HIPPO – An Adaptive Open Hypertext System

Paul K. Newton

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy, October, 1998

Acknowledgements

I would like to thank all my friends and colleagues who have helped me with this research. I would like to extend particular thanks to my supervisor, Prof. David Brailsford for his guidance and to Dr Helen Ashman for her invaluable comments on this thesis. I would also like to thank the members of the Electronic Publishing Research Group for their continued support and encouragement.

Paul Newton
1998

Contents

Overview **1**

1 Introduction To Hypertext **5**
1.1 Defining Hypertext Terms 7
1.1.1 The Node 7
1.1.2 The Link 10
1.1.3 The Anchor 15
1.2 Advantages Of The Hypertext Approach 17
1.3 Problems With Early Hypertext Applications 19
1.3.1 Navigation – the *disorientation* problem 19
1.3.2 Premature Segmentation 22
1.3.3 Maintaining Hypertext Structures 22
1.3.4 Cognitive Overhead 23

2 Developing The Hypertext Model **24**
2.1 Open Hypertext 25
2.1.1 Integrating Applications 28
2.1.2 Hyperbases: hypertext-specific storage engines 31
2.1.3 Computation in open hypertext 31
2.1.4 Extensibility and tailorability 33
2.1.5 Formal models and hypertext standards 34
2.2 Distributed and Collaborative Hypertext 36
2.2.1 Degrees Of Distribution 36
2.2.2 Maintaining Distributed Hypertexts 38
2.2.3 Concurrency and Access Control 39
2.2.4 Modes Of Collaboration 40
2.3 Adaptive Hypertext 42
2.3.1 Adaptive Features 42
2.3.2 Adaptive Techniques 44
2.4 Summary 45

3 Fuzzy Anchors **47**
3.1 Limitations Of The Current Anchoring Model 49
3.1.1 Anchors As First-Class Objects 49
3.1.2 Over-Specific Addressing Mechanisms 50
3.1.3 Static Anchors 52
3.2 Fuzzy Anchors 53
3.3 Advantages Of Fuzzy Anchors 55
3.4 Adaptive Fuzzy Anchors 57
3.5 Anchorbases 58
3.6 Prototype Implementation 60
3.6.1 Implementation Environment 60
3.6.2 Internal Representation Of Fuzzy Anchors 61
3.6.3 Presentation Of Fuzzy Anchors 64
3.6.4 File Format Of Anchorbases 65
3.6.5 Tools And Functionality 68
3.6.6 Adaptive Server 74
3.7 Summary 76

4 Building Adaptive Trees Using Linkbases **79**
4.1 Separate Link Structure Using Linkbases 81
4.2 Using Linkbases As Building Blocks 83
4.3 Limitations Of The Linkbase Approach 84
4.4 Linkbase Inheritance Trees 85
4.4.1 Advantages Of Linkbase Trees 88
4.4.2 Extending Trees Using Linkbase Visibility 89
4.5 Adaptive Trees 91
4.5.1 Weighted Links 92
4.5.2 Weighted Inheritance Relationships 93
4.5.3 Adapting Inheritance Values 95
4.6 Distributed Trees 96
4.7 Combining Weighted Links, Weighted Trees and Fuzzy Anchors . . . 98
4.8 A Simple Example 99
4.9 Implementing Adaptive Linkbase Trees In The HIPPO System 103
4.9.1 Linkbases In The HIPPO Prototype 104
4.9.2 Linkbase Format 105
4.9.3 Linkbase Trees In The HIPPO Prototype 108
4.9.4 Linkbase Tree Format 110
4.9.5 Extending The Adaptive Server 111
4.10 Summary 112

5 HIPPO+ – Distributing The HIPPO Model **114**
5.1 Summary Of The HIPPO Prototype 116
5.2 The Distributed HIPPO+ Model 119
5.3 Advantages Of A Distributed HIPPO Model 120
5.3.1 Scalability 121
5.3.2 Openness 121
5.3.3 Distribution 121
5.3.4 Heterogeneity 122
5.3.5 Interoperability 122
5.3.6 Extensibility 122
5.3.7 Computation 123
5.3.8 Tailorability 123
5.4 Existing Distributed Architectures 124
5.4.1 Inter-Process Communication 124
5.4.2 Distributed Frameworks 125
5.4.3 Compound Documents 126
5.5 The CORBA Model 127
5.5.1 Object Request Broker (ORB) 128
5.5.2 Interface Definition Language (IDL) 129
5.5.3 Object Services 130
5.6 Implementing The HIPPO+ System 134
5.6.1 Node Browser 134
5.6.2 HIPPO+ Buffers 135
5.6.3 HIPPO+ Services Using ONC-RPC 138
5.6.4 HIPPO+ Service Interfaces 141
5.6.5 Execution Manager 142
5.6.6 HIPPO+ Registry 143
5.7 Extending The HIPPO+ Computational Model 145
5.7.1 Query Interface 147
5.7.2 HIPPO+ Trading Service 148
5.7.3 Hypertext Component Hierarchy (HCH) 150
5.8 Summary 152

6 A Proposed Adaptive Model For HIPPO+ **156**
6.1 Advantages Of An Adaptive HIPPO+ Model 157
6.2 Example Adaptive Services 158
6.2.1 Example 1 158
6.2.2 Example 2 160
6.2.3 Example 3 162
6.3 Service Contexts 166

6.3.1 User Stereotypes 166
6.3.2 Goal-Based Model 168
6.3.3 Document Objects 170
6.4 An Example Using Document Objects 171
6.5 Weighted Services 175
6.6 Automatic Identification Of Useful Services 177
6.7 Combining With Other Adaptive Models 179
6.8 Summary 179

7 Discussion **183**
7.1 Overview 183
7.1.1 Adaptive Fuzzy Anchors 185
7.1.2 Adaptive Linkbase Trees 187
7.1.3 Adaptive Distributed Systems 189
7.2 Future Research 192
7.2.1 Developing Fuzzy Anchor Model 192
7.2.2 Developing Linkbase Trees 198
7.2.3 Developing HIPPO+ 200
7.2.4 User Evaluation 205
7.3 In Conclusion 207

A Early Hypertext Applications **208**
A.1 Augment/NLS 209
A.2 Xanadu 209
A.3 TEXTNET 209
A.4 ZOG/KMS 210
A.5 NoteCards 211
A.6 Intermedia 212
A.7 Neptune 212
A.8 Guide/OWL 213
A.9 HyperCard 213

B Open Hypertext Systems **215**
B.1 Sun's Link Service 215
B.2 SP1/2/3 216
B.3 Microcosm 218
B.4 Chimera 220
B.5 Dexter Hypertext Reference Model 221
B.6 MultiCard 222
B.7 D^2 224
B.8 HyperDisco 225

B.9 The Trellis Model 226
B.10 The Hypertext Design Model (HDM) 227
B.11 MAX 228
B.12 Hyper-G 229
B.13 MHEG 230
B.14 Hypermedia/Time-based Structuring Language (HyTime) 230
B.15 Portable Document Format 232
B.16 World Wide Web 234

C Fuzzy Anchor Specification **238**
C.1 Lexical Specification 238
C.2 Grammar 239

D Linkbase Specification **241**
D.1 Lexical Specification 241
D.2 Grammar 242

E Linkbase Tree Hierarchy Specification **244**
E.1 Lexical Specification 244
E.2 Grammar 245

F Linkbase Tree Example **247**

List of Figures

1.1 A simple hypertext example 7
1.2 Frame-based nodes in Aquanet 9
1.3 Hierarchical links and cross-hierarchy links 12
1.4 The anchor provides an endpoint for the hypertext link 15
1.5 Anchor definitions using spans of text 16

2.1 Open hypertext using link services 26
2.2 The HAM hyperbase engine 32
2.3 CGI link computations in the World Wide Web 33
2.4 Hypertext interchange models 36
2.5 Distributed hypertext in the WWW 37

3.1 Separation of anchors, nodes and links 50
3.2 Fuzzy sets use partial truth values to model uncertainty 54
3.3 Example of a fuzzy anchor 54
3.4 A typical example of a fuzzy anchor on a graphic image 55
3.5 Initial specification of a fuzzy anchor 57
3.6 Fuzzy anchor after adaptive modelling 58
3.7 Separating anchor definitions into anchorbases 59
3.8 Example of Acrobat Exchange software in use 62
3.9 Logical representation of fuzzy anchors 63
3.10 Fuzzy anchors using path descriptions 64
3.11 A fuzzy anchor using a matrix of values 64
3.12 Using colour to represent anchors 65
3.13 A complete fuzzy anchor definition 68
3.14 The HIPPO toolbar 69
3.15 Selecting anchors in HIPPO 69
3.16 Defining new anchors in HIPPO 70
3.17 Creating a new anchor in HIPPO 71
3.18 Editing anchor details 72
3.19 Reusable anchor patterns in HIPPO 74
3.20 Remote adaptive server 75

3.21 Adapting fuzzy anchors in HIPPO 76

4.1 Using linkbases in open hypertext applications 82
4.2 Update problems with linkbases 84
4.3 Object-Oriented inheritance hierarchies 86
4.4 Linkbase inheritance trees 86
4.5 A simple engineering linkbase hierarchy 87
4.6 Deriving a new engineering linkbase 89
4.7 Access control in inheritance hierarchies 90
4.8 Weighted hypertext links 93
4.9 Weighted linkbase hierarchy 94
4.10 Adapting a linkbase tree 96
4.11 Distributing linkbase trees 97
4.12 Simple biology link definition 100
4.13 A medical linkbase tree 101
4.14 Combining weighted links with weighted trees 101
4.15 Defining a fuzzy anchor on a picture of the human body 102
4.16 Using fuzzy anchors to amend the link weights 103
4.17 Anchor definitions maintained separately from links 106
4.18 HIPPO linkbase format 106
4.19 An example weighted inheritance hierarchy 109
4.20 Extract from the parent linkbase 109
4.21 New link weights inherited by child linkbase 110
4.22 An example tree definition 110
4.23 Using the adaptive server to modify a linkbase tree 112

5.1 The HIPPO+ distributed model 119
5.2 The Remote Procedure Call (RPC) 125
5.3 Compound Document Frameworks 127
5.4 The Open Management Architecture 128
5.5 The ORB in the a client-server transaction 129
5.6 Interface Definition Language (IDL) example 130
5.7 Static and dynamic invocation in the ORB 131
5.8 The CORBA Naming Service 132
5.9 Advertising services using trader objects 133
5.10 Federated traders 133
5.11 The HIPPO+ Architecture 135
5.12 The HIPPO+ Node Browser 136
5.13 The HIPPO+ buffer browsing tool 137
5.14 An example RPC interface 139
5.15 The RPC development cycle 139

5.16 Using *rpcbind* in the ONC RPC model 140
5.17 The Execution Manager 142
5.18 Some arguments automatically filled by HIPPO+ client 144
5.19 An extract from the HIPPO+ registry 145
5.20 Extending the computational model to incorporate additional services 147
5.21 The *Query Interface* browser 149
5.22 The HIPPO trader model 150
5.23 The HIPPO+ trader tool 151
5.24 An example HCH hierarchy 152
5.25 The Hypertext Component Hierarchy browser 153

6.1 A simple default link service 159
6.2 A more complex medical link service 159
6.3 Default viewing service 160
6.4 Multiple viewing services tailored to content formats 161
6.5 Alternative semantics of *get selection* 165
6.6 Stereotype model matches user categories to sets of service implementations 167
6.7 Each node has an associated service profile 171
6.8 A basic document object for a simple text node 172
6.9 New document object for graphical archive node 173
6.10 Document object for programming example 174
6.11 Weighted document objects 177
6.12 Adapting weighted service values 178
6.13 User stereotyping using multiple document objects 180

7.1 Merging multiple inheritance trees 200
7.2 An adaptive HIPPO+ implementation based on compound documents 203

B.1 Sun's Link Service 216
B.2 The SP3 architecture 217
B.3 The Microcosm model 219
B.4 The use of views in the Chimera system 220
B.5 The Dexter Reference Model 222
B.6 An example using the Dexter model 223
B.7 The MultiCard system 224
B.8 HyperDisco class hierarchies 226
B.9 HyTime modular architecture 232
B.10 Using Acrobat Exchange to view PDF documents 233
B.11 An example HTML definition 235
B.12 Netscape Navigator web browser 236

Abstract

The hypertext paradigm offers a powerful way of modelling complex knowledge structures. Information can be arranged into networks, and connected using hypertext links. This has led to the development of more open hypertext design, which allow hypertext services to be integrated seamlessly into the user's environment. Recent research has also seen the emergence of adaptive hypertext, which uses feedback from the user to modify objects in the hypertext. The research presented in this thesis describes the HIPPO hypertext model which combines many of the ideas in open hypertext research, with existing work on adaptive hypertext systems.

The idea of *fuzzy anchors* are introduced which allow authors to express the uncertainty and vagueness which is inherent in a hypertext anchor. Fuzzy anchors use partial truth values which allow authors to define a "degree of membership" for anchors. Anchors no longer have fixed, discrete boundaries, but have more in common with *contour lines* used in map design. These fuzzy anchors are used as the basis for an adaptive model, so that anchors can be modified in response to user actions. The HIPPO linking model introduces *linkbase trees* which combine link collections into inheritance hierarchies. These are used to construct reusable inheritance trees, which allow authors to reuse and build on existing link collections. An adaptive model is also presented to modify these linkbase hierarchies. Finally, the HIPPO system is re-implemented using a widely distributed architecture. This distributed model implements a hypertext system as a collection of lightweight, distributed services. The benefits of this distributed hypertext model are discussed, and an adaptive model is then suggested.

Overview

The field of hypertext has experienced renewed interest with the recent success of the World Wide Web. In particular, the hypertext community has seen the emergence of a more open approach to hypertext applications. Hypertext systems are no longer viewed as closed, monolithic applications, but as ubiquitous services which can be incorporated into existing environments. This move towards open hypertext has continued to develop and refine existing hypertext abstractions. Linking information can be maintained separately from the underlying node contents to provide a more loosely-coupled linking model. The role of the anchor in a hypertext has also been developed to provide a clean separation between the addressing and linking mechanisms.

Hypertext research has continued to develop in other directions with distributed and large-scale hypertext environments. Hypertext has been used to manage large numbers of users and to support collaborative working environments. The idea of adaptive hypertext has also emerged as a new research area. This incorporates information about the user into the hypertext model, which can be used to modify and change the hypertext.

This thesis introduces the HIPPO model which attempts to combine many of these areas into a single environment. The HIPPO system develops the anchoring and linking models which are widely used in existing open hypertext systems. This research incorporates adaptive modelling techniques into this new hypertext model, to provide a responsive and tailorable approach to open hypertext. The idea of a distributed hypertext environment is explored, and used to implement some of the ideas in this thesis. This distributed implementation attempts to provide a more open environment, which embraces ideas from open hypertext, adaptive systems and distributed software architectures.

Chapter 1 introduces the main ideas behind hypertext modelling. Basic hypertext objects are introduced – the *node, link* and *anchor*. The discussion shows how hypertext can be used to offer new opportunities for knowledge structuring, and how this has been realised in early hypertext applications. The chapter identifies many of the

common problems associated with the hypertext model, and shows how these have been addressed in the hypertext community.

Chapter 2 shows how hypertext research has developed since these early applications. The idea of open hypertext is introduced which attempts to move away from closed, monolithic hypertext applications towards more open link services. Some of the issues involved in open hypertext are explored, along with a summary of some important systems. Distributed and collaborative models of hypertext are discussed which address the problems of large-scale hypertext systems. Distributed hypertext systems must address additional problems such as concurrency control and support for collaborative working practices. Finally, the ideas of adaptive hypertext are introduced which attempt to incorporate some elements of the end-user into the hypertext model. Some of these existing approaches to adaptive hypertext are discussed along with some of the most influential adaptive systems.

Chapter 3 introduces the idea of *fuzzy anchors*. Fuzzy anchors attempt to address some of the limitations of existing anchoring models. These anchors incorporate the idea of fuzzy-set membership, to support a more expressive and less discrete notion of anchoring. The discussion shows how fuzzy anchors can be used in a hypertext, and explores some of the advantages that this offers. The research introduces an adaptive model which uses feedback from the user to modify fuzzy anchor definitions. Finally, a prototype implementation of the HIPPO system is discussed which implements the ideas in the chapter.

Chapter 4 develops the idea of the linkbase which has been used in many open hypertext systems. The linkbase is used to separate the hypertext linking information from the underlying node contents and offers significant advantages over conventional approaches which embed links in the nodes themselves. The discussion explores some of the limitations of the linkbase approach, and argues that the linkbase should be viewed as a first-class object in the hypertext model. *Linkbase trees* develop the idea of inheritance to support a new model of linkbases based on reuse and sharing. Linkbases can be expressed in terms of existing link collections, and can refine and tailor other linkbases. The idea of *weighted links* and *weighted inheritance trees* are introduced to support an adaptive environment, and these are combined with the fuzzy anchor model described in Chapter 3. The discussion develops the ideas to support a widely distributed model, which allows linkbases to

be located throughout the network domain. The HIPPO system implemented in Chapter 3 is also extended to support these ideas.

Chapter 5 builds on the ideas which are presented in earlier chapters, and applies these to a distributed hypertext environment. Chapters 3 and 4 developed the HIPPO anchoring and linking models to provide a more flexible and adaptive hypertext model. Anchor and link definitions can be manipulated and modified in response to user actions, and these new hypertext abstractions help to model more subtle and changing relationships. The HIPPO prototype which is discussed in these chapters is implemented as a single closed and monolithic application, which contrasts with many of the aims of open hypertext described in Chapter 2. This chapter attempts to re-implement many of the ideas presented in earlier chapters using a more open and distributed environment. This distributed HIPPO+ application identifies fine-grained hypertext operations and implements these as remote hypertext services. The functionality of the system, and the actual hypertext system itself is no longer implemented as a single application but is distributed throughout the network environment. The user invokes remote services to support common hypertext operations, and the HIPPO+ client provides a number of tools to help the user manage this new environment. The new prototype builds on some of the ideas developed in other distributed research fields, and applies some of the lessons from distributed software systems, to the hypertext community. The chapter identifies some of the advantages of implementing a distributed system over the existing HIPPO client, and suggests that a more loosely-coupled, widely distributed approach may be more appropriate for supporting the ideas introduced in this thesis.

Chapter 6 develops the idea of a distributed HIPPO+ system further, and proposes an adaptive framework for the HIPPO+ system. The chapter identifies some of the limitations of the current HIPPO+ implementation, and suggests some of the advantages of applying adaptive modelling techniques to a distributed hypertext system. This chapter introduces an adaptive model which allows the particular components and hypertext services which are presented by the user, to change and adapt to meet the precise needs of the user. Hypertext operations can be mapped on to different remote services and instances, so that the definitions of common hypertext operations changes over time. A number of alternative adaptive techniques are considered, before developing the concept of *document objects*. Document objects define mappings between abstract hypertext operations and actual remote instances, and associate these mappings with a particular node. Each node has a different

document object which allows the hypertext service profile to be defined at the document level. These document objects are developed further to include the notion of *weighted* services, which provide a basis for adaptive modelling. This model has not been implemented in the current HIPPO and HIPPO+ prototypes.

Chapter 7

This final chapter includes a complete summary of the thesis, and outlines the key ideas to emerge from the HIPPO research. The chapter identifies the main achievements of the thesis, and the key contributions of the research to the hypertext community. The discussion also identifies problems which were encountered during the course of the research, and suggests some alternative approaches that could be considered. The chapter closes with a discussion of future research directions, and areas that could be developed further in future work.

Appendix A includes a more detailed review of early hypertext applications.

Appendix B provides a detailed summary of existing open hypertext systems.

Appendix C The HIPPO anchor definition language.

Appendix D The HIPPO linkbase definition language.

Appendix E The HIPPO linkbase tree hierarchy language.

Appendix F includes a sample linkbase hierarchy, and shows how this can be specified using the HIPPO tree language.

Chapter 1

Introduction To Hypertext

The field of hypertext has generated much interest in recent years, most notably with the success of the World Wide Web. While many researchers approach the discipline from different perspectives, most agree with the hypertext ideas put forward by Ted Nelson in the 1960s [Nel93], as a way of expressing non-linear writings. Items of information can be arranged into complex structures, connected together by a series of links and relations. Hypertext frees authors from the linear restrictions of conventional media such as paper books, and allows alternative layouts and structures to be explored. Readers can explore a hypertext by choosing their own route through the information space – wandering off along links which grab their attention or examining some branches in more detail. A hypertext does not offer a single, authoritative reading path, but instead provides *many* alternative routes. A hypertext embodies many different paths and opinions, freeing the reader from the constraints of the printed book.

Researchers such as Engelbart [Eng84a, Eng95] and Nelson [Nel93, Nel95] did much to popularise the field of hypertext during the 1960s. However, it is widely acknowledged that the original ideas for hypertext can be attributed to the work of Vannevar Bush several decades earlier. Bush was Science Advisor to President Roosevelt, and was becoming increasingly concerned with the overwhelming volume of information made available to academics of the day. Bush looked to better ways of managing information, and published his findings in a paper titled *As We May Think* [Bus45]. In this paper he observed:

> *...our methods of transmitting and reviewing the results of research are generations old and by now are totally inadequate for their purpose.*

Bush went on to describe the *memex* system, which an individual could use to store all their books and records, using microfilm technology. Items could be called up at the press of a button and records could be easily copied or incorporated into an individual's private library. Furthermore, Bush suggested mechanisms for associa-

tive linking which allowed related items to be joined together using *trails*. Bush describes a scenario where a reader explores the information space, creating a trail of items of interest, and occasionally wandering off on related paths. Future browsers would explore these trails, benefiting from the previous readers' work, and perhaps adding extra information of their own. The actual realisation of the *memex* device relied on the technology of the day, using microfilm, projectors and photocells, yet it is the ideas of associative linking and re-use of trails that went on to inspire future generations of hypertext researchers.

