W. K., (1968) A Research Centre for Augmenting Human Intellect, In AFIPS Conference proceedings, 33(1)