When Bush's vision appeared in Atlantic Monthly in 1945 (although some of the ideas were developed some 10 years earlier), it provoked considerable discussion, yet we would have to wait almost 20 years before research would begin in the hypertext field. In 1962, Doug Engelbart began work on the Augment project, designed to "augment human intellectual capabilities". The system included support for group collaboration, and pioneered the use of multiple windows and mouse control — concepts that we take for granted in modern systems. The system was implemented as NLS (oN-Line System), before being marketed commercially as Augment. This early work pioneered many of todays modern computing concepts, and was to prove a major influence on the developing hypertext community. Also around this time, Ted Nelson was developing his Xanadu system — an ambitious vision of a unified literary environment, where all the world's literature is connected, or *intertwingled*, in one universal hypertext [Nel93]. These early ideas were among the most extravagant, and although Nelson's vision has never been fully implemented, work still continues on the Xanadu project to this day.

The study of hypertext theory is a very young discipline compared to many other areas of computer science research, yet it has a relatively rich history. Further systems continued to be developed, and some of these more influential projects are discussed in greater detail, later in the chapter. Early systems created complete working environments, addressed issues such as group collaboration and sharing and was designed for processing large volumes of information. In contrast, the next generation of hypertext applications were aimed at the single user, with more of a focus on graphics support and presentation issues. These included systems such as KMS, NoteCards, Intermedia, Guide and HyperCard, many of which were developed for workstations for personal use. Systems such as HyperCard did much to popularise hypertext, and make hypertext functionality available to the ordinary user.

The introduction of powerful graphics engines and workstations has had an important effect on the development of hypertext applications. In particular, the widespread use of the *point-and-click* paradigm, whereby users interact with multiple windows using a mouse, has had a profound impact on modern hypertext systems. The majority of contemporary hypertext applications require the user to

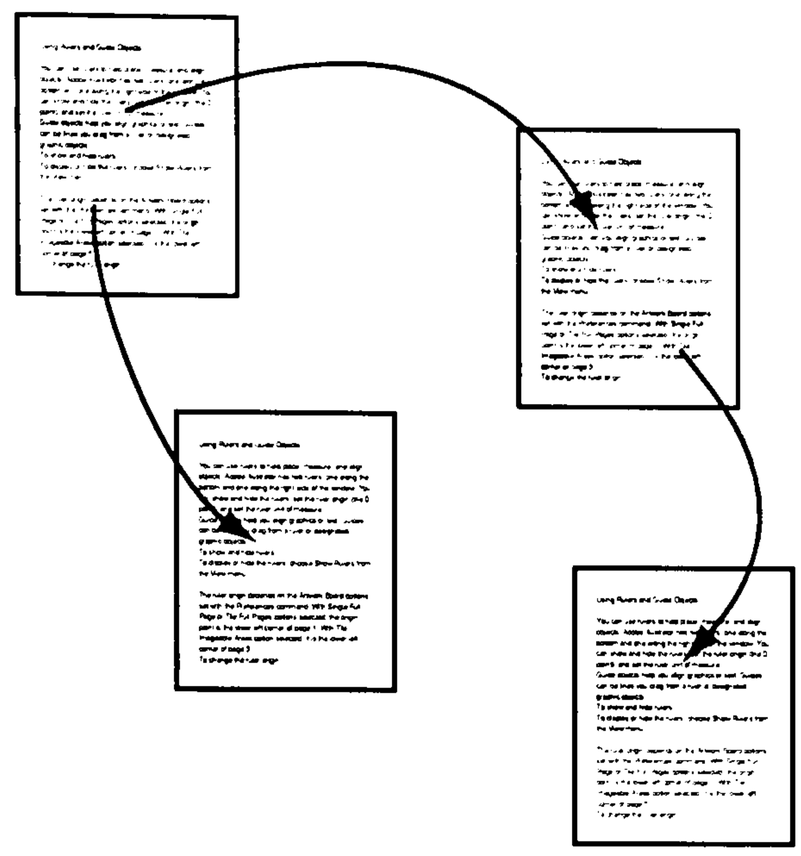


Figure 1.1: A simple hypertext example

select items of information by clicking on a region, which results in the traversal of a link. Recent developments in computer graphics and hardware technology have also given rise to the idea of *hypermedia*, where information is no longer limited to textual data but may contain multiple media types such as video and sound. This terminology has been the cause of much confusion, as many hyper*text* systems provide support for graphics and other media. The motivation behind hypertext is to explore alternative ways of structuring and managing complex information spaces, and while alternative media can raise interesting technology-specific issues, the underlying principles and goals remain the same. As such, the author does not believe the distinction to be significant, and the terms are used interchangeably in this thesis.

1.1 Defining Hypertext Terms

The underlying principle behind hypertext is very simple – chunks of information, connected together using links (figure 1.1). However, this simple concept has been interpreted in vastly different ways by researchers and academics. For this reason, it seems useful to define what exactly is meant by hypertext, and hypertext systems, and to introduce some accepted terminology. These terms are then evaluated in light of some of the more popular hypertext applications, to demonstrate the differences between competing applications.

1.1.1 The Node

The node is the fundamental building block of a hypertext, and provides a basis for structuring and shaping an information space. A hypertext node represents the

fragments of a text – the essential information chunks, or objects, which go to make up a piece of prose, a book, a model etc. The node aims to encapsulate the very *essence* of information, providing a tangible object which can be manipulated and arranged as a whole. The term *node* has become widely accepted in the hypertext world, although it has been known under many different names (eg. a *card* in NoteCards [HMT87, Hal87], Landow's *lexia* [Lan92], *writing spaces* in Storyspace [BJ87] etc). Similarly, the hypertext node has been realised in widely differing ways, applying different interpretations and constraints. This section highlights some of the more significant issues involved in supporting hypertext nodes.

Content Types

One of the most immediate differences between early applications and later generations of hypertext systems lies with the content restrictions placed upon the hypertext node. Traditionally, hypertext systems focussed on the application of hypertext ideas to conventional writing practice. Hypertext applications were concerned mainly with textual representation, so many early systems such as TEXTNET [Tri86, Tri83] and ZOG [AM84b, AM84a] were limited to simple textual node content. Early systems were limited by the hardware of the time, so as graphics capabilities developed, so hypertext systems were extended to allow multiple content types (HyperTIES [Shn87], KMS [AMY88], NoteCards [HMT87]). It is important to note that the introduction of diverse media types such as graphics and audio raise a number of issues and problems. Many of the accepted hypertext abstractions and presentation methods no longer apply to alternative content formats, and have led to the development of alternative hypermedia technologies. However, the majority of modern hypertext systems now offer support for additional node contents and media types.

Size Restrictions

Another issue involved in the definition of the hypertext node concerns the size of the actual node and the amount of content which is contained. Some systems such as KMS [AMY88] have fixed size nodes, which restricts the volume of information which can be stored. Other applications such as TEXTNET [Tri86], do not place any limit on node size, and allow readers to scroll through a node at their leisure. It is not clear what the optimum node size should be or exactly how the size of each node affects the hypertext experience. For example, Akscyn *et al* [AMY88] argue that KMS intentionally restricts node size to reduce the reliance on scrolling. Larger nodes can often be represented as several small nodes; indeed, in some cases a large node can suggest that the node has not captured the essence of a fragment. However, despite these arguments there are many cases when larger nodes are required, and it can be useful to include larger volumes of information. As such, most contem-

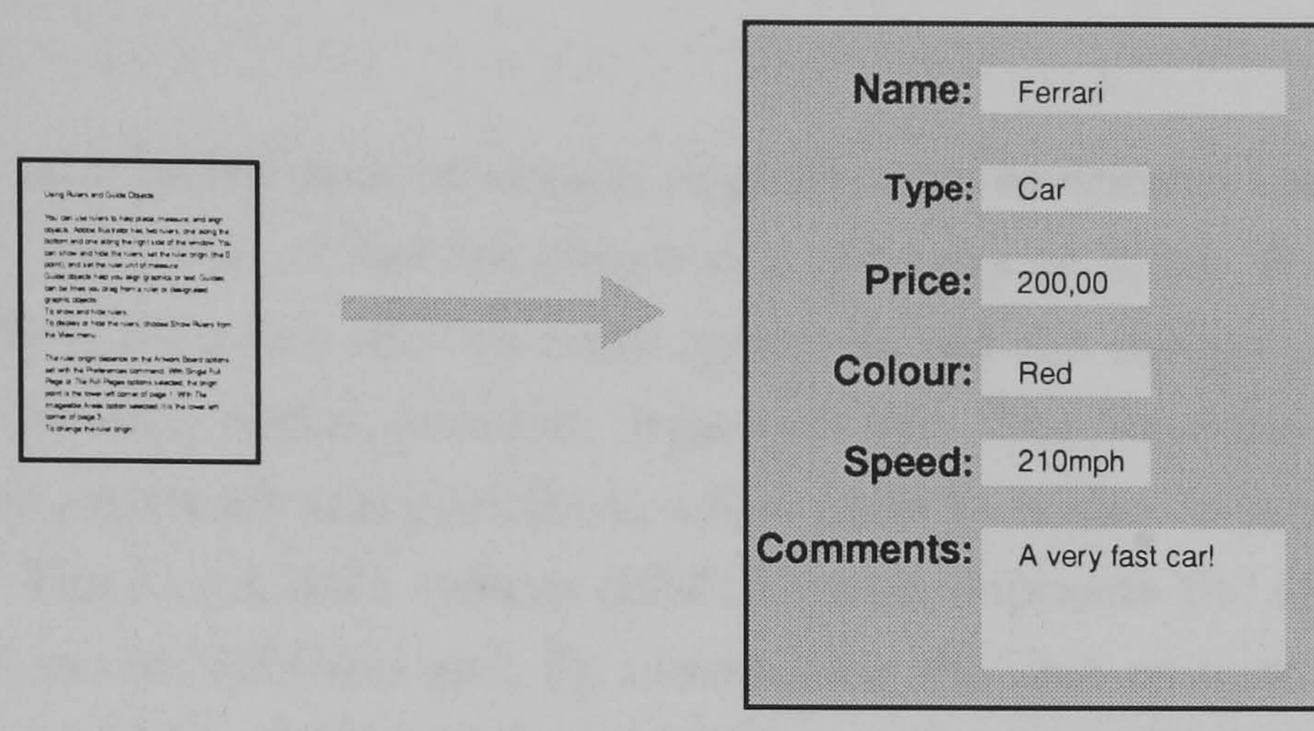


Figure 1.2: Frame-based nodes in Aquanet

porary hypertext applications adopt the more flexible policy of allowing arbitrary sized nodes in the hypertext. This problem of identifying nodes and fragmenting a body of information in to chunks is explored in more detail later in the chapter.

Node Structure

The previous discussion has focussed on the type and quantity of information encapsulated by hypertext nodes, but it is also important to consider the internal structure of the node. The approach taken by the majority of hypertext applications is to ignore the issue of content structure. Node information can appear in any form and can be arranged in any way that the author deems necessary. Other systems impose a more structured view of nodes; for example, Aquanet [MR92, MIRJ91] uses highly-structured, *frame-* based nodes, which consist of fixed fields (figure 1.2). Similarly, the VIKI system [MS95, MIC94] borrows from the Aquanet approach, adopting a more relaxed, semi-structured node definition. The World Wide Web [WWWa] uses the HTML markup-language to define node contents, by identifying generic blocks, paragraphs, lists etc. This form of *logical* structuring is largely used to simplify the display of node contents and interchange between heterogeneous platforms, but can be used as the basis for further node processing.

Again, it is unclear which approach to node structuring is the most useful – highly structured objects or more informal definitions. Akscyn *et al* discuss how unstructured approaches can be used to support more formal definitions of node content. Users of the KMS system would repeatedly use familiar layouts and practices when defining nodes, so that a form of *de-facto* structure would emerge (eg. a title followed by the main text body, then a set of links which can be selected). A familiar layout helps the reader identify common information and to navigate around the contents of a node. Also, highly-structured nodes allow the hypertext system to perform additional processing of node contents, to help users in their reading tasks.

Node Typing

In addition to an internal node structure, hypertext environments can introduce the idea of *typing*, such that nodes are assigned a category or label. For example, the gIBIS tool [CB89] uses this idea of node typing to aid the design process, so that users can define *issue* nodes, *positions, arguments* etc. The Neptune system [DS86] allows users to attach arbitrary attribute-value pairs to nodes, to support a form of node typing. The NoteCards system [HMT87] also supports the idea of different node types (*Browsers, Fileboxes* etc). By introducing the idea of node typing, hypertext tools can provide additional functionality to the user. Nodes can be viewed and displayed according to types, unwanted nodes can be filtered out, node types can be used to enforce particular behaviours or to constrain the available operations etc. Many systems do not support a node typing system, choosing instead to utilise a single node type (for example, the KMS system unifies all nodes into a single *frame* type). A single node type can greatly simplify the user's interaction with the hypertext, and can simplify the implementation of the hypertext environment (eg. reduced command set, no type-checking system etc).

1.1.2 The Link

The motivation behind hypertext is to build a non-linear, branching information space by arranging nodes into more expressive structures. Objects and pieces of text do not have to follow sequentially, but can form branching networks which are connected together using hypertext *links*. These links encapsulate the relationships between each of the items, and introduce a richer semantics into the hypertext. Researchers have developed the simple notion of the *hyperlink* along different lines, and this section explores some of the more significant issues surrounding the implementation of hypertext linking.

Link Direction

A hyperlink denotes a relationship between two items of information and is most commonly associated with some form of traversal or navigation. A user typically selects a link to explore, and is then presented with the destination item which resides at the other end of the link. The idea of *navigation* associated with a link raises the issue of link *direction*. Can a user traverse the link in both directions, or must the user navigate the structure in a particular direction? The HyperCard [App87] and Intermedia systems [YHMD88] place no limitation on direction, and allow users to traverse the link from either end. Conversely, other systems such as NoteCards [HMT87] and HyperTIES [Shn87] adopt the idea of directional links. The author specifies a direction for each link, and this determines the direction that the user

must navigate. In some cases, hypertext environments support the idea of a *history* or *backtracking* mechanism which allows readers to return to previously visited nodes (eg. WWW browsers [Net, MSE]).

However, the notion of *directionality* becomes more difficult when we explore the idea of a hypertext link in more detail. A link represents some relationship or dependency between two objects (a link may support more than two end-points, but this is discussed later in the section). A hypertext link has an actual meaning, some *semantics*, over and above a simple navigational role. In this sense, the idea of a link direction can be confusing, and may mean many different things. In the discussion of the Dexter model [GT94], Grønbæk *et al* identify several interpretations of link directionality:

- *Semantic direction*
  For example, item A *contradicts* B. This relationship has a single direction; the reverse direction is not always valid.

- *Creation direction*
  This depends on the order in which the author created the link.

- *Traversal direction*
  This is the conventional view of directionality, in which the hypertext application determines which direction the user can traverse.

Complex Structures

The concept of hypertext linking discussed so far, has focussed on the idea of a simple connection between two nodes. This allows nodes to be arranged into some form of interconnected network, which allows readers to explore branches and choose their own route through the network (eg. HyperTIES [Shn87], Xanadu [Nel93, Nel95]). However, this simple network can sometimes prove inadequate to express more complex relationships, and many hypertext researchers have explored alternative structuring methods.

A form of hypertext structure which is commonly seen in hypertext systems is the *hierarchy* (eg. Emacs INFO [Sta], ZOG [AM84a]), which allows nodes to be arranged into layers of *parent-child* relationships. This provides a means of supporting increasing levels of abstraction, and has proved to be a natural structuring paradigm for the hypertext user. Users can identify a node of interest, and explore the children of this node for more information. The hierarchy is simple to support in most hypertext applications, although it does have some limitations. The main disadvantage of a hierarchy is that the structure is a function of the criteria used to construct it – what may prove to be a useful hierarchy for some applications may be ill-suited to other domains. For this reason, many modern systems

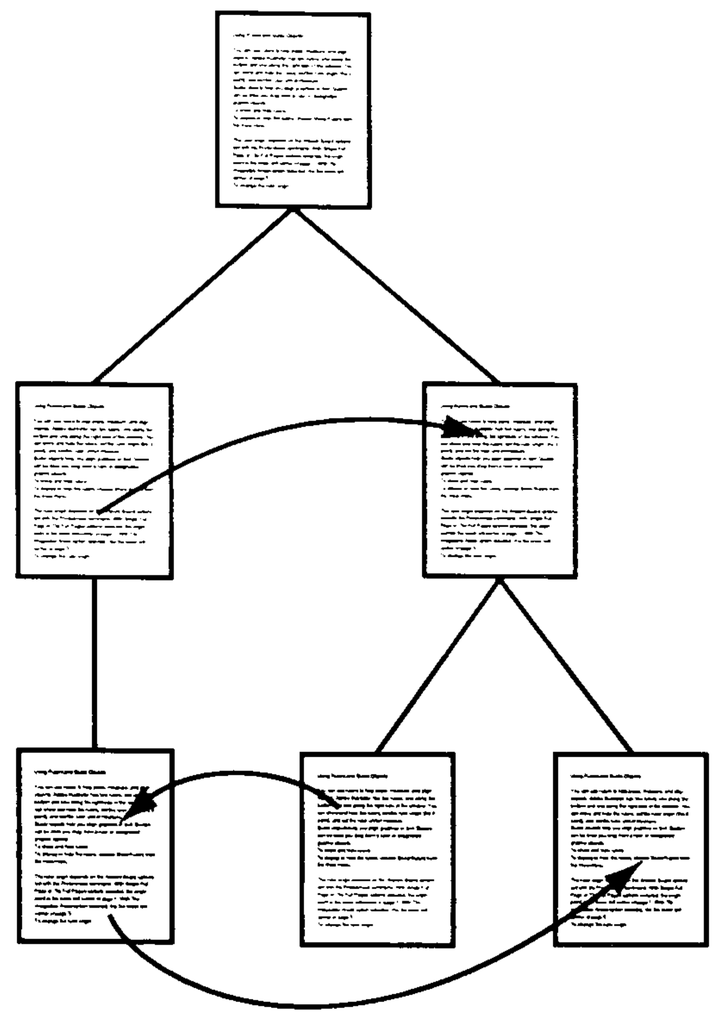


Figure 1.3: Hierarchical links and cross-hierarchy links

support both hierarchical structures, and cross-hierarchy links (TEXTNET [Tri83], Intermedia [YHMD88, HKRC92, Mey86], NLS [Eng84a], WE [SWF87], NoteCards [HMT87]) (figure 1.3).

Linking mechanisms have been developed in other ways to augment the hypertext environment. Multi-way, *n*-ary links are supported in many applications, so that a single anchor can act as the source for several branches. Similarly, applications such as Microcosm [DHHH92] support a *generic* link which allow multiple source anchors to map on a single destination. It can also be useful to allow links to act as the target for other links; in this way, links can be connected together to form more flexible structures [Tri83].

However, despite the flexibility of the linking schemes described previously, many researchers have called for better support for complex structures. Parunak [Par91] explored ways of supporting aggregations using set-based linking, and Marshall *et al* introduced the notion of *spatial* hypertext to express relations such as collections. Halasz [Hal87] identified limitations with conventional linking techniques and called for improved support for composite objects. Authors should be able to structure nodes *within* nodes, to create a *truly* aggregate object which can be treated as a first-class entity in the system. While support for composite structures has improved, many systems still fail to fully address the issues involved in supporting complex structures.

Link Typing

The idea of an object typing system was introduced in the previous discussion of nodes, and has also been applied to the area of hypertext links. Link typing allows the author to express additional properties when defining relations and node connections. A link type can represent the nature of the link, and provide additional information to the user. Also, a typing scheme allows the application to provide additional processing and functionality (link filters etc). Early attempts at link typing provided informal support through user-defined labels [HMT87, DS86]; users could attach their own labels and attributes to links, and these would then be interpreted in the appropriate way. Other systems such as TEXTNET [Tri83], gIBIS [CB89] and PHIDIAS [MBD 90] borrowed ideas from semantic networks, and introduced more formal typing schemes. WE [SWF87] and ABC [SS91] introduced typed *structures* (graphs, hierarchies, paths) which support structure-specific behaviour.

Static vs Dynamic Links

Most approaches to hypertext linking have adopted a very static, stable view of a hypertext environment. Authors construct links between fixed points, which remain unchanged for the lifetime of the hypertext. While this can be useful, it is clear that hypertext, as a knowledge structuring paradigm is widely applicable to more dynamic situations. Real-world applications demand more flexible linking mechanisms to handle constantly-changing information. Halasz [Hal87] called for increased support for *virtual structures* and hypertext researchers have attempted to address these requirements by introducing the idea of *dynamic* links.

Dynamic links can be realised in many different forms, by applying dynamic computations to different areas of the linking cycle. The ZOG [AM84b] system was an early application which introduced a simple form of dynamic link, by automatically constructing a link every time a new node was visited (eg. a link back to the previous node). Intermedia [YHMD88] introduced *hot* and *warm* links which allow the contents of a link endpoint to be dynamically updated on demand. Other systems attached scripts to hyperlinks, which would be invoked each time the user traversed a link (KMS [AMY88], Notecards [HMT87], HyperCard [App87], Intermedia [YHMD88] etc). These would typically be implemented by providing some limited scripting language, to provide access to hypertext operations – examples of this are NoteCard's scripting language based around *lisp*, and the *HyperTalk* language used in HyperCard. This approach to dynamic linking, by attaching computational components to links provides a very powerful extension to a hypertext environment. Link destinations can be evaluated at traversal-time, a link can execute queries to locate objects, and destination node contents can be constructed dynamically on request.

Brown [Bro88] identifies other problems with static hypertext linking mechanisms in large hypertext environments. Links can be difficult to maintain in a changing environment and can soon appear out of date. It is often unreasonable to expect authors to provide all links for a user; indeed, it is often simply not possible to provide a set of links which are suitable for *all* users. Dynamic links allow queries and link destinations to be evaluated on a *per-user* basis, tailored to the needs of each reader. Also, dynamic links are largely transient so do not interfere with other users, unlike permanent, static approaches to linking.

However, dynamic links raise many problems and difficulties over conventional linking schemes. Dynamic links are often less accurate, and it is difficult to construct a query to locate the precise objects of interest. Dynamic links can introduce side-effects into a hypertext environment, so that users can traverse links, unaware that they are executing some script. Should a user be aware of a dynamic link? How does a user save a dynamically constructed node (for example, a WWW bookmark which refers to the results of a CGI script [WWWa]) – do they save the node contents or the query which generated the node? Much of the work on dynamic links ties in with the development of more open, computational hypertext models, and these are discussed in more detail in the following chapter.

The previous sections have explored some of the common issues arising from hypertext linking models; however the issue of hyperlinks raises all kinds of additional problems. Are links stored as separate components [YHMD88] or are they simply data values associated with nodes [AMY88]? Are links modelled as formal relations or should a hypertext environment adopt a more informal, implicit approach [MS95]? This raises the question of what exactly a link represents – is a link a purely navigational abstraction or does the link capture some semantics about a relationship, and if so, what are these semantics? Marshall *et al* separate links into *permissive, emergent, descriptive* and *prescriptive* categories based on the constraints of the author [MS95]. Similarly, DeRose [DeR89] offers a taxonomy of link types used in hypertext applications.

Many researchers have attempted to identify different link types and to classify hypertext relations, yet the diverse range of approaches suggests the role of the hypertext link is not entirely clear. A hypertext relation can represent different semantics depending on the domain and context of the hypertext. Different users may interpret relationships and collections in different ways depending on their knowledge and goals. Indeed, Bolter argues that the separation of nodes and links is somewhat artificial, and that nodes and links can exhibit a form of *oscillation*. Links can take on node-like properties, just as nodes can exhibit properties similar to those of links [Bol91].

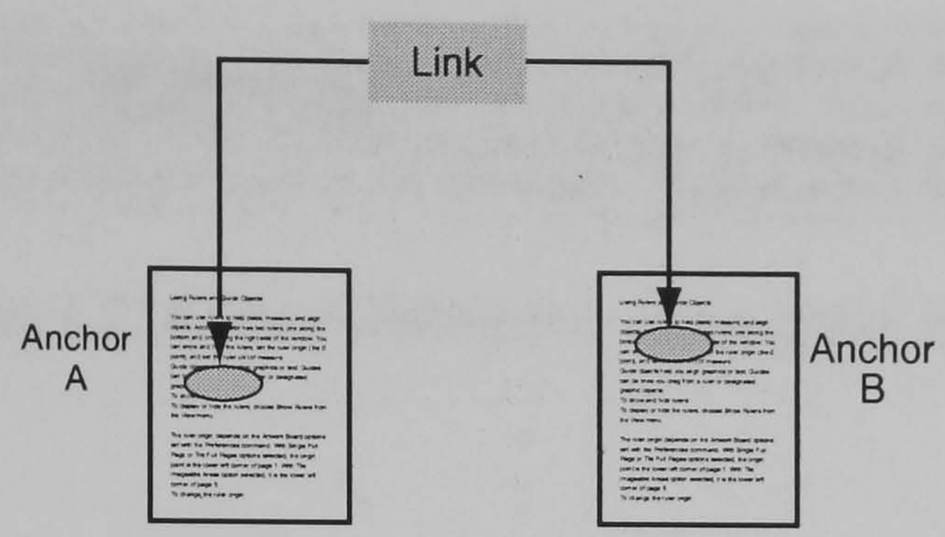


Figure 1.4: The anchor provides an endpoint for the hypertext link

1.1.3 The Anchor

The previous discussion has described the hypertext paradigm as a model based on nodes and links. Hypertext nodes represent the atomic fragments of a text, which are structured and arranged using hypertext links. While this simple model is sufficient, it does have many limitations. Most notably, this model ignores the problem of addressing mechanisms, and ways in which links can be connected to the nodes themselves. How does a link connect to a node? Can a link connect to objects *inside* the node? How are users made aware of links? These problems stem from the idea of *anchoring*, and this section includes a brief discussion of some approaches to hypertext anchors.

The concept of hypertext anchors is used to unify relations with the actual hypertext objects themselves. While the link expresses the relationship between two components, it is the anchor which is responsible for addressing the node content at each end (figure 1.4). The idea of the anchor is simple, yet hypertext developers have taken very different approaches to anchoring.

Anchor Granularity

The primary role of the anchor is to support addressing mechanisms – to provide a means of identifying an object in the hypertext. Many systems have adopted a very simple view of this, by restricting anchors to the granularity of the nodes themselves (WE [SWF87], TEXTNET [Tri83], ZOG [AM84a]. Links can only connect entire nodes – pages, documents, paragraphs – but are unable to address objects at the *sub*-node level. Other systems such as NoteCards [HMT87] and KMS [AMY88] provide additional support for anchoring, by allowing links to originate from *within* the node. However, the destination of each link must still resolve to an entire node; this restriction has been removed in other applications (HyperTIES [Shn87], Neptune [DS86], World Wide Web [WWWa]). Akscyn *et al* [AMY88] argue that it is sufficient to provide addressing at the level of the node, on the assumption that the node represents an atomic concept, and is a sufficient logical unit.

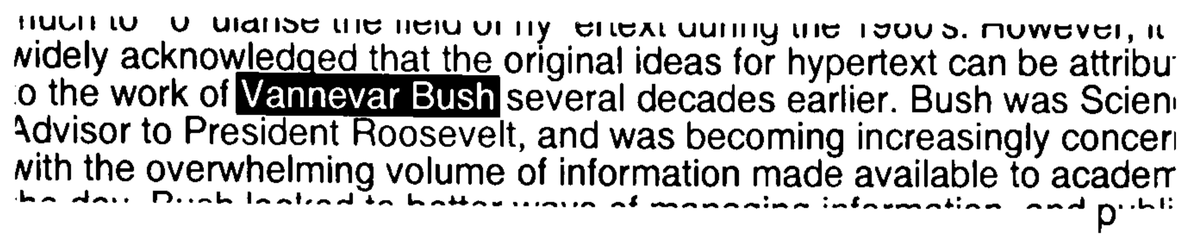


Figure 1.5: Anchor definitions using spans of text

Addressing

Another issue surrounding the problem of anchoring is that of content addressing – how can a link identify the source and destination anchors, and with what precision? Small-scale systems can identify nodes with relative ease, perhaps by assigning a unique identifier to each object, or by using the filename corresponding to the content. This problem is magnified in larger environments – for example distributed systems – when nodes can no longer be allocated unique references, and resources can move location. The World Wide Web [WWWa] in particular is a good example of a large-scale hypertext system, which has developed naming schemes and addressing mechanisms to help solve this problem [WWWb]. Hypertext applications also differ in the precision that they offer the author, to address content within nodes. The Augment environment [Eng84a] in particular provides a very flexible addressing model ranging from section numbers and identifiers to relative addresses and regular expression matching. Similarly, some systems allow anchors to include a span of text [YHMD88, Nel95] (figure 1.5) while others limit anchors to a single point [HMT87]. The anchoring model is developed further in Chapter 3 which introduces a more flexible notion of addressing.

Anchor Representations

Not only do applications support vastly different addressing models, they also adopt diverse representations (both *internally* and *externally*) for anchors. While the term *anchor* is widely used, and is used consistently in some abstract sense, the majority of applications do not implement the anchor as a primary, first-class object. Anchors are often implemented as simple data values, which are then associated with links. This is an area which has been developed in later systems, especially in work emerging from open hypertext research [Kac90, GT94]. This move towards anchors as first-class hypertext objects is discussed in more detail in the following chapter, and the anchor is further developed in the work from chapter 3.

Designers must also address the problem of how links (and in particular, anchors) are represented to the user – how are the users made aware of the existence of a link, and how do they select the anchor? Some applications highlight spans of text [YHMD88], while others provide a link marker to denote a link endpoint. Other approaches provide explicit interface objects such as buttons to denote anchors while

others may adopt more subtle techniques such as cursor changes, colour codes etc. Other issues involved in the representation of anchors can cause problems – for example, how are overlapping anchors represented to the user? How are separately-stored anchors maintained when authors edit the underlying content? Chapter 3 also discusses some of the issues involved in representing anchors.

The previous sections have discussed some of the defining features of the hypertext discipline, and explored ways in which researchers have developed this model further. So far, hypertext has been loosely described as a set of node and link abstractions, which can support the task of knowledge structuring. However, this section attempts to explore the idea of hypertext in more detail, and identify some of the advantages that hypertext can offer to the author and reader.

1.2 Advantages Of The Hypertext Approach

The conventional means of presenting and structuring information has traditionally involved the medium of printed text. This linear representation has been widely used for centuries, and has been used as a vehicle for diverse domains – scientific texts, prose, popular culture etc. However, it is questionable whether this linear representation is always the ideal method for knowledge transfer.

The sequentiality of text and the printed book arise from the sequentiality of language. However, there is nothing which says that the presentation *should* be sequential [Nel93]. Our thoughts and ideas are far from linear; they consist of many complex, deeply connected thoughts. Ideas emerge from other ideas, opinions are re-evaluated in light of others; particular ideas gain more importance and reinforce others. No single idea appears first, and there is no single, linear order to our thoughts. Conventional methods of presentation force the author to arrange these ideas into a single, universal order which often loses much of this richness and can be a destructive practice. A well-constructed linear text may provide a clear direction and can integrate many different threads. However, linear representations often fail to express the many complex connections and dependencies between ideas, and do not acknowledge the subtle processes which gave rise to these threads.

A hypertext model allows a knowledge space to be constructed using a more expressive, non-linear structure. Ideas lead on to other ideas, which give rise to branches in a story and alternative paths through a text. This is not to say that this is the way the human mind manipulates information, but it can be a useful and effective means of expressing structured information. Human thought seems to be a parallel, iterative activity and hypertext models can help to represent these ideas more easily than conventional practices.

One could argue that hypertext is nothing new, and that we have been using these ideas of branching, connected texts for centuries, in the printed book. Conklin [Con87] points out that literary work often has non-linear elements; readers wander off to locate references and qualify terms, some sections are skipped and returned to at a later date and ideas are balanced against previous work. Traditional writing techniques have long been accepted for signalling branches and changes in the flow of thought – footnotes refer the reader to items of interest, citations connect isolated texts to other relevant sources. Authors often use sidebars to offer additional explanations, and for providing detailed indexing systems to help the reader navigate a course through a book. These are all common practice in modern writings, and do exhibit many of the characteristics of hypertext. The study of hypertext aims to explore these informal, ambiguous techniques, and to provide a more formal, richer environment for non-linear writing. Hypertext encourages authors to think about the relationships and connections between ideas. Also, hypertext tools support a new form of knowledge elicitation, in which ideas and structures gradually evolve into a complete text.

Hypertext offers many advantages to the author, but also provides new opportunities for the reader. The conventional role of the reader is that of the *passive observer* – the author prescribes a strict reading order and offers a single interpretation for the reader. This is not to say that readers do not engage in the subject matter or form their own opinions, but that the reader is largely seen as a passenger in the reading experience.

The hypertext approach to knowledge structuring views the reader as a more interactive component; readers decide their own route through an information space and which branches they explore. Readers become their own author, discovering new areas and ideas and ignoring others. The reader dictates his or her own experience as the reading process becomes much more of a collaborative effort. Michalak *et al* [MC93] discuss the new role of the reader as receiver of information, and as maker of meaning. Indeed, hypertext encourages a much more cooperative model of writing – readers can add additional links to a text which can be shared with other readers. Experienced readers might add annotations to certain nodes, or join them with other hypertexts and related sources. Hypertext reduces the replication of redundant information and promotes sharing of common resources. Conklin emphasises that hypertext is not simply a collection of nodes and links, and compares this to describing a meal by simply listing the ingredients [Con87]. Hypertext promotes a dynamic and interactive form of writing in which the knowledge space grows and evolves to reflect the experiences of the readers.

Just as hypertext offers a new writing medium, so authors must adopt and refine new writing techniques. Moulthrop argues that new forms of hypertext rhetoric need to be developed to support the particular needs of hypertext [Mou91]. Hyper-

text does not aim to replace the book, and direct comparisons with printed media will always be unproductive. Hypertext no longer has the reassurance of stability and certainty which traditional books provide [BJ87]; stories change with each reading and readers experience a text from different perspectives, reusing ideas and expressing them in different contexts. A hypertext has no *"primary axis of organisation"* [Lan92] so allows each reader to choose, and change the focus of the document. A hypertext is an *"infinitely de-centerable and re-centerable system"* [Lan92] which can be moulded and twisted as the reader sees fit.

Many researchers have explored the role of hypertext in modern writing, and developed ideas of hypertext literary theory [Lan92, BJ87, MC93, Mou91, Mou92]. The effects of a hypertext model on conventional authoring practices are complex and far from clear, however this section has attempted to highlight some of the most notable contributions. Appendix A discusses some of the most influential systems which have emerged from the hypertext community. This chapter closes with a brief exploration of some of the key problems that researchers have encountered regarding the hypertext paradigm and current implementations.

1.3 Problems With Early Hypertext Applications

This chapter introduced the fundamental ideas behind the hypertext discipline, and explored some of the more influential systems that have emerged. Hypertext systems have been applied to diverse problem domains and many have been developed into commercial ventures. However, many people consider that hypertext has failed to deliver its promises, and to achieve its early potential [Ras87]. Researchers have identified a number of problems with the conventional hypertext model, which have prevented the widespread adoption of hypertext techniques. The remainder of this chapter examines some of the more general criticisms, and Chapter 2 discusses some of the problems which are specific to open hypertext systems.

1.3.1 Navigation – the *disorientation* problem

This is perhaps the most common complaint levelled at the hypertext community; the problem of navigating through a hypertext. System designers developed applications to support hypertext structuring – non-linear, branching texts – and provided methods to browse these networks. However, it was soon discovered that users had great difficulty exploring a hypertext and forming a meaningful cognitive model of the overall structure [Nie90]. Users would explore branches in the

hypertext, but find it difficult to decide how these fitted into the main structure. Readers would find themselves visiting nodes which they had already encountered, wandering aimlessly through a hypertext without any meaningful direction. While hypertext offers the reader multiple paths, users often found this freedom overwhelming, and missed the certainty and structural rigidity of conventional texts.

Early attempts to address this *disorientation* problem focused on the construction of maps to provide an overview of the entire hypertext [HMT87]. In this way, it was hoped that users would be able to find their position in the hypertext easily, and achieve some sense of context in the larger hypertext. Hypertext maps have proved very useful in reducing navigational problems, and have been widely supported in many hypertext applications ([HMT87, DS86, YHMD88]. However, the support for hypertext maps raises many other complex issues. For example, hypertext networks can typically include many nodes and links, which proves impractical to present in the form of a map. Similarly, the construction of maps does not scale well to larger numbers of nodes and is computationally intensive. A complex map can overwhelm the reader with a tangled mess of meaningless links. What is the best way to lay out a complex map? Should some detail be filtered out from the map? Does the user need a map of the entire hypertext? Researchers have explored different methods of displaying and presenting overviews of a hypertext in an attempt to make navigational mapping more feasible for large-scale networks [Fur86, Fei88, Noi93, TD92, ZR97]. Other systems offer filtering techniques, and support for multiple maps at different levels of detail [YHMD88].

However, maps and overviews are not always appropriate for many hypertext applications. Maps cannot easily embrace dynamic information or hypertexts with rapidly changing topologies. Shneiderman [Shn87] suggests that much of the disorientation problem originates from poor authoring styles and design. Similarly, Moulthrop [Mou92] calls for a rhetoric of hypertext writing, and further evaluation of hypertext literary theory. Brown [Bro88] goes further to suggest that readers should *not* be made aware of the hypertext structure, any more than users are made aware of the sub-structure of other information retrieval tools.

Other approaches to the problem of navigation include the idea of *breadcrumbs* [Ber88] to inform the user about which nodes have already been visited (eg. WWW [WWWa]). The *guided tours* [MI89] in NoteCards and Zellweger's *scripted documents* [Zel89] have proved useful in addressing these navigation problems. Nielsen [Nie90] suggests other techniques: overviews, support for backtracking, timestamps etc. The idea of link typing has been introduced into many systems (section 1.1.2), while others offer the reader a summary of the node contents before they traverse the link [Shn87, SCG89]. Akscyn *et al* [AMY88] focus on providing a quick response to link traversal, so that the overhead associated with navigation is minimal. Halasz [Hal87] calls for better support for queries and searches to help the user locate items

of interest. This view of queries and dynamic computation, to help users explore a hypertext, has been developed by many systems and forms the basis for many of the open hypertext systems described in chapter 2.

Indeed, the problem of *resource discovery* – locating items of interest – is proving to be a difficult problem for hypertext developers. In particular, users of the World Wide Web [WWWa] find this to be an acute problem as they become overwhelmed by the sheer volume of information at their disposal. Search engines and *spiders* which attempt to index the information space have gained popularity with many users [Alt]. Although these tools can be useful, they have little idea of context so cannot deliver accurate, "intelligent" results. Other approaches such as *Yahoo!* [Yah] provide a more structured indexing service, in which nodes and web pages are filed manually. This can provide a more accurate, higher-quality service but suffers from scalability problems as the information space continues to grow.

Some systems have focussed on the problems of changing contexts as one of the main causes of disorientation. An interested reader might follow a link, which results in the presentation of a new node; the previous node (and context) is completely removed, and the user can experience some confusion. Traditional media allow readers to decide when (and if) they follow a link, and it is often accepted what form this will take (eg. look in the index, move on several pages etc). Guide [Bro89, Bro92] addresses this problem by introducing the idea of *replacement* buttons – when the user selects a link, the button is replaced with the node contents, rather than relocating to a new node. Modern environments which support multiple overlapping windows can also help to alleviate this problem, by presenting several nodes concurrently.

It is not clear that a solution based around maps, which encourages a spatial, geographical view of hypertext, is entirely beneficial. Hypertext encourages the author to concentrate on the relationships and dependencies between items of information. Hyperlinks are not simply connections to be traversed, but represent rich semantics about a network. Kilov [Kil94] draws an analogy between simple hypertext links and the infamous *goto* statements which plagued software developers. Just as programmers moved away from these "spaghetti-like" dependencies [Dij68], so too should the hypertext developer. De Young [You90] also identifies some of the shortcomings of conventional hypertext implementations, and calls for richer linking models. Conklin [Con87] argues that the experience of disorientation is in some ways inherent in the hypertext paradigm – there is no natural topology for an information space so until one is familiar with the document, then they are by definition, *disorientated*. Modern systems continue to address this navigational problem, but this still continues to be an issue in the hypertext community.

1.3.2 Premature Segmentation

The task of building a hypertext is a very difficult process, which can place heavy demands on both the author and reader. Complex ideas need to be distilled into simple, self-enclosed nodes, and arranged into meaningful structures. This rigid method of authoring does not reflect the true nature of writing, and provides a somewhat artificial environment for authors and readers. Ideas are not such tangible objects; they emerge and develop as the author builds the hypertext. The role of a node in a hypertext is not immediately obvious, and its place only becomes apparent as the hypertext structure takes shape.

Most hypertext systems require an author to create nodes in their entirety, then to place them in the hypertext. This can often force the user to make difficult decisions about the layout and structure of the hypertext, before the true nature of the information is really known. Authors must decide which content is included in a node, and which is left out. Halasz [Hal87] noted that users would adopt various strategies in order to delay this premature segmentation and filing of information by placing several ideas in a single node before dividing them up into smaller nodes at a later date. Sketch cards were used to arrange nodes into piles and informal structures, before finally creating hypertext links. Conklin [Con87] explores this in more detail, and the VIKI system [MS95] has developed the idea of *spatial hypertext* to capture this more informal, emergent view of hypertext. Halasz [Hal87] also calls for better support for *virtual structures* to support dynamic, changing information.

1.3.3 Maintaining Hypertext Structures

The hypertext paradigm supports the development of more expressive structures, to represent the many complex relationships in an information space. However, this can make hypertexts very difficult to maintain and support over the lifetime of the network. Some links may become obsolete and irrelevant, others may *break* as nodes are moved or deleted. Users may update nodes in the hypertext, which requires all related links to be modified. Similarly, changes in some areas of the network can affect other areas of the hypertext, as the changes permeate through the hypertext.

These problems become more serious in large-scale hypertexts which support multiple users and many nodes and links. The World Wide Web is in a constant state of flux, containing many millions of nodes – all constantly changing and moving location. Users frequently encounter links which reference out of date pages or pages which no longer exist. This is very problematic for hypertext designers, especially in distributed environments which have no single point of control. The WWW community has seen the emergence of many tools to test the integrity of links, and help with the administration of hypertext sites. Other research has explored the problem of versioning nodes and maintaining multiple versions of nodes over the

lifetime of a hypertext [AMY88, WL92]. Some of these issues are developed further in Chapter 2, in the discussion of distributed and collaborative hypertext.

1.3.4 Cognitive Overhead

The term *cognitive overhead* was introduced by Conklin [Con87] which identified the additional mental demands made on authors, in creating and keeping track of hypertext links. An author may think of some new idea which they wish to capture – they then have to interrupt their current task to create a new node, then link this new node into the hypertext. What should the link represent? Where should the link originate from? Should the link be labelled to suggest the contents of the node, or to indicate the relationship to the new node? Beyond this, the author needs to consider how the reader will be affected by this new branch in the hypertext; is the link meaningful in the current context? Perhaps the node would be better placed elsewhere in the hypertext? Also, the author cannot know how the reader arrived at a node and which nodes they have already visited. This can make it difficult to produce a cohesive hypertext which flows smoothly between nodes (although Moulthrop argues that this is not a disadvantage [Mou91]).

Similarly, this mental overhead is experienced by the reader of the hypertext. Just as a hypertext offers the reader new freedom and choices, so they are forced to make choices at every branch, deciding which links to explore and which to ignore. How does the reader make these decisions? Has the author provided enough information to choose a link? It is also difficult for authors to provide links which are appropriate to all readers, with different levels of expertise and interests [NK89]. These additional tasks all place extra demands on the reader, and can prove distracting, in some cases overwhelming, for both author and reader alike.

This cognitive problem has much in common with the earlier disorientation problems which were discussed, and some of those solutions can be applied to this area. Hypertext systems can support filtering of nodes and links [SCG89] to reduce the amount of information available to the reader, and some systems incorporate adaptive, intelligent elements to tailor the hypertext to the user (see Chapter 2). Some applications allow users to maintain their own sets of links [GB80, DHHH92, YHMD88] and personal items of interest [Mos, Net] – this is an area which becomes more important in open hypertext research. Other systems support the idea of *activity spaces* which provide task-specific operations [SWF87, HHL 92] to help the authors and readers construct hypertexts. The cost of introducing hypertext into existing legacy domains is also significant and has been addressed by some systems using automatic linking [DHHH92] and information retrieval techniques [FC91, Gol97].

Chapter 2

Developing The Hypertext Model

The previous chapter introduced the ideas behind hypertext, and identified some of the more influential systems that have been developed by researchers in the field. A number of issues relating to node, link and anchor implementations were explored, together with ways in which different system designers have approached this. Users and researchers have identified a number of problems with early implementations of hypertext systems, and these were explored in more detail. This chapter aims to follow on from this, by discussing ways in which the hypertext paradigm has been developed. The discussion centres around those areas which have influenced the work in this thesis, and have provided the basis for much of the work. In particular, the chapter covers *open hypertext systems (OHS), distributed and collaborative systems* and *adaptive hypertext*. These ideas are then incorporated into the remainder of the thesis, which describes various aspects of the HIPPO system.

2.1 Open Hypertext

The previous discussion of hypertext focussed on a number of significant hypertext applications, and explored each of these in some detail. These systems made many contributions to the hypertext discipline, and helped shape the view of hypertext. Researchers would develop their particular system by providing additional functionality and support for different hypertext abstractions. System designers might develop applications in response to user response, or borrow ideas from other hypertext applications. Users could then choose the hypertext system which best-suited their needs and provided the appropriate functionality for the task in hand.

However, this form of development is a very closed and insular approach to hypertext design. Hypertext applications are developed independently from others, and provide a completely immersive, closed hypertext environment for the user. Users are forced to adopt a single hypertext application to perform all their knowledge structuring, and must commit themselves to the functionality and services made available by the designer. Hypertext applications are designed as monolithic systems which aim to provide all the functionality required by the user, and do not allow other existing applications and tools to be used concurrently. This means that hypertext users are forced to give up any existing tools and practices, and instead adopt the view projected by a single hypertext system.

This is an unrealistic situation in modern computing environments; users already have a diverse range of tools and applications at their disposal which perform domain-specific, highly-specialised tasks. It is simply not practical to ask a user to relinquish all these tools, in favour of a single hypertext application. Developers cannot hope to implement all of the user's existing applications in a single hypertext system; this is neither practical nor sustainable, and may explain why many users have been reluctant to adopt hypertext ideas in their environment [Mey89].

Malcolm [Mal91] discusses a typical scenario in an engineering environment, and identifies many areas which closed, insular hypertext applications fail to address – concurrent engineering tasks, collaborative processes, sharing of information and access control mechanisms. Engineers use many diverse applications on heterogeneous platforms; specialised tools and services; diverse data formats. Malcolm calls for more *open* hypertext systems which no longer aim to *"own the world"*, but instead offer a more integrated approach. Hypertext can be seen as an overarching framework which underlies *all* applications. In this way, hypertext services can be made available throughout the environment, as a way not only of structuring information, but also of integrating applications (see figure 2.1).

This view of *open hypertext systems (OHS)* have proved popular with many researchers, and signals a move away from closed, isolated, monolithic hypertext applications, towards more open, ubiquitous hypertext services which are integrated

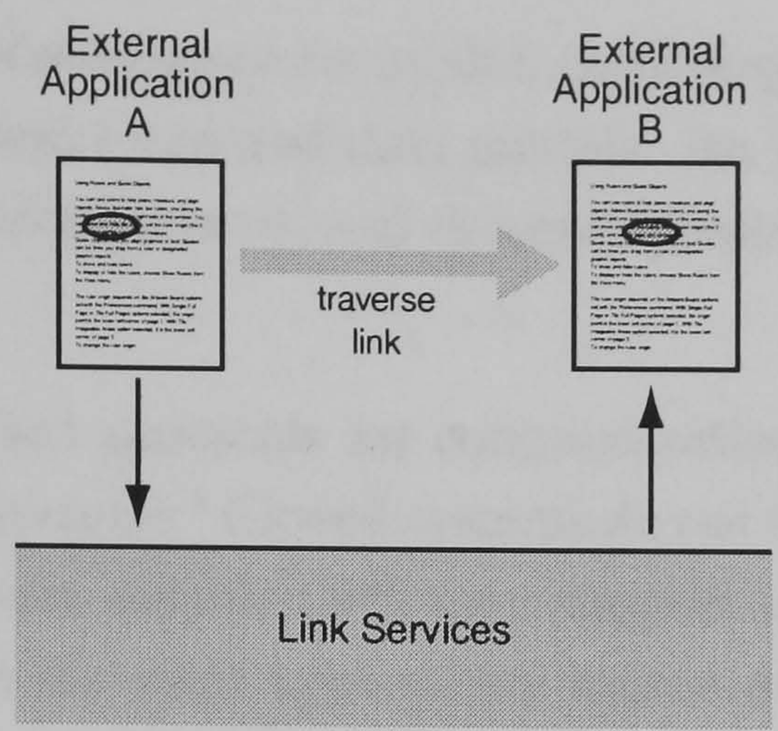


Figure 2.1: Open hypertext using link services

into the user environment [YHMD88]. Wiil *et al* [WL96] identify a number of criteria for evaluating *industrial-strength*, open hypertext applications:

- *scalability*
  Many early hypertext systems focussed on the needs of the single user, working in isolation. Modern computing environments need to support many users and tools, and large quantities of information. Open hypertext systems must be capable of managing these large spaces, and scale to meet the demands of future environments.

- *openness*
  Closed hypertext systems provide a fixed set of functions and operations, implemented as a single, tightly-coupled application. Open hypertext systems should provide a means of integrating new tools into the environment, so that they too can make use of the hypertext services.

- *distribution*
  The majority of early hypertext research was aimed at single users, working on a single, local machine. Modern computing environments need to embrace much larger topologies, incorporating many distributed platforms and machines. Users need to access remote data and resources, and share information with many users. The ideas behind distributed hypertext are discussed in more detail, later in the chapter.

- *heterogeneity*
  Closed hypertext systems present a homogeneous view of the user environment – all the tools operate on a single, known platform and share the same data model. Many aspects of the domain are fixed and defined by the system designer. While this greatly simplifies the design and implementation of the hypertext application, it also proves very limiting. Open hypertext systems

must adopt a more heterogeneous model, consisting of diverse tools and services, different architectures and data models. An OHS must support many different tools and data formats, and diverse operations.

- *interoperability*
  There are no standard protocols for communicating between hypertext services and user applications [1] Closed systems do not offer any communications protocol, while a more flexible OHS may support several protocols. This allows users to select the most appropriate means of integrating applications with the underlying hypertext services.

- *extensibility*
  Many closed systems support a number of diverse data formats and operations, which can be used by the user. Similarly, open hypertext systems should provide similar levels of support, to meet the demands of heterogeneous computing. However, no system designer can anticipate all the operations and data types which may be needed by users. For this reason, OHSs should provide some means of extensibility and tailorability so that the hypertext services can grow and develop with the user. Hypertext services can be augmented with additional functionality which the user finds lacking, and hypertext can be integrated with new applications when appropriate.

- *computation*
  The previous chapter explored some of the limitations with conventional hypertext systems, which adopted a very fixed, static view of hypertext. Modern hypertext support must adapt to a changing world and provide support for dynamic information, virtual structures and rapidly changing hyper structures [Hal87]. This idea of computation has been developed by many systems, and has been applied to different levels of the hypertext model. This support for computation in open hypertext systems is explored in more detail, later in the chapter.

Open hypertext systems provide users with a flexible, extensible environment, which is better suited to meet the demands of modern-day computing tasks. Hypertext is no longer simply a means of structuring information, but of also managing and integrating applications into a single framework. Open hypertext introduces the idea of *link services* which provide hypertext functionality to *all* applications. Users can continue to use their favourite tools and applications, instead of relinquishing control to a single, closed hypertext application. It is the application which

[1] Readers are referred to the current research into open hypertext protocols and standard open hypertext frameworks [DLR96, GLD97, And97].

chooses the abstractions and operations which can be supported, and uses the definition of *"node"* which best suits the problem domain. A number of criteria and design requirements for OHSs have been included, and the remainder of this section explores some of these in more detail, and discusses the different approaches taken by system designers.

2.1.1 Integrating Applications

This new approach views hypertext as a means of integrating applications into a common framework, so that *all* applications can make use of hypertext linking services [SLH94, LS94]. Users can continue to use their favourite editor and any specialist software, but can augment these with additional hypertext services. In order to provide these link services, designers must address the problem of *integrating* applications – that is, provide a means of communicating between the hypertext layer and the user applications. This section discusses some of the issues which must be addressed by any OHS.

The conventional approach to integration is for the link service to provide some form of protocol, which any new application must support if it is to integrate with the hypertext layer[2]. Some systems such as Sun's Link Service [Pea89] and the PROXHY system [KL91, Kac90] adopt a very simple, minimal protocol which greatly simplifies the integration of applications into the OHS framework. However, while this simplified protocol can reduce the effort required to integrate applications, it is often at the expense of functionality. Other applications such as MultiCard [RS92], implement a more complex, expressive hypertext protocol which allows a closer integration between hypertext service and application. User applications can make more efficient use of the link service, and utilise more powerful hypertext abstractions. This balance between simple, loosely-coupled protocols and more complex protocols is a difficult problem which must be addressed by system designers. Pearl [Pea89] summarises this problem of conflicting goals:

> *There is a tension between support for heterogeneity that is a goal of such open systems and the notion of integration ... which is aided by homogeneity.*

The effort involved to integrate new applications into an open hypertext is an important issue, and has significant impact on the cost and popularity of open hypertext systems. Many researchers have attempted to address this problem by offering several *degrees of conformance* [FHHD90, WL96, RS92]. For example, the Microcosm project [DHHH92] introduces the idea of *fully-aware, partially-aware* and *unaware* applications, which indicate the degree to which the tool can be integrated

[2]This section distinguishes between *hypertext* protocols and any underlying transport protocols (eg. TCP/IP, UDP etc)

with the link layer. Some applications reveal the entire internal structure of the application, which can make integration a relatively simple task. Other applications are designed as closed tools which cannot be easily tailored, and so may not be able to support all hypertext services. For example, an application may only be able to act as a viewer of node contents, and is unable to support links which originate from the node. This was the approach taken in early generations of the World Wide Web which used *helper applications* to view data formats which were not supported by the hypertext browser. This is a simple method of providing a more open, extensible hypertext environment, which greatly increases the flexibility of the system, and allows the users to use their existing tools.

There are other ways in which applications can be integrated with hypertext services, instead of defining hypertext protocols. Some systems use *wrapper* applications which communicate between the application and the link service [Kac90, DKH94, CS88]. Other systems provide programming libraries which support the appropriate hypertext operations [ATJ94] which can be linked with the application executable. Some applications such as HyperDisco [WL96], Intermedia [YHMD88, HKRC92] and Hyperform [WL92] use Object-Oriented techniques to support application integration, by providing a set of core classes which can be tailored and extended by applications. Some applications support some form of scripting or macro language which can be used to add menus etc, and provide some level of integration with an open hypertext system [DKH94]. Also, Kacmar [Kac95] suggests a method of monitoring events and messages to integrate applications which are unaware of any hypertext functionality. Whitehead [Whi97] describes an architectural framework which is used to categorise applications and evaluate different integration models. Also, there have been moves to standardise interchange formats [GNKN97, MHEG97, HS90], and develop a standard open hypertext protocol [DLR96, GLD97, And97]. Davis et al [DKH94] summarise the different approaches to integration as:

- *tailor-made viewers*
  Applications written specifically for integration with the hypertext system.

- *source code adaptation*
  Source code for the application is available, which can be modified or extended to support communication with the hypertext system.

- *object oriented re-use*
  A basic hypertext viewer class is created to implement the general integration requirements. Viewers for specific data types then inherit from this common class.

- *application interface level adaptation*

Many applications and packages provide interfaces and macro programming languages, which allow hypertext functionality to be added.

- *shim or proxy programs*
  These programs sit between the hypertext link service and the viewer. Actions from one system are translated into actions that the other can understand.

- *launch only viewers*
  The application cannot support any hypertext functionality, and can simply be launched with a specific data set.

Another issue related to the integration of applications, is the problem of storage and the responsibilities assigned to applications. Many OHS systems provide dedicated storage engines (*hyperbases*) which are explored in more detail later in the section. These storage services are responsible for managing all of the objects in the information domain – nodes, anchors, linking information, users, groups etc. Other hyperbase systems manage the hypertext-related information, but expect the application to store and manage the actual node contents. Some OHS projects go further and make additional demands on the applications; for example, the PROXHY and SP*x* systems expect the applications to manage the anchoring information, and make provisions for its storage. This is a problem for open hypertext systems which often assign responsibility for node storage to the native applications, and which can cause problems with versioning, collaboration, data integrity etc. The separation of concerns between native applications and some form of hypertext link service presents the problem of interface consistency – how to maintain the same look and feel across heterogeneous applications. Some benefits can be achieved through the development of user interface conventions [HF92, Hoe89] and combining hypertext tools with windowing libraries.

An approach which has gained popularity in many OHS systems, is to separate the linking information from the underlying node contents [DHHH92, GB80, YHMD88]. These link collections have been described variously as *linkbases, webs, contexts* etc, and allow links to be interchanged and combined arbitrarily. Links are no longer bound intrinsically to nodes, so users can select the appropriate link sets for their task, or maintain their own personal linkbases. Links can be generated dynamically or retrieved on demand, before being combined with the node objects. Nelson makes the distinction between *applicative* and *embedded* linking mechanisms [Nel95], and shows how the data remains untouched, and so can still be understood by the native applications. The separation of hypertext information can raise consistency and data integrity problems in open hypertext systems, if node data which is managed outside the hypertext hypertexts is modified by external applications. However, this separation of hypertext structure can be useful in shared environments where users access a common corpus of information, or where nodes cannot

be edited (eg. read-only material, dynamic nodes) [FHHD90]. This idea of separate linkbases is developed further in chapter 4.

2.1.2 Hyperbases: hypertext-specific storage engines

Previous implementations of hypertext applications used conventional methods for managing and storing hypertext information. Nodes and links would be stored as native files and directories, and all access would be done using the underlying file system operations (*read, write, append* etc). However, it soon became apparent that hypertext applications required more complex operations which were not available using native filesystems. Hypertext applications require efficient retrieval of node objects, and links needed to be treated as first-class objects. A hypertext engine must provide efficient support for complex structures, transaction management and multiple, shared access.

Some hypertext applications provided additional storage management using database services to manage hypertext objects (eg. Intermedia [YHMD88], VNS [SCG89]). This led to the development of more specialised, dedicated storage engines which implemented hypertext-specific operations, known as *hyperbases*. The hyperbase exists as a separate service which can provide hypertext management operations for a shared environment. For this reason, hyperbases have gained popularity in the open hypertext community, and are discussed briefly here.

Early work on hyperbases began with the HAM [CG88] hyperbase, developed by Tektronix. This was a general-purpose hypertext engine which supported multiple users and was used as the storage engine for the Neptune [DS86] hypertext engine. HAM supported hypertext-specific objects and data structures, and provided operations for manipulating these (see figure 2.2). The HAM model proved very influential and was typical of early hyperbase research, such as DGS [SSS93], Aalborg's HyperBase [KWO90] and GMD-IPSI's HyperBase [SS90]. Research on hyperbases has continued with the development of more open, extensible hypertext engines such as Hyperform [WL92], HB3 [SLH 93, SLHS93, SLH94], HyperStorM [BWAH96] etc. Hypertext designers have developed a diverse number of hyperbase implementations, and offer differing storage models. Some hyperbases attempt to manage all objects in the hypertext domain, while others only focus on the management of links. Readers are referred to the literature for further details of particular hyperbase engines, and for more information on design criteria etc.

2.1.3 Computation in open hypertext

Halasz identified a number of areas which hypertext systems had failed to address satisfactorily, in his keynote address entitled *"Seven Issues For The Next Generation Of Hypermedia Systems"* [Hal87]. In particular, Halasz called for better support for dy-

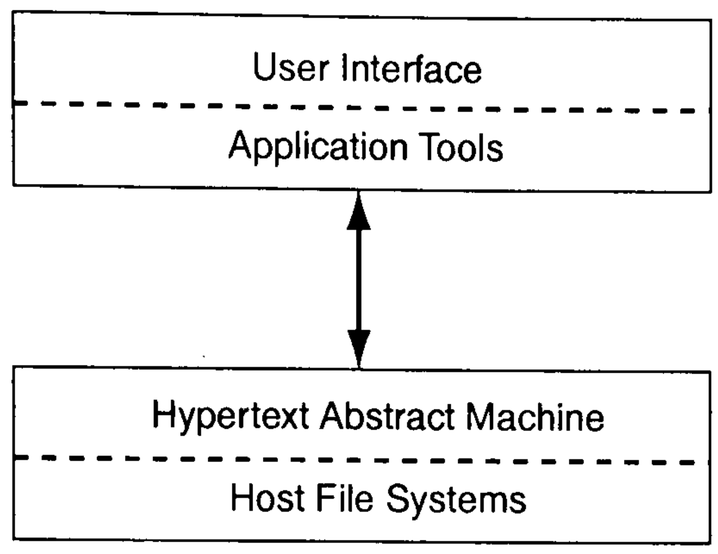


Figure 2.2: The HAM hyperbase engine

namic data structures and computation over hypertext networks. Existing systems failed to provide adequate search and query functionality, and were based around a very static notion of hypertext links. Authors could create links between nodes and group nodes together, yet these were supported as simple, static navigational links. Many system designers recognised the need to provide a more computational view of links, by attaching scripts to links (eg. HyperCard [App87], KMS [AM93], Neptune [DS86] etc). These computations could be evaluated when the user traverses a link, and allow the author to include more complex, flexible actions in the hypertext.

A computational view of hypertext can offer new opportunities to authors and users, and allows hypertext to be used in more dynamic, changing environments. A link can execute a query, generate some additional content or a link computation may be used to present novel data types etc [CS88, PYS90]. Schnase *et al* describe a scenario using the KMS system, in which scripts and computations are used to maintain a simulation and support program development [SL89]. Bieber [Bie91] shows how computation in hypertext can be used to support dynamic *decision support systems*. The Intermedia system introduces the idea of *hot, warm* and *cold* links to include dynamic information [CBY89]. Some systems even developed specialised scripting languages to support link computations, such as the HyperTalk language in the HyperCard system [App87] or the NoteCards Lisp environment [HMT87]. Ashman also [AV94, AVC94] describes a functional linking model which uses predicate logic to dynamically compute links in a hypertext. There has also been some work on the *types* of behaviours introduced into the hypertext model, with the development of some taxonomy systems [NLS97, Wat97].

The introduction of computations and link scripts builds on much of the work done on *active documents* [Spi88, TB90, CCS92, CHP88, ET94, Act]. Active documents associate a set of behaviours with conventional electronic documents, to provide a more reactive, dynamic document model. Document content can be generated on demand, to include up-to-date data or customised information. Actions

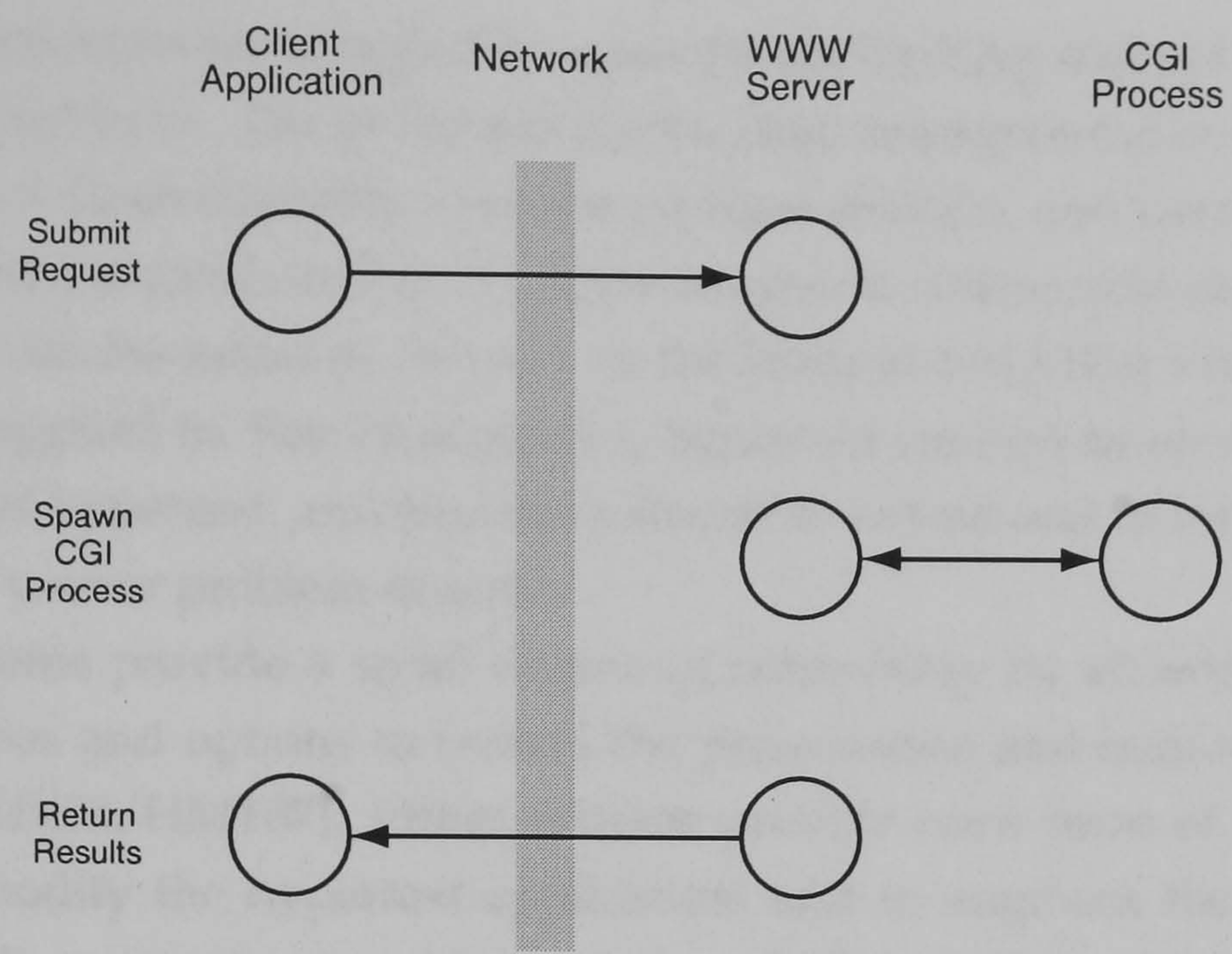


Figure 2.3: CGI link computations in the World Wide Web

can be triggered in response to user actions – scrolling a page, opening a document etc. The layout and structure of the active document can be altered and customised according to the requirements of the user or some other criteria. Some researchers have also applied some levels of computation to the hypertext paradigm, to construct presentations and *guided tours* [MI89, Zel89].

The computational view of hypertext has proved very successful in the field of open hypertext with many systems now providing more dynamic, behavioural support. The Trellis project [FS89, SF89, FSD92] explores the idea of *browsing semantics* and how these can be introduced into a hypertext model. The PROXHY system [Kac90] has done some excellent work on combining a process model with the conventional hypertext model, and implements a model based on sets of processes. There are many other influential systems which offer interesting approaches to behavioural hypertext, such as D^2[HGC94], SP*x* [SLH94, LS94], Microcosm [DHHH92, FHHD90], HOSS [NLS96], HyperDisco [WL96], Hyperform [WL92] etc. The World Wide Web has also seen the introduction of link behaviours using *Common Gateway Interface (CGI)* [WWWa] (figure 2.3) and the development of mobile programming languages such as Java [GM95], JavaScript [JS] etc. These have all inspired the work on HIPPO discussed in Chapter 5, which introduces an adaptive behavioural model, based on communicating processes.

2.1.4 Extensibility and tailorability

The established hypertext model provides the user with a number of very powerful abstractions which can be used to manipulate information spaces, and can be used for a wide range of problems and applications. However, users often find that

existing implementations of hypertext concepts are limiting and not ideally suited to particular problems. The particular abstractions implemented by the hypertext system may not sit comfortably with the problem domain, and users may wish to edit and extend the functionality of the environment. The system designer cannot always anticipate the needs of the user, or the kinds of tasks that a hypertext environment will applied to. For these reasons, hypertext systems must support a more flexible view of hypertext, and provide a means to extend and tailor the system to the particular user or problem-domain.

Some systems provide a small degree of tailorability by allowing the user to select properties and options to control the presentation and interaction with the hypertext [AMY88, HMT87]. Other systems provide some form of programming interface to modify the hypertext application, and to augment the functionality. The NoteCards system is a notable example, which provides a *Lisp* interface, and has been used to introduce additional node types etc [TMH87]. Some approaches to open hypertext have adopted an object-oriented paradigm, and provide a set of core classes and behaviours which can be extended and tailored to suit the particular needs (Intermedia [HKRC92], HyperDisco [WL96]). However, there are few guidelines as to how, and when a hypertext system should be extended and tailored. The work on HIPPO in Chapter 5 helps to address some of these problems, in an attempt to design open hypertext systems which are better suited to the demands of diverse computing environments.

2.1.5 Formal models and hypertext standards

Hypertext researchers have taken a diverse range of approaches to the problem of hypertext design and open hypertext. This can result in incompatible data models, and widely differing data formats and protocols. One of the fundamental goals of open hypertext research is to provide a ubiquitous, integrated environment in which applications can operate simultaneously, and users can incorporate hypertext functionality into their existing tools. However, the hypertext community has developed systems independently of each other, systems which are largely incompatible and cannot easily be integrated together.

This problem of hypertext systems which operate in isolation has seen the development of *formal models* and *hypertext standards*. Formal models aim to provide a rigorous framework for defining hypertext applications and hypertext operations. They offer a number of advantages over more informal techniques by allowing different applications to be compared and contrasted. Interchange and interoperability standards can be developed to allow hypertexts to be exchanged between diverse hypertext applications. Also, a formal model establishes a common language for communicating hypertext ideas, and allows hypertext models to be evaluated in an abstract fashion. Hypertext abstractions which are expressed in a formal framework

are independent of any implementation, so can be reused in different applications.

One of the first formal models to be devised was the Dexter model [HS90] which has proved hugely influential in the hypertext community. The Dexter model was developed towards the end of the 1980s, and was intended to capture the fundamental concepts and abstractions which were common to hypertext systems of the time. Although the Dexter model does not strictly address open hypertext systems, it does attempt to explore the fundamental abstractions found in classical hypertext systems, and has become one of the most important and influential models to emerge from the field of hypertext research. The Dexter reference model includes many of the concepts which have found their way into open hypertext theory, such as the separation of application data from hypertext information, and has influenced the design of many current open systems.

The Dexter model was defined as a layered model, and is explored in more detail in Appendix B. Although some researchers have identified problems with the model [LS94, GHMS94, Gro94b], the Dexter model has been largely successful in capturing the essential characteristics of earlier hypertext systems. However, the Dexter model does not capture many of the more recent advances in hypertext research, and Leggett *et al* [LS94] argue that it will not scale well to meet the demands of future open hypertext.

The Dexter model has been developed in a number of ways, through the DHM system [GT94], and more recently with the Flag Taxonomy [OKW96]. The Flag model extends the Dexter model, to address issues that relate specifically to open hypertext, such as application integration etc. Other formal models have been developed which emphasise different aspects of hypertext design [GP93, FS90, CB89, Lan90, CT91, dBH92], and a number of formal design methodologies have emerged from this work [SR95].

The development of hypertext standards and models has helped to address the problem of hypertext interchange and the exchange of information between diverse hypertext systems (see figure 2.4). The Dexter model has been used as the basis for exchanging hypertexts between KMS and Intermedia [LS94, LK91]. Similarly, The HyTime standard [GNKN97] emerged from work on the SGML [Gol90] document markup language, to provide a platform independent way of describing hypertexts. Other standards which relate to hypertext interchange include MHEG [MHEG97] and CMIF [RvOHB97], and these have much in common with other document exchange standards (DSSSL [DSSSL96], ODA [ODA85], SGML [SGML85], PDF [BC93] etc). The World Wide Web has also been very influential, both in gaining widespread acceptance of hypertext, and in the development of interchange standards. The HTML [WWW98d] language provides a means of describing simple hypertext documents, and has seen a number of recent developments such as XML [WWW98c], CSS [WWW98a], XSL [ABC 97] and the *Document Object Model*

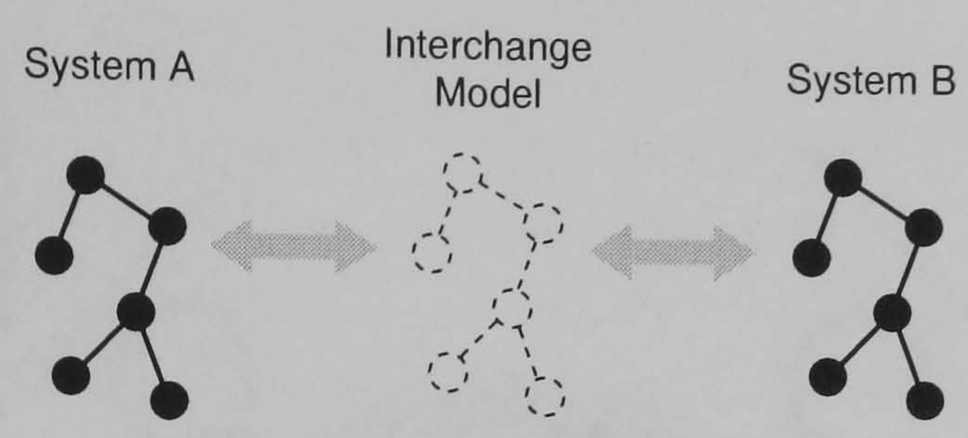


Figure 2.4: Hypertext interchange models

(DOM) [WWW98b] etc. The WWW is described in more detail in Appendix B along with other significant open hypertext systems.

2.2 Distributed and Collaborative Hypertext

As the popularity of hypertext applications has increased, the limitations of a single user working in isolation have became more apparent. Modern computing tasks demand an increasingly collaborative environment involving many users and resources. Users need to share information with users who may resides large distances away, or may need to access information which reside on different machines. While some open hypertext systems do address some of these aspects, the discussion of modern hypertext has mostly ignored the problems of *distributed* and *collaborative* hypertext.

Distributed systems provide access to remote applications and data repositories, and allow efficient sharing of information between users; colleagues are able to make their work available to others, and discuss and communicate ideas with the minimum of effort. Distributed systems are more resistant to system failures, and users can make better use of resources which can be accessed across the network. A distributed approach to hypertext reduces the redundancy and replication of information, and allows computations and data access to be performed at the appropriate location. Collaboration and distribution are vital for communicating ideas and effective information processing and must be supported by future systems; indeed, the hypertext paradigm is particularly suited to these areas, by providing a natural platform for referencing and sharing data and expressing complex relationships.

2.2.1 Degrees Of Distribution

Distributed ideas can be applied at different levels in the hypertext paradigm, and different systems have interpreted the idea of *distribution* in widely differing ways. Many early systems such as KMS [AMY88] allowed users to access remote nodes by using mechanisms for accessing remote data which were already present in the underlying platform (eg. Sun's Network Filing System [NFS89] which presents remote

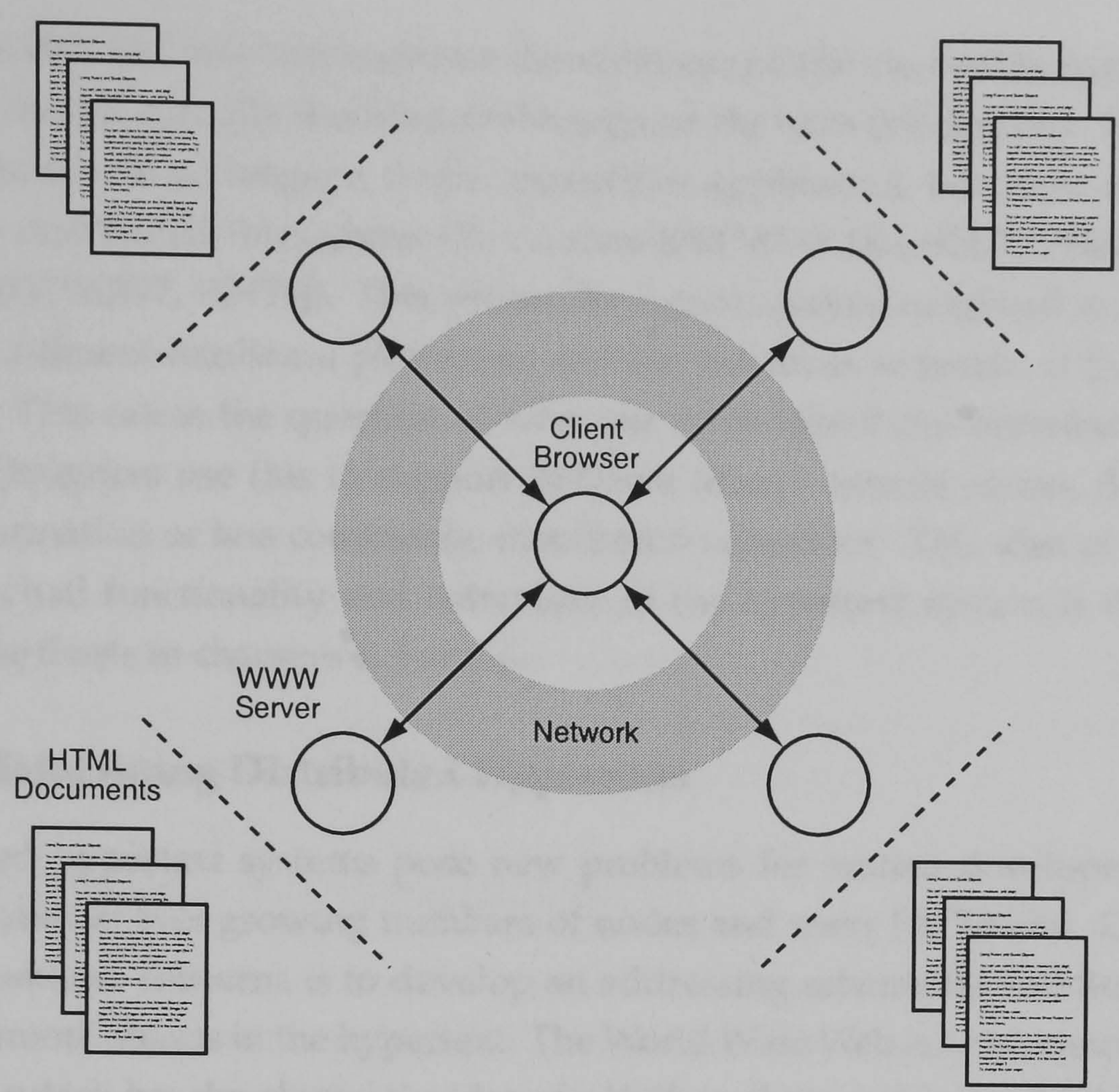


Figure 2.5: Distributed hypertext in the WWW

disc partitions as if they are local). In this case, the hypertext system is completely unaware of any distribution, and treats all node accesses as local operations.

Other hypertext applications, such as the WWW [WWWa][3] define their own transfer protocol to support distribution. In this case, client applications communicate with the document servers using the *http* protocol [HTTP], to request remote nodes (figure 2.5). This additional overhead requires all client-server tools to understand this protocol, but avoids the need for the platform-specific distribution services described previously (NFS etc).

Other hypertext applications have adopted a more centralised view, by applying distribution at the application level, while maintaining a single, centralised server (eg. EHTS [Wii91a]). This can limit the domain of information available to the user, but has the advantage that all hypertext interaction and requests for information are performed through a single server. This allows the server to monitor all transactions, and has significant implications for collaborative hypertext – this is discussed in more detail in the following sections.

However, perhaps the most interesting (and most demanding) approaches to distributed hypertext attempt to apply distributed concepts to *all* levels of the hy-

[3]The term *application* is used loosely to refer to the collection of client tools, protocols, document servers and related standards which collectively form the World Wide Web

pertext model. Not only can nodes reside on remote platforms, but the hypertext application itself is actually distributed throughout the network domain. The hypertext application is no longer a single, monolithic application, but consists of many resources distributed throughout the domain (PROXHY [Kac90], D^2[HGC94], Microcosm [GDHR97, HH94]). This allows the computational overhead to be shared between different hardware platforms, and for resources to reside at the optimal location. This raises the question of what the term "*distributed hypertext*" actually means. Designers use this to support different ideas – remote access, distributed node information or less commonly, distributed behaviour. This idea of distributing the actual functionality and behaviour of the hypertext system is developed later in the thesis in chapters 4, 5 and 6.

2.2.2 Maintaining Distributed Hypertexts

Distributed hypertext systems pose new problems for system developers, as hypertexts contain ever growing numbers of nodes and users [ACDC96]. One of the most immediate concerns is to develop an addressing scheme for locating and accessing remote objects in the hypertext. The World Wide Web is an obvious example [WWWa] which has developed the idea of a *Uniform Resource Locator* (URL) [URL94] to access remote documents. This addressing method defines an arbitrary object in the Web, in terms of the protocol used to receive it, the symbolic name of the document server it resides on, and some unique path within the remote server space. This simple method has proved to be quite effective in the web community, and makes use of existing transfer protocols and the DNS address service. However, the existing WWW addressing scheme does not scale well to a changing, dynamic system in which nodes constantly appear, disappear and move location. Users often have to deal with broken links and out of date nodes, which has led to some research into alternative naming schemes such as URIs, URNs etc[WWWb]. This problem is also seen in more general distributed systems which use trading services to manage object locations (eg. CORBA [Objc]). These component frameworks are described in section 5.4.2, and have been explored further in the design of HIPPO in chapters 4 and 5. It is uncertain whether users should be made aware of the distributed nature of the system, as this can influence the way the user understands the topology of the hypertext. Some systems such as the World Wide Web emphasise the location of objects, and make remote accesses apparent, while other systems shield the user from any distributed model.

A distributed hypertext system must also address the problems of data integrity and versioning of objects. Distributed systems have many more users than localised systems, and have the potential for large numbers of complex transactions. Nodes and links are constantly changing and being updated, and the hypertext environment must provide some means of managing this change. Halasz [Hal87]

was among the first recognise the need for versioning support, and some early systems provided version management (Neptune [DS86], PIE [GB80], Intermedia [YHMD88]). In addition, much of the work on hyperbase development has incorporated specific support for versioning of hypertext objects (CoVer [Haa92], HB3 [Hic93], Hyperform [WL92] etc). The application of existing versioning techniques to the hypertext discipline raises many interesting questions – if an object is updated, should existing links refer to the old or new object? at what granularity should the versioning mechanisms operate? should links be versioned with the data objects or separately? Distributed environments also need to ensure that the system recovers appropriately after system failures, and address the problem of transmission delays, network errors etc. There is often an increased overhead involved in accessing remote objects, and a distributed system may offer support to minimise this, or at least make the user aware of any cost or delays.

2.2.3 Concurrency and Access Control

A large-scale, shared environment must provide adequate support for access control and concurrent transactions [Mal91]. When multiple users can access and update shared information, this raises many problems relating to concurrency — reading out-of-date information, competition for access rights, unexpected results from caused by interleaving operations, deadlocks etc. Most of the approaches to concurrency control implement some form of *locking* mechanism, which grants a user exclusive access to particular hypertext objects. EHTS [Wii91a], DHM [GKM93], SEPIA [HHL 92], ABC [SS91], VNS [SCG89], HB3 [Hic93] etc all provide support for some form of access control and locking mechanisms, and readers are referred to the literature for more details of these policies.

There are many issues involved in providing a flexible, scalable concurrent architecture – how are locks allocated? how are conflicting lock requests resolved? how long can a user maintain a lock? how does a system prevent deadlock and livelock? how are others users notified about the status of locked objects? Wiil and Leggett [WL93] explore the support for concurrency control in a number of popular hypertext systems, and show the importance of a fine-grained, user controlled locking policy.

Developers of distributed systems also encounter the problem of access control and security issues. Some components of a distributed system may restrict the availability of information or prevent the execution of certain operations. A distributed hypertext system must also control the way changes are propagated through the network – are changes made to nodes immediately presented to other users, and how are redundant copies updated with the appropriate changes?

Many of the traditional concurrency techniques prove unsuitable when used with hypertext applications, as the access and sharing of information is compli-

cated by hypertext concepts and the working practices of users within hypertext environments. Alternative concurrency policies have been suggested for hypertext environments, for example, by creating new versions of objects then attempting to merge the changes after the objects have been edited [WL93]. Greif [GS87] suggests relaxing the concurrency requirements so that users *may* read out-of-date information in situations when absolute data integrity is not of paramount importance. Similarly, the KMS system [AMY88] adopts an *optimistic concurrency* policy, in which it is assumed that the number of user conflicts in a large hypertext system will be minimal, and so can be ignored in almost all cases. The development of effective access control and concurrency mechanisms continues to improve with the interest in dedicated hyperbases, which are seen in an increasing number of open hypertext systems.

2.2.4 Modes Of Collaboration

The distribution of hypertext systems offers many advantages to the hypertext developer – larger information spaces, robust systems, scalable architectures. However, many of the tasks which hypertext have been applied to, demand a more *collaborative* model which allows users to share information and work together with other colleagues. Malcolm outlines a scenario based around collaborating engineers [Mal91], and Streitz [Str96] identifies many advantages of collaborative hypertext systems – multiple views, links to other people's contributions, development of a shared space, annotations, comments, reuse of information etc. The ideas of hypertext are ideally suited to collaborative environments, because they support complex information spaces, and encourage these to develop over time. However, it is important to note that support for multiple users does not indicate a collaborative environment. Computer supported collaborative work (CSCW) requires support for the complex demands of group work – sharing information, brainstorming, discussion, decision making, meetings, argumentation, versioning etc.

Many systems have implemented varying degrees of support for collaboration ranging from simple sharing of information, to shared views and video-conferencing (Augment [Eng84b], NoteCards [IT89, TSH86], gIBIS [CB89], DHM [GKM93], SEPIA [HHL 92], InterNote [CBY89] etc). Many researchers have noted the importance of notification and awareness [Haa97, Hal87, WL93, HHL 92], so that users are made aware of any changes to the shared information space. Grønbæk *et al* [GHMS93] identify six modes of cooperation in collaborative hypertext systems:

- *Separate responsibilities*
  The information space is divided into disjoint areas, and each part is manipulated by at most one user (eg. WWW [WWWa]). Other areas of the hypertext can be explored by users, but the collaboration is minimal.

- *Turn taking*
  As before, each section of the information space is only manipulated by a single user, but users can take it in turns to manipulate the same section (eg. NoteCards [TSH86]).

- *Dynamic exchange*
  Users can exchange parts dynamically, using some form of locking mechanism; in this scenario, a user can request a lock which may be held by another user (eg. EHTS [Wii91b]).

- *Alternative versions*
  This approach supports concurrent access by maintaining multiple versions of hypertext objects. These versions may then have to be merged together at a later date [WL93].

- *Mutual sessions*
  Several users may work on the same data objects at the same time, and any changes are immediately made available to all other users [HHL 92].

- *Fully synchronous sessions*
  This is similar to the previous mode of cooperation, although users will use a shared window space, and will each have the same view of the hypertext [HHL 92].

Halasz also suggests the importance of social interaction support, so that users can communicate with each other, and discuss ideas before committing them to the hypertext [Hal87]. The EHTS application [Wii91a] also supports a limited form of communication, using *talk nodes*. Collaborative hypertext systems raise many issues in addition to those already mentioned – privacy, security, group management, group awareness etc – yet this thesis does not explore collaborative hypertext in any further detail. However, it is important to mention some aspects of this area to show how the hypertext models have developed since early systems. Hypertext is a natural paradigm for collaborative work, and many aspects of group interaction and the effects of multiple users are incorporated into the development of the HIPPO system. In particular, the development of distributed model of HIPPO in Chapter 5 explores some distributed and collaborative issues.

Distributed systems must address a large and diverse number of issues to meet the demands of modern information processing, and must tackle many problems which can be avoided or simplified in single-user, isolated applications. Distributed systems require a different approach to software design, and these distributed ideas

must be incorporated into the underlying hypertext model at an early stage. Many hypertext systems were designed as single, localised applications, and designers have often found it difficult to implement distributed ideas on top of these systems [Bro94]. Distributed models are an important consideration when designing hypertext systems, and designers such as Schnase *et. al* [SLH94] identify distribution as a key design objective. In turn, a distributed approach to hypertext design often promotes a more scalable and open environment, through the use of clearly defined interfaces and protocols etc. The previous discussion has explored some of the more important issues involved in implementing large-scale collaborative hypertext systems. Much of the work on the HIPPO system is influenced by these distributed ideas, and the work in chapters 4, 5 and 6 builds on some of the ideas presented here.

2.3 Adaptive Hypertext

Adaptive hypertext is a relatively new area of research, which argues that a hypertext model should be tailored to meet the needs of the individual user. While a general-purpose approach to hypertext design can be useful, it is clear that each user has a different level of expertise and has different goals and interests. An adaptive hypertext aims to reflect the needs of each individual user by changing and *adapting* with the user. An adaptive system identifies some features of the user, and uses these to shape the content of the hypertext and manipulate the underlying hypertext model. Adaptive hypertext techniques have been applied to many different areas – education [PLGU95, Bea94, BSW96], on-line information systems [HBG96, MS96, BE94], help systems [dRCP93, Gru93, GH95], information retrieval [MC94, KFC93], program development [ON94] etc. This adaptive approach to hypertext has been central to the development of the HIPPO model, and some of the current approaches are discussed here in this section.

2.3.1 Adaptive Features

Adaptive hypertext systems can identify many different criteria and user features to influence the adaptation of the system. Brusilovsky [Bru96] identifies four key areas which are commonly used in current adaptive hypertext systems:

- *Knowledge*
  One of the most important features used to influence adaptation is the knowledge of the user. This means that an adaptive system must be able to recognise changes in the user's knowledge, and use this accordingly to update the hypertext. A popular technique to achieve this is the *overlay model* which structures the domain as a series of related concepts. The concepts form a type of

semantic network, which is intended to represent the subject domain – these concepts may be information topics, subjects, objects etc. This overlay model is then used to represent the current state of the user's knowledge about a subject; each concept is assigned a confidence value to indicate the level of knowledge (these may be binary values, probabilities etc). This simple approach can be widely generalised to many different subject-domains and has been used in a number of adaptive hypertext systems (Hypadapter [HBG96], HyperTutor [PLGU95], PUSH [HKr 96] etc). Other applications have adopted a simpler method for modelling user knowledge, using a *stereotype model*. This approach identifies several categories or stereotypes of users, which are considered representative of the user community (eg. *novice-intermediate-expert, student-teacher-developer* etc). This is very simple to implement in adaptive systems, but often does not have sufficient granularity for many applications. For this reason, the stereotype model is often combined with an overlay model.

- *Goals*

  Another feature used by adaptive models looks at the aims and goals of the user, and how best these can be achieved. A user may be looking for information on a particular subject area or attempting to solve a particular problem. The application can represent the goals and tasks that it recognises, and may structure these as some form of hierarchy, or using probability values. The system can then identify the user's current task, and use this to guide them through the hypertext, or present with a particular information set [Gru93, HKr 96, MS96]. Most applications use a goal-driven approach to provide some adaptive navigation techniques. The goals of the user will often change between each session, and an adaptive system may identify a set of low-level goals (perhaps the current problem) and also high-level goals (an overall aim).

- *Background and Experience*

  This is an area which is similar in nature to the user's *knowledge*, but has some important differences. The user's *background* addresses the information and user knowledge concerning subjects outside the realm of the particular hypertext. Background knowledge looks at other information which, although initially unrelated to the hypertext, may affect the way the user interacts with the hypertext [dRCP93, Bea94]. Similarly, the user *experience* is interested in the familiarity of the user with the hypertext – can they navigate easily around the hypertext? are they familiar with the layout and topology? These are all areas which will affect the user's interpretation of the hypertext, and so can be used to influence the adaptation of the hypertext.

- *Preferences*

The final area which is explored here is the idea of *user preferences*. This approach allows users to overrule any of the other features, and express a particular preference for (or rejection of) certain criteria. Unlike the previous features, user preferences cannot be deduced by the system, and the application must rely on explicit feedback from the user. Some hypertext systems use preferences in a very simple way to alter the presentation of information, and configure the hypertext environment (eg. KMS [AMY88], NoteCards [HMT87], Netscape [Net] etc). However, Brusilovsky categorises this as *adaptability* rather then *adaptivity*; adaptive systems attempt to generalise these preferences and use them to influence the hypertext in other ways. An adaptive model based on user preferences can be useful for building *group models* in which several users inherit the common preferences of an overall group.

2.3.2 Adaptive Techniques

The previous discussion explored a number of common techniques for adapting hypertext systems, but it is important to establish *which* areas of a hypertext model can be adapted. The contents of the nodes in the hypertext can be altered and changed to reflect the needs of the users (content-level), as well as manipulating and adapting the links in the hypertext (link-level). Links can be filtered and created in response to user features, and global maps and indexes can be tailored for the individual user. A novice user may be presented with an outline of a hypertext node, while more experienced users may have access to more detailed information. Similarly, expert users may be provided with additional links to further material, or some links may be included which reflect the particular needs of the individual. Hypertext links can be adaptively ordered to present the optimal link to follow, or links which are considered irrelevant to the user can be hidden from view.

Research into adaptive hypertext has developed many different techniques to support the adaptation of hypertext content – prerequisite and comparative explanations, variant pages and fragments, conditional text, frame-based adaptation etc. A number of techniques which focus on adaptive navigation have also been employed – global and local guidance techniques, orientation, personalised views etc. Adaptive systems can also monitor the user's browsing patterns to infer information about the user, or can ask the user for direct feedback on the hypertext. These are all ideas which have been used in different adaptive hypertext systems, with varying degrees of success.

The idea of adaptive hypertext is very promising, and one which has been largely ignored by the open hypertext community. Adaptive methods provide some use-

ful solutions to many of the problems of disorientation and confusion seen in the hypertext field, and can be useful for managing the large-scale systems which are discussed later in this thesis. Adaptive hypertext shares many of the ideas used in artificial intelligence and information retrieval research [Sal89]. Some document management systems such as Grif [QV86] also allow the user to decide on presentation mappings and support multiple document views and outlines. Many of these ideas also lead on from the early work on dynamic hypertext, and some aspects of adaptation have been used in more general hypertext applications (eg. Trellis [Sto91], Knowledge Weasel [LS93], HieNet [Cha93], VNS [SCG89], PHIDIAS [MBD 90]).

2.4 Summary

This chapter has explored some of the more significant developments in the hypertext field, and has discussed some of the issues arising from this work. The idea of open hypertext has been introduced which views hypertext, not simply as a paradigm for structuring information, but as a service for *integrating* the user's environment. Hypertext services are no longer implemented as closed, monolithic applications, but are made available as ubiquitous services for *all* applications. It is the role of the application to then implement the suitable domain-specific abstractions, which are of no concern to the hypertext services. These hypertext architectures must be scalable to meet future requirements, and cope with diverse information types, and support extensibility at all levels.

While the benefits of open systems over current working environments are clear, these future systems give rise to a number of problems. Many of the abstractions and concepts found in conventional systems do not translate to the idea of open systems, for example the concept of a node is no longer clearly defined, and the browsing and authoring of the hypertext becomes the shared responsibility of the application and the hypertext link services. Open environments cannot assume the functionality of applications which are to be used within the hypertext, and encounter problems with applications which cannot be fully integrated with the hypertext services. Also, many of the object management issues become more complicated in open systems, giving rise to problems of consistency and storage. Open environments cannot enforce a single interface across all applications, and encounter problems with dynamic information where the system must process objects which managed externally by other systems. Modern hypertext must also address the problems of *hypermedia-in-the-large* [LS94], and the idea of distributed, collaborative environments has been examined in the chapter. Finally, the ideas of adaptive hypertext have been introduced, which aim to build a more responsive model of hypertext. A hypertext system is no longer defined as a fixed, static structure which

must suit all users, but can be tailored to meet the demands of individual users.

These open, distributed and adaptive ideas are used to influence the design and development of the HIPPO hypertext system which is described in remainder of the thesis. In particular, the HIPPO research aims to identify key areas which are established in the open hypertext field – anchoring mechanisms, separation of hypertext structure and distributed hypertext models – and develop these further. Chapter 3 recognises the need for hypertext anchors to be viewed as first-class objects, and proposes a new anchoring mechanism based on *fuzzy anchors*. This model of hypertext anchors offers a flexible way of addressing which better reflects the true nature of hypertext authoring. The idea of adaptive anchors is introduced so that anchor definitions can adapt and change in response to user browsing patterns. Chapter 4 continues the idea of separating hypertext structure into *linkbases*, which is an approach taken by a number of open systems. These linkbases are combined with object-oriented techniques to provide a model based on inheritance hierarchies and distributed link brokers. This hopes to provide a linking model based on reusing and sharing link sets, and shows how this can support a scalable and distributed hypertext model. This chapter also incorporates an adaptive model using a weighted inheritance tree, and shows how this can use feedback from the user to adapt the tree. Finally, Chapter 5 shows how all of these ideas can be supported using a distributed hypertext environment based on a set of communicating services. This chapter also presents an adaptive distributed model using *document objects* and fuzzy, weighted services. The HIPPO research which is described aims to develop key abstractions in the open hypertext model, and to show how these can be used to provide a responsive, adaptive open hypertext model. Future open hypertext environments must incorporate adaptive modelling techniques if they are to provide a view of hypertext which better suits the need of the user.

Chapter 3

Fuzzy Anchors

One of the key contributions of open hypertext theory has been the elevation of conventional hypertext abstractions to the position of first-class objects. Hyperlinks have been developed to encapsulate dynamic behaviour and exist separately from the nodes they connect. Similarly, the traditional view of the anchor has undergone many changes and is no longer viewed as a simple attribute of a hyperlink. Chapter 1 outlined many issues relating to hypertext anchoring – granularity, flexible addressing, presentation, selection etc. The anchor was also developed further in Chapter 2, with systems such as Dexter [HS90], PROXHY [KL91] and Chimera [ATJ94] which viewed anchors as independent objects.

Open hypertext aims to focus on the essential characteristics of hypertext, and the fundamental abstractions which are needed to provide effective linking services. However, most open hypertext research seems to centre around the development of the link – dynamic links, link computations, separate linkbases etc. The anchor has been largely ignored by the hypertext community, and while some systems offer interesting contributions, the anchor remains much the same as in early systems. The anchor provides the basis for all linking structures, and is fundamental to a flexible hypertext linking service. Open hypertext has taken a *deconstructive* approach which identifies key features in the hypertext domain, and reduces these to their essential characteristics. Nodes are viewed as separate objects, and linking structures are managed at a separate level to the node contents. However, with the notable exception of a few projects, the hypertext community is less confident about the anchor. Anchors are still implemented as attributes of links, and have not been fully developed as first-class abstractions.

This chapter aims to develop the anchor in open hypertext, by introducing the concept of *fuzzy anchors*. Fuzzy anchors provide a more ambiguous, less discrete notion of link anchoring, which the author feels sits more comfortably with modern computing tasks. Some adaptive ideas are incorporated into the model, which attempt to provide some level of tailorability and adaptation. Fuzzy anchors respond

to the usage patterns of the user community, and use this feedback to refine the anchors in the hypertext. The chapter describes this new approach to anchoring and explores some of the advantages this has over conventional anchoring techniques. The implementation of a prototype which supports fuzzy anchoring is discussed, along with some of the key design issues which were considered.

3.1 Limitations Of The Current Anchoring Model

This section identifies some of the problems with current approaches to anchoring in modern hypertext systems. Anchors are usually implemented as data attributes belonging to the hypertext link, but it is suggested that anchors should be supported as first-class objects. Furthermore, the limitations of the current approaches to addressing are explored, along with the fixed, static nature of hypertext anchors.

3.1.1 Anchors As First-Class Objects

Early hypertext models were based around closed, monolithic systems [App87, HMT87, AMY88], which hid the details of the system from the user. The hypertext functionality was fixed, and hypertext abstractions were implemented as internal data structures. Recent research into open hypertext attempts to identify these key abstractions, and how they can be supported in a more open, loosely-coupled environment [OHSa]. Hypertext links are viewed as first-class objects which exist in the model in their own right; these links have data values (link types, ownership information, directionality), and also behaviour associated with them. Also, many researchers have explored the benefits of storing and maintaining hypertext links separately from the node contents [Pea89, FHHD90, GT94, HS90].

However, the anchor is rarely viewed as a fundamental abstraction in a hypertext environment, and is usually supported using data values which are associated with the appropriate links. While this approach can be sufficient for many situations, the separation of anchors from other hypertext objects can be very useful. Leggett *et al* [KL91] separate the anchor behaviour and the anchor addressing from the links, and show how this provides a more extensible environment. Links can be maintained independently of any anchoring objects and anchors can be updated and manipulated without interfering with the links themselves. The separation of the addressing mechanisms from any hypertext structures means that the hypertext service need not be aware of any particular data format or unusual form of addressing. The view of anchors as first-class objects allows anchors to be shared and managed as separate entities, so that links can share the same anchor etc. Furthermore, the separation of anchoring concerns allows the anchoring mechanisms to be developed and refined without affecting the rest of the hypertext system. This is very important for a scalable, open hypertext system.

These are all persuasive reasons why hypertext anchors should be elevated to the status of first-class objects, equal to that of nodes and links. The separation of anchoring mechanisms encourages further research into hypertext anchors, and provides a more extensible approach to hypertext linking (see figure 3.1). While a few systems have begun to adopt this approach [Kac90, ATJ94], the majority of open hypertext systems continue to adopt the conventional view of anchors as link

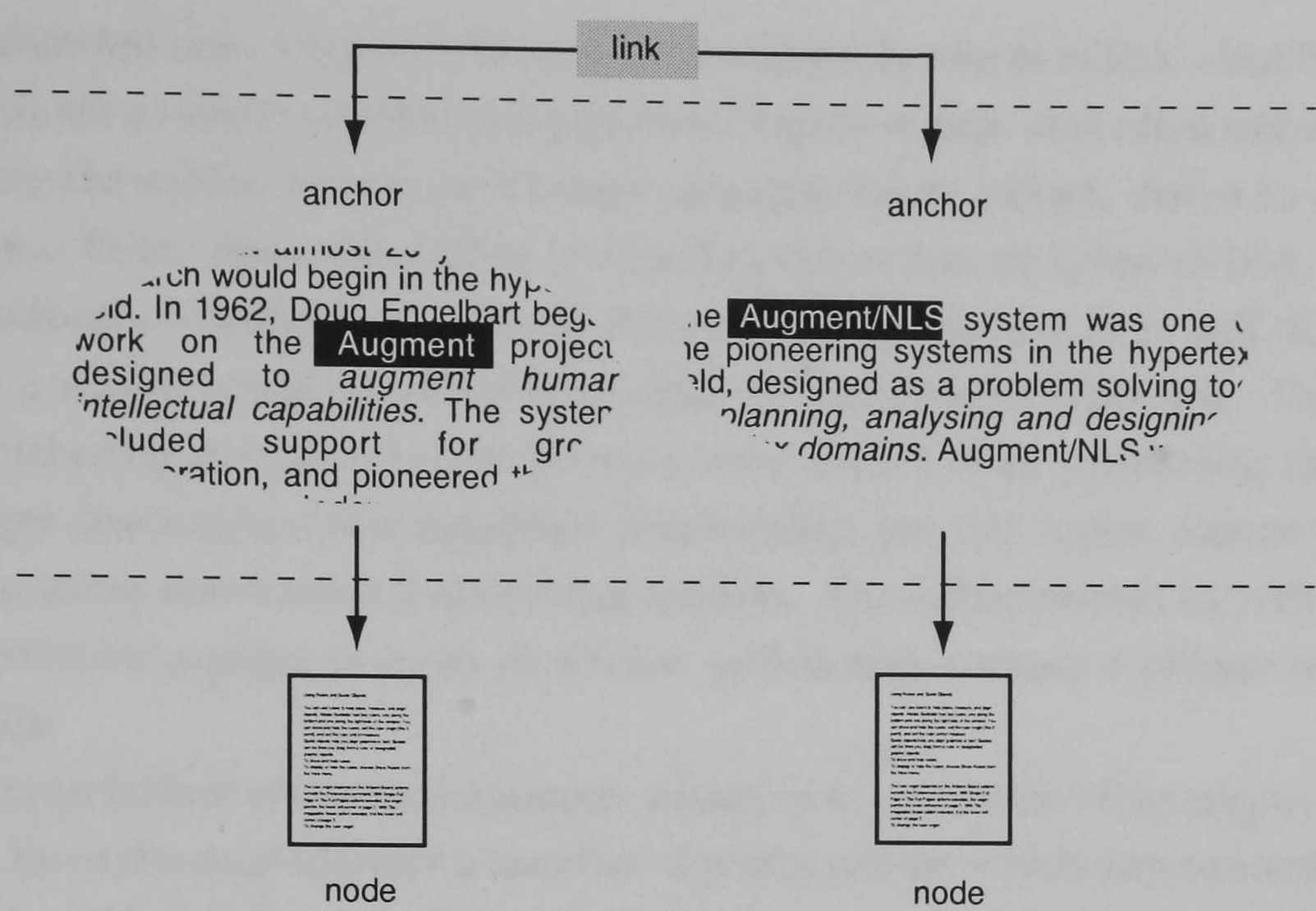


Figure 3.1: Separation of anchors, nodes and links

attributes. The concept of fuzzy anchors which is developed in this chapter views anchors as separate objects which are manipulated and managed separately from the linking structures.

3.1.2 Over-Specific Addressing Mechanisms

The development of hypertext modelling emerged from the ever-increasing volumes of information made available to the researcher. Users found themselves with access to overwhelming quantities of data, but with only primitive means of controlling it, so were unable to make full use of the information. Hypertext offered an alternative way of structuring and managing this data, and aims to provide a more natural and effective means of retrieving information. Hypertext provided ways of expressing the many complex and important relationships between objects, and ways of arranging information into coherent and useful structures. These relationships can be very subtle and intricate, and allow users to model very complex and specialised information spaces. While hypertext is a useful tool for knowledge management, in many respects current methods for defining links and specifying anchors are not well-suited to this process. Current methods of anchoring hypertext links do not allow the user to reflect the uncertainty and ambiguity which is inherent in any complex knowledge task, and these methods need to be developed further to meet the demands of modern applications.

When authors wish to model a relationship between a set of concepts, they must express this relationship within the constraints of the model provided by the hypertext system. The author may well be able to express a relationship using a hypertext

link – a directed link, a typed relationship or some other form of link – but it is much more difficult to select suitable end points for this new link. It is often unreasonable to require the author to reduce the two concepts being linked, down to a pair of endpoints. Even when the author is afforded the notion of spans of text, or some other method of content selection to define the link anchors, it is still difficult to identify a region which sums up the source or destination of the link. The reason for an anchoring mechanism is to provide some means of encapsulating the source and target destinations of a hypertext relationship, yet this seems almost impossible to do using conventional anchoring models. An anchor cannot be reduced to a single point on a page, or even an anchor which may contain a phrase or several keywords.

When an author creates a document, a page of text or some other corpus of information, he or she may identify a number of relationships which join concepts within the work and between other documents. However, each of these relationships originate, not from a single point or span of text, but from more flowing, ambiguous origins. Concepts and ideas do not simply appear out of the text in an instant, persist for several words, before disappearing again. Instead, an idea emerges much more gradually; initial ideas are introduced to support the main thread of discussion; the author develops these ideas, and rephrases and reiterates them. Concepts are referred to in later passages and reinterpreted in light of previous discussion, before fading away to give rise to further ideas.

Often, there is no single phrase that encapsulates an idea, and the *essence* of some concept cannot be expressed as a single word or a point on a page. The true notion of an anchor is a much more ambiguous, contextual object which does not have a definite start and end, which cannot be reduced to a single point, or captured in a span of several words. An anchor should not have specific boundaries which include some arbitrary *special* words, while excluding others which may still have a contribution to make to the anchor.

The conventional notion of a discrete, specific anchor with clear boundaries can be useful in some situations, indeed there are some applications where single-point anchors or anchors containing a few select words are preferable. For example, an on-line dictionary or thesaurus is a natural medium for hypertext, where terms can be linked to further definitions or alternative phrases. A source anchor would span the length of the word of interest, while the destination anchor would mark the appropriate definition of the term. Similarly, hypertexts which link citations to the full documents, or refer readers to footnotes are all applications where the conventional definition of anchors can be used unchanged.

However, these are very simple applications and do not exploit the full potential that hypertext promises. The real benefits of hypertext come from the modelling of more complex information domains, with more subtle and unusual relationships.

The construction of a link is no longer a simple, automated task, as the author identifies less clearly-defined relationships; relating the ideas in an engineering paper to a discussion on properties of materials; connecting a piece of literary work to a critique of the author's style etc. This form of linking demands a more subtle anchoring method, in which an author can identify ideas in a text which do not have a definite start and end, and which ebb and flow within the document. An anchor should be more supportive and should recognise the difficulties and uncertainty that are inherent in hypertext authoring.

3.1.3 Static Anchors

Once the authors have identified a relationship within a hypertext, they must define a hypertext link, along with the appropriate anchors at each link endpoint. Whichever anchoring mechanisms are offered by a hypertext system, the author still decides on an anchor definition, then commits this to the hypertext. These anchor (and link) definitions then become part of the hypertext structure, and persist for the remaining lifetime of the hypertext. The anchor definition is now fixed and immutable, and remains completely static once it has been stored within the hypertext environment.

While this seems to be a natural reflection of the authoring process, it does rely on the author selecting an accurate and suitable definition for the anchor. The author must identify the correct phrase or content specification for each anchor, and must ensure that they define the anchor correctly at the first attempt. It seems unlikely that the author can provide this ideal specification for each anchor on every occasion. Hypertext authoring is widely acknowledged as a very difficult and time-consuming process, and is very different from the task of writing in a linear medium. The optimal endpoint for a link is not immediately obvious, and it is only as the hypertext is used that the true nature and value of an anchor becomes apparent. The author cannot view the hypertext from the perspective of the users, and so may not understand which anchor definition is most appropriate for the task. What may appear to be a perfectly natural and accurate anchor definition for the author, may prove confusing and counter-intuitive for many users.

The construction of a hypertext aims to simplify and structure complex information spaces for the user. It therefore seems useful to involve the user in the construction of the hypertext, and to use the experiences and interaction from users as an important part of the authoring process. As users continue to browse through a hypertext, it may become clear that the original definition of an anchor could be improved or altered in some way to better capture the endpoint of a relationship. A hypertext system can monitor the browsing patterns of users to see how they perceive and understand hypertext anchors, and then use this feedback as input into the authoring cycle. The definition of an anchor need not be a *one-shot* process

in which the author must produce the ideal definition, but can be a more iterative process which constantly changes and updates the anchor.

Chapter 2 reviewed the idea of adaptive hypertext, and showed how knowledge about the user could be used to infer changes to the underlying hypertext model. However, the existing work on adaptive hypertext has focussed on the manipulation of links and node contents, and has ignored how adaptive methods could be used to alter hypertext anchors. This chapter argues that adaptive hypertext techniques should be used to modify and direct the definition of hypertext anchors, and shows some ways in which this can be achieved with fuzzy anchors. The author argues that hypertext anchors are an *emergent* abstraction which only become truly clear, as users continue to interact with the hypertext. Hypertext anchoring is an important part of the open hypertext model, which can benefit from more adaptive techniques. The anchor must be seen as a key abstraction in its own right, and should be afforded some of the advantages of adaptive methods.

3.2 Fuzzy Anchors

The concept of fuzzy anchors is introduced in this section, which aims to address the problems outlined in the previous discussion. The conventional approach to anchoring adopts a very rigid and discrete addressing mechanism; the author must select words and images which belong inside the anchor, and reject those that appear outside of it. An anchor has a strict boundary – a start and an end – and does not entertain the view that the true definition of an anchor is an emergent, analogue entity (for example, figure 1.5). Fuzzy anchors try to overcome this rigidity by incorporating the idea of *fuzzy membership*. This idea emerged from the field of set theory [Zad65], and proposes that objects can have a *degree* of membership. Objects can belong inside a set, outside a set, but also somewhere in between – slightly inside, almost outside etc.

Classical set theory uses boolean classification to define collections – *true, false, black, white, large, small*. While this can be sufficient for many applications, some situations demand partial truth values. For example, if we consider the concept of height – a person who is 6 feet tall might be classified as a *tall* person. Classical set theory therefore would define tall people, to be those who are above, or equal to this height. Similarly, anyone below this height would be classified as *small*. Clearly, this is not a true reflection of the real world – a 5' 11" person does not suddenly change from being small to tall, simply by growing 1 inch. Fuzzy sets allow this uncertainty and vagueness to be modelled explicitly, by using partial membership values. A fuzzy set may define anyone above 6 foot to be tall, but can also apply a *degree* of membership to those who are neither tall nor small (figure 3.2).

A fuzzy anchor uses this less discrete notion of membership, so that areas on

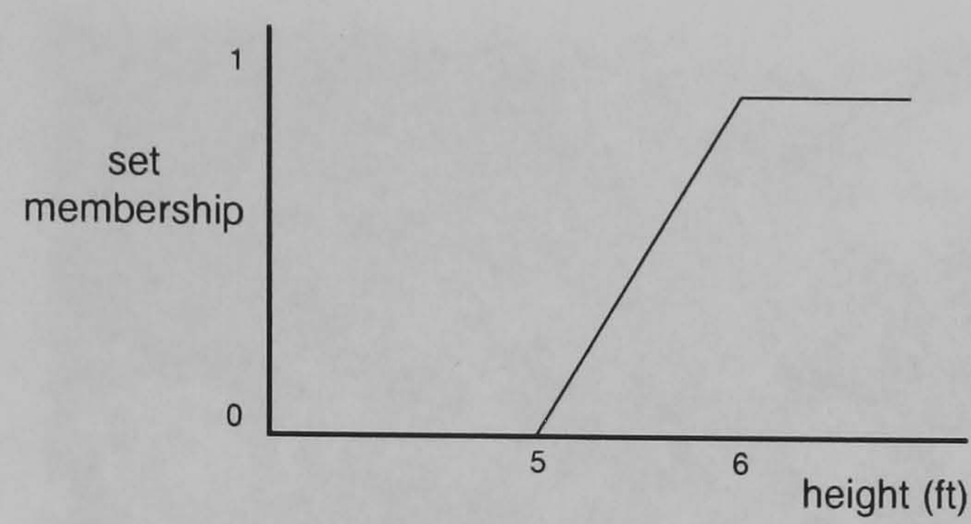


Figure 3.2: Fuzzy sets use partial truth values to model uncertainty

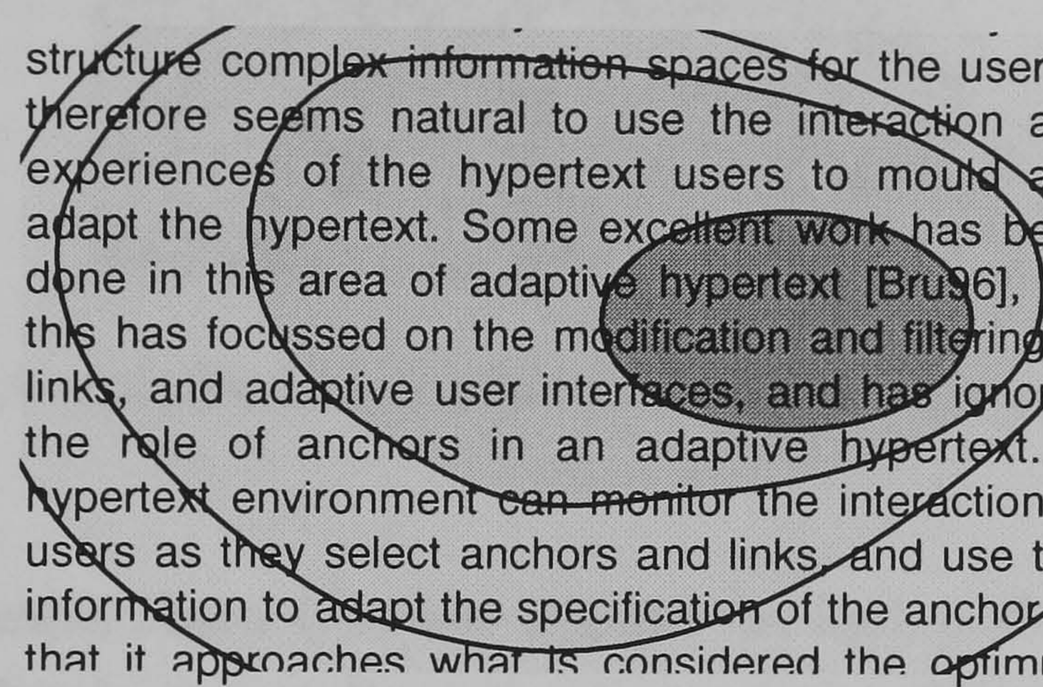


Figure 3.3: Example of a fuzzy anchor

a page can be anchored, unanchored, strongly anchored, or perhaps slightly anchored. For example, a phrase or sentence which discusses a particular concept has a natural affinity with an anchor, and should be included inside the boundary of an anchor. This would be true when using conventional anchoring models, but unlike these approaches, this is not the end of the anchor definition. The neighbouring text which leads up to the anchor also has a part to play. This introduces the concept which the anchor hopes to encapsulate, and places the central idea in some form of context. While not as important as the central text, this introductory text does have some relevance to the concept being anchored, and will be of use to the reader who encounters the anchor.

The traditional anchor forces the author to make a decision as to whether the preceding text belongs in the anchor, with equal status to the central text, or should be excluded as completely irrelevant to the hypertext link. A fuzzy anchor allows the author to include it but assign it a lower membership than other items which are more important. Similarly, some items which appear even earlier in the text may be considered important in an anchor definition, and so can be included in the fuzzy anchor, with an even lower membership value. The same is true of any trailing text which follows the central anchor concept, and perhaps closes the argument or summarises the anchor in some way (figure 3.3).

The fuzzy anchor no longer defines an anchor as a simple textual element, but

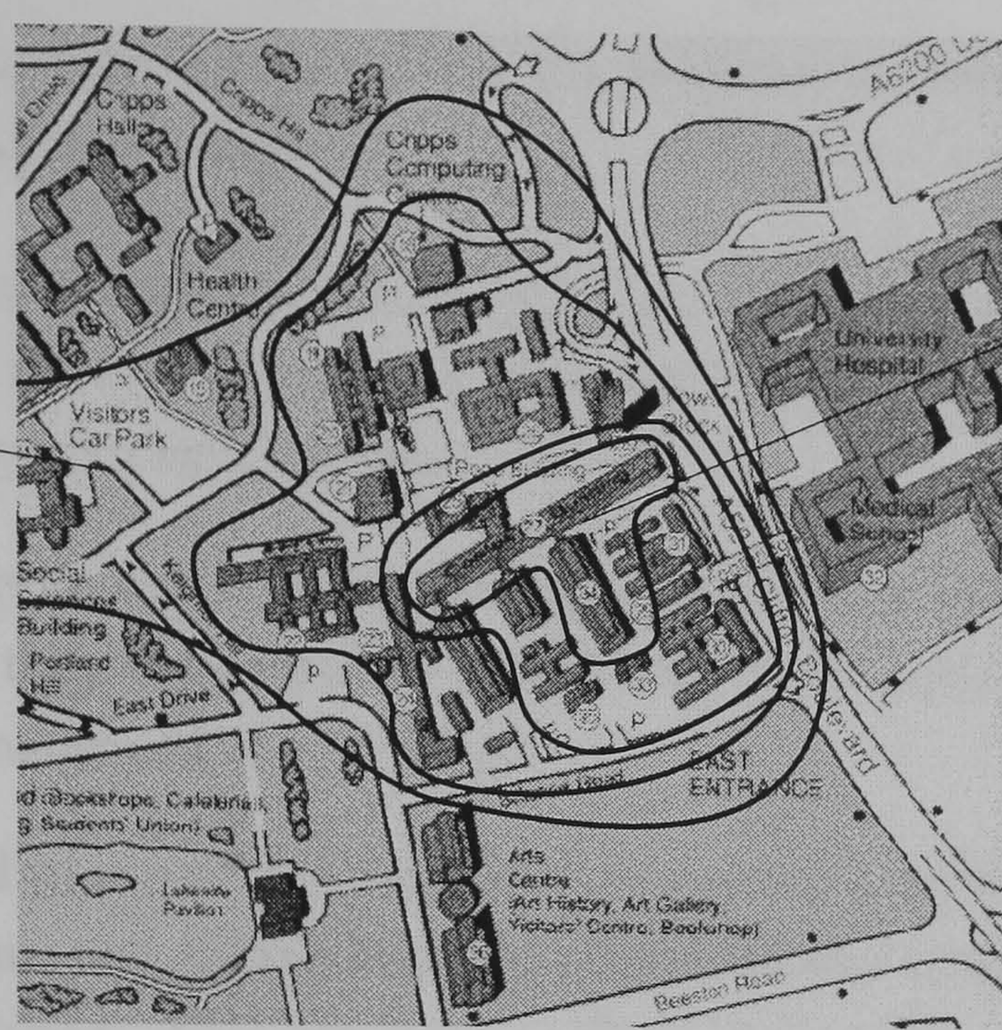


Figure 3.4: A typical example of a fuzzy anchor on a graphic image

as a collection of elements combined into a single set. Each of the constituent objects which form the anchor have membership values which reflect their importance and contribution to the anchor as a whole. A typical fuzzy anchor might be used in a graphical image to encapsulate the endpoint of a link. The main focus of the anchor might be around a particular object on the painting, and this region would have the highest membership value. Other surrounding objects might also be included in the anchor, but with lower fuzzy values. This might take the form of concentric rings which emanate out from the centre of the anchor (figure 3.4). Similar examples could be applied to textual nodes and other non-graphic media. This use of fuzzy anchors has a strong analogy with the idea of *contour maps* used in maps and engineering surveying. Each contour boundary reflects a change of height, and an implied change of importance and membership. As with hill heights, fuzzy anchors are not limited to *conical* shapes, but can contain any number of complex shapes, peaks and valleys.

3.3 Advantages Of Fuzzy Anchors

The application of fuzzy membership to an anchoring scheme offers a new flexibility to the author compared to more discrete, rigid anchoring models. Fuzzy anchors can help express the uncertainty and ambiguity inherent in a hypertext anchor, rather than rejecting it and forcing the author to make unnatural compromises. The fuzzy anchor aims to reflect the true nature of node contents, not as a series of discrete objects, but as a collection of flowing, emerging ideas and concepts which develop and reinforce each other. In particular, the notion of fuzzy anchors helps

to avoid some of the common practical problems which authors experience when using conventional anchoring models:

- The author is not forced into reducing a complex idea to a single word or phrase — this is a frequent problem with the World Wide Web, with phrases such as *"...click here for more information"*

- Conversely, authors can have such difficulty collapsing an idea to a single sound-bite, that they create anchors which are very large, to avoid the risk of leaving any elements out of an anchor.

- If a link is forced to use an anchor which is a short, precise phrase or a single span of continuous text, then it is unlikely that the anchor can fully capture the concept being linked. This can only confuse the reader, perhaps suggesting that the anchor leads to a definition or contains information specifically about the phrase that is highlighted. A fuzzy anchor which contains elements of varying membership and relevance is more descriptive and better embodies the concept being used in the link.

- With the increasing volumes of information available to the on-line user, and with the enormous success of the World Wide Web, users are no longer always satisfied with a single set of links provided by a single author. The Internet and WWW offer the potential for an almost infinite number of hypertexts, where everything is linked, and anything can be a potential anchor. Many of the more open systems are addressing these issues by offering interaction models based around query mechanisms, whereby users select objects, and ask *"is this linked to anything?"* [Hal94]. The existing anchoring methods which dictate specific, discrete selections make this situation more difficult for a user, by forcing them to make the very same precise selections that are stored in the hypertext link-anchor specifications. A fuzzy anchor offers a larger, less well-defined area, which allows the user to select anchors by selecting the overall concept or ideas in a document. In this way, users are more likely to find matching anchors.

- Fuzzy anchors have particular benefits for more graphical content and multimedia applications. Often a user will select a region on an image which represents a concept, with the hope that the author has provided some hypertext link, and an anchor at that point. Current anchoring mechanisms require the author to define anchors with definite boundaries, and the user may find that their selection lies outside of this region. However, it is clear that, just as a graphic image has no formal boundaries or distinct elements, so anchors should avoid this. Fuzzy anchors would allow users to select a region, with

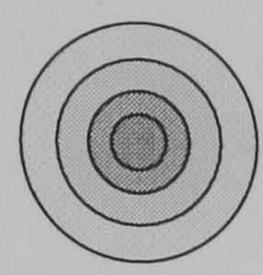


Figure 3.5: Initial specification of a fuzzy anchor

a realistic expectation that they will locate an anchor, instead of *"just missing"* the anchor.

3.4 Adaptive Fuzzy Anchors

The previous discussion criticised the static, permanent nature of traditional anchoring implementations, and argued for a more adaptive environment. It is not reasonable to require an author to identify the optimal anchor definition, and assume that this will be ideal for the users of the hypertext. A more flexible model should incorporate the users' actions and knowledge to modify and adapt the anchor definitions. An anchor should change throughout the lifetime of a hypertext, as it converges on the most favourable anchor definition.

Fuzzy anchors are well-suited to this notion of adaptive hypertext, because the fuzzy membership values which reflect the importance of the anchor elements are a natural means of supporting adaptation. An adaptive system can monitor the way a user interacts with a particular anchor, and observe the user's browsing patterns. These can be then be used to increase particular areas of the fuzzy anchor, and perhaps to decrease less important elements in the anchor. In this way, the fuzzy *contour map* will change and shift in subtle ways to reflect the understanding and interpretations of the users.

For example, consider an anchor which identifies a particular region of a graphical image, and considers this to be vital to the particular anchor definition. A fuzzy anchor would include this area and assign a high membership value to reflect its corresponding importance. The author may consider the surrounding area of this region to have a smaller, lesser role to play in the anchor, and would assign this a smaller fuzzy value. The author could continue this process arbitrarily, identifying areas which lie further from the focus of the anchor, until he or she feels the anchor has been fully described. This fuzzy anchor pattern may look like something resembling figure 3.5.

Once this anchor has been defined, it becomes part of the hypertext, and users are able to explore the hypertext by selecting the anchor. However, the author may observe that the user (or particular groups of users) consistently select a different region of the anchor – a region which may not correspond with the existing central

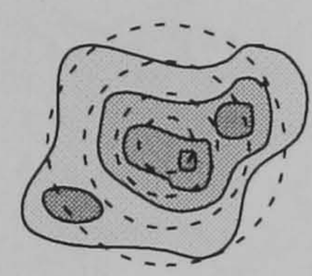


Figure 3.6: Fuzzy anchor after adaptive modelling

part of the anchor. In this case, the user does not appear to agree with the author's definition of the anchor. The user does not feel that the current *"peak"*[1] represents the true centre of the anchor which embodies the *essence* of the link endpoint. This suggests that the author made some form of misjudgement when defining the fuzzy anchor pattern, and that a more accurate anchor definition could be used, which corresponds more closely with the users' interpretation. The fuzzy values of the existing anchor could then be adjusted slightly to reflect what appears to be the true *centre of the anchor* (figure 3.6). In this way, the anchor incorporates feedback from the user, to become a more adaptive hypertext abstraction.

The scenario described above uses a very simple adaptive approach, by simply increasing regions which are popular, while reducing those that are not. However, there are many alternative adaptive methods which could be used to modify anchors, and many of the existing techniques used in adaptive hypertext were outlined in chapter 2. The current approach used to modify adaptive anchors is described later in the chapter, in the discussion of the prototype implementation. What is important is that the fuzzy anchor, and application of fuzzy membership values, provides a natural and convenient mechanism for adaptation. This was not possible with existing anchoring models which used simple spans of content or single points of origin. It is true that the author can modify and edit anchor definitions using conventional anchoring models, but this does not support *adaptation*. The fuzzy anchoring concept introduced in this chapter provides a useful method of *actively* and *automatically* adjusting hypertext anchors. This should provide more accurate anchor definitions, and support an approach to hypertext which better reflects the needs of the user. A number of suggestions for other adaptive techniques and future work are discussed in more detail in chapter 7.

3.5 Anchorbases

The development of this fuzzy anchor model elevates the anchor to the status of a first-class, primary object in the hypertext model, currently enjoyed by nodes and

[1]The term *"peak"* refers to the area of the fuzzy anchor with the greatest intensity, and therefore the logical centre of the anchor. In this case, the terms *peak* and *centre* can be used interchangeably. However, it is recognised that a more complex anchor may have several areas of equal importance, and therefore several *centres*.

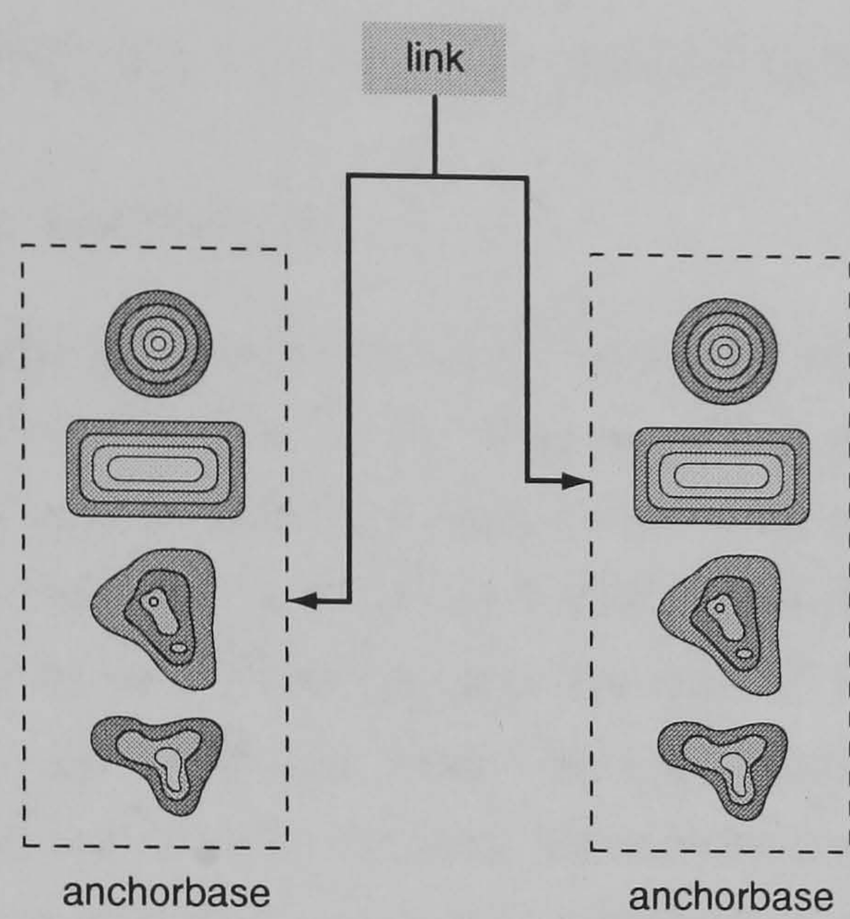


Figure 3.7: Separating anchor definitions into anchorbases

links in many open hypertext systems. Many hypertext systems have shown the benefits of separating hypertext linking information from the node contents, and have introduced the idea of the *linkbase*[FHHD90, Pea89]. Linkbases allow the hypertext system to maintain complex hypertext structures without affecting the underlying node contents which are managed by native applications. Users can also select the appropriate link sets to use, which allows them to easily tailor their hypertext.

The linkbase has proved a useful abstraction towards achieving open, scalable hypertext architectures, and this is developed further in chapter 4. However, it seems natural to apply this idea to the notion of fuzzy anchoring which has been developed in this chapter. This work has suggested that anchors be viewed as first-class hypertext objects which should be developed independently of other links and nodes. It seems natural to group related anchors together into common *anchorbases*, just as common hypertext links are collected together into linkbases. This helps to promote the idea of the anchor as a separate object, and to encourage its development, independently of other hypertext objects (figure 3.7).

The anchorbase allows the author to provide different representations for a set of anchors, tailored to the needs of a particular user, or group of users. For example, a user who is familiar with the hypertext environment may benefit from very specific fuzzy anchors, while a less experienced user may require much larger, more general, fuzzier anchors which cover larger sections of the page. The separation of anchors from the hypertext links also allows a single anchor to be shared between many links. If the anchor is edited or adapted (see section 3.4), then these changes will be reflected in each of the links which use it. Indeed, the anchorbase can be useful for supporting more specific adaptive models, so that particular sets of users can be targeted more accurately. The following section describes an approach to user

stereotyping using anchorbases, which can be used in the HIPPO model.

User Stereotypes using Anchorbases

The idea of user stereotypes has been used in a number of adaptive hypertext systems, and was discussed in section 2.3.1. This identifies particular categories and sub-groups of users that share some common behaviour, and uses this as the basis for an adaptive model. Section 3.5 suggested that the anchorbase could provide a suitable method of support user stereotypes in the HIPPO prototype, by identifying particular anchorbases with groups of users. An anchorbase represents a collection of anchor definitions which are maintained separately from the hyperlink definitions, and can share some common semantics. Similarly, an anchorbase can be used to encapsulate anchor definitions which are associated with particular user categories – for example novice users, or perhaps a user with expertise in some field.

In this way, a group of users with common interests, or shared backgrounds, experiences etc, can be assigned a stereotype label. All users who belong to this common group, can share the anchor definitions in a particular anchorbase. When a user interacts with the anchor definitions, their behaviour is only used to adapt those definitions belonging to their user group. Their actions do not affect anchors which are defined outside of their stereotype anchorbase, and so do not affect other user groups. The adaptation is confined to other members of the group, and does not affect the rest of the user base. This is a simple, but effective of developing the adaptive model for fuzzy anchors, and allows adaptive behaviour to be modelled at a finer level of granularity. The set of user stereotype groups can be easily extended by simply creating new anchorbases which correspond to a new group.

3.6 Prototype Implementation

This chapter has introduced the idea of fuzzy anchors, and shown how these can be used to address some of the problems with current anchoring models. The concept of fuzzy anchors has been implemented as part of the work on the HIPPO system, to demonstrate fuzzy anchoring, and to evaluate the effectiveness of this new approach. This section examines the prototype implementation, and explores some of the issues that were raised during the development of the application.

3.6.1 Implementation Environment

The conventional interface which has been used for hypertext applications is the familiar *point-and-click* metaphor (although other paradigms have been used – for example, non-graphical hypertext applications). Users interact with the hypertext using a mouse pointer, and are able to select anchors and traverse hypertext links.

This is a natural representation for hypertext applications, and has proven to be effective in the hypertext community. In addition, the idea of fuzzy anchors has a very strong visual element, and it is important to convey this to the user. For these reasons, it was decided to use a strong graphical environment for the prototype implementation of fuzzy anchoring.

This emphasis on a graphical front-end was very influential in the choice of development environment, and while a number of graphical and windowing environments were considered, it was decided to use the Adobe Acrobat suite of tools as a framework for development. The Acrobat software is a collection of browsers and support tools, based around the *Portable Document Format* (PDF) document interchange format [BC93]. This format has been discussed in more detail in the summary of open hypertext research in Appendix B. The PDF format provides a platform independent representation of document pages which can be viewed on heterogeneous platforms. PDF adopts a physical representation of electronic pages, based on the printed, presentational characteristics (images, text, markings etc). A flexible imaging model based on the PostScript page description language [Ado90] is supported, which allows the incorporation of complex text and images. The Acrobat software also supports a rich programming interface to allow the customisation and extension of the Acrobat tools. Additional functionality can be incorporated into the browsing tools by developing a *plug-in* component, and this is the approach adopted by the prototype. An example of the original Acrobat software is shown in figure 3.8, along with a number of example plug-in components.

The Acrobat programming interface supports the C/C++ language which was important – C++ in particular is a widely used language which offers a number useful features, and a flexible object-oriented development environment. A number of core classes were developed using C++, which implement the fuzzy anchoring and provide additional tools and services to the user. The X11 windowing environment [Nye92] was also used to provide low-level graphical operations when required, and is used as the platform for the entire prototype application. The X11 environment has been widely ported to many platforms, and offers a reliable and powerful distributed graphical environment.

3.6.2 Internal Representation Of Fuzzy Anchors

One of the main problems encountered when implementing fuzzy anchors in the HIPPO system, was to develop a flexible internal representation for the anchors. Initial designs intended to use a generic markup language to define hypertext nodes – for example the HTML language [WWW98d] – and anchors would be embedded in the node using special tags. These tags could then be used to indicate changes in fuzzy membership, and would provide a more descriptive method of defining fuzzy anchors (figure 3.9). However, this logical representation of hypertext nodes

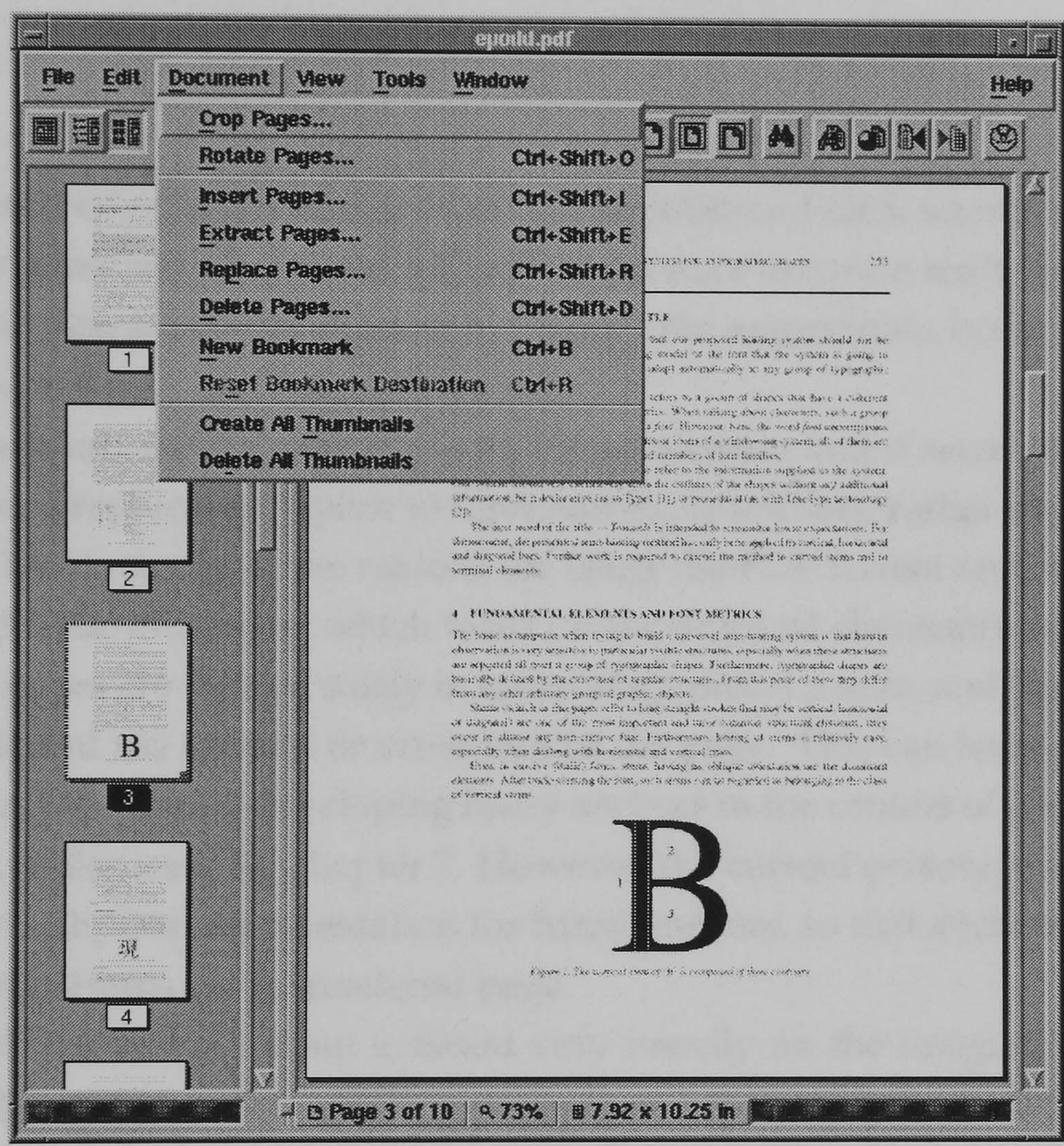


Figure 3.8: Example of Acrobat Exchange software in use

```
This is a piece of text, which contains
<fuzzyAnchor ID=1799 subID=1 weight=0.2>
fuzzy anchors.
<fuzzyAnchor ID=1799 subID=2 weight=0.4>
These anchors are defined using a
<fuzzyAnchor id=1799 subID=3 weight=0.8>
logical markup
<fuzzyAnchor id=1799 subID=4 weight=0.6>
However, this was considered too complex
to implement reliable graphical display.
<fuzzyAnchor id=1799 subID=5 weight=0.3>
This logical markup would be a useful
area to explore in future research.
</fuzzyAnchor>
```

Figure 3.9: Logical representation of fuzzy anchors

and fuzzy anchors did not transfer well to the point-and-click interface which was considered important. It proved difficult to produce accurate renderings of fuzzy anchors, and also a non-trivial issue to identify the appropriate fuzzy anchor for a given mouse click.

Fuzzy anchors have a very physical, visual element and it seems more natural to use a more graphical metaphor to represent fuzzy anchors, rather then the logical approach. This was one of the reasons for using the PDF format and Acrobat tools to implement the prototype, which focus on the physical characteristics of a document. PDF pages are viewed solely in terms of graphical marks, and the application has no notion of the content or semantics of the page. This can have some disadvantages, and the idea of developing fuzzy anchors in the context of generic, logical documents is discussed in Chapter 7. However, the current prototype implementation adopts a physical representation for fuzzy anchors, so that anchors are viewed as graphical patterns on the rendered page.

The PDF document format is based very heavily on the imaging model used in the PostScript language which is popular as a print engine for laser printers. This makes extensive use of the notion of *paths* – mathematical descriptions used to describe lines, curves, outlines, letters etc. The strong analogy between fuzzy anchors and contour maps which has been discussed previously, suggested that this idea of paths could prove useful for describing anchors. In this way, each concentric ring or change in fuzzy value could be described using a path (see figure 3.10). This provides a very efficient and compact representation for describing anchors, and suggests a closer integration with the underlying model used in the Acrobat tools.

However, after some initial development, it was decided that this use of paths presented an unacceptable overhead. For example, whenever a user selected a region on the page, the application would have to expand each of the path descriptions for each anchor, and generate the actual screen coordinates for each path. The system would then have to ascertain which paths enclosed the selected region, be-

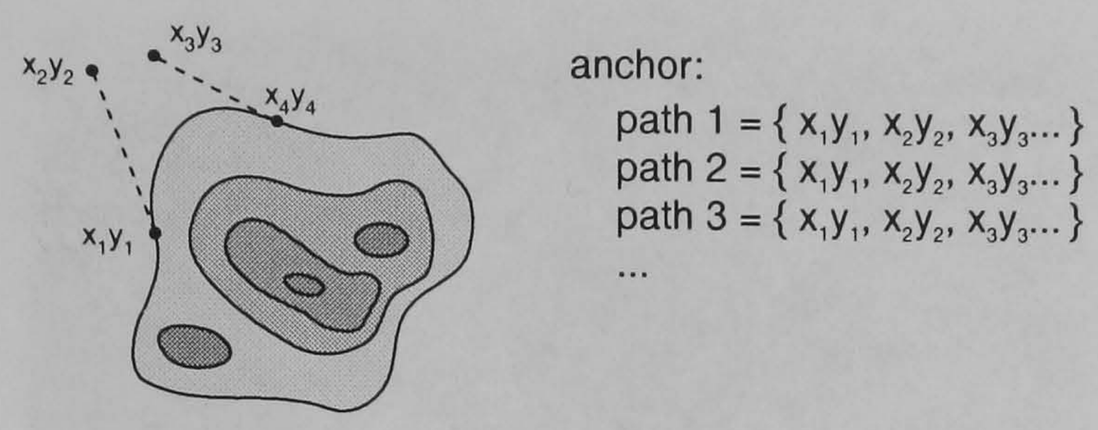


Figure 3.10: Fuzzy anchors using path descriptions

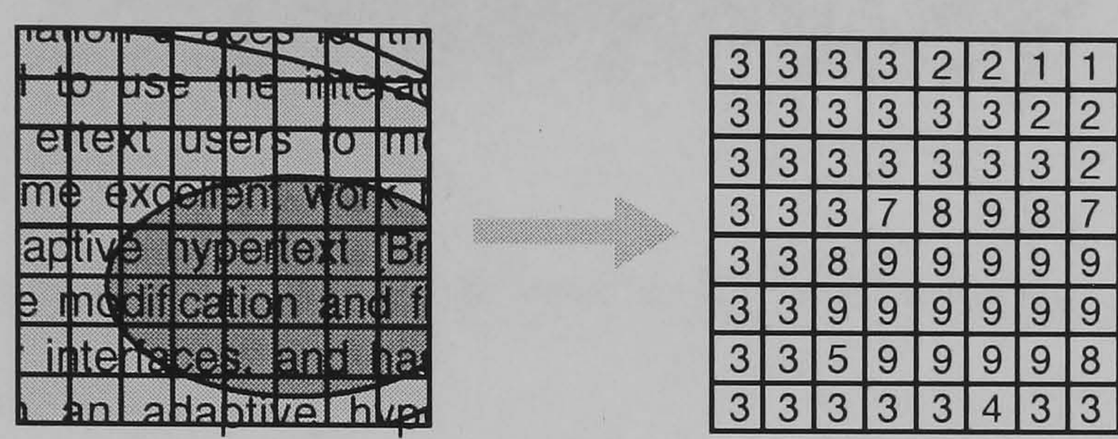


Figure 3.11: A fuzzy anchor using a matrix of values

fore matching the appropriate anchor. Similarly, the prototype application intended to offer a number of tools and utilities to help the user manipulate fuzzy anchors (see section 3.6.5), and it was decided that a representation using mathematical paths would complicate these tools.

For these reasons, it was finally decided to adopt an internal representation for fuzzy anchors based on a matrix model. Each anchor is represented using a matrix which corresponds to the physical page, where each cell represents a pixel on the screen. Each matrix cell contains a single integer value, to indicate the fuzzy membership value at that point (see figure 3.11). This anchor model is very simple to implement, and can be manipulated easily by simply changing the values stored in the cells. This matrix approach simplifies the process of mapping user selections on to fuzzy anchor regions, and also allows the easy implementation of tools to manage and manipulate fuzzy anchors. The section which discusses the adaptive methods used in the implementation also shows how this matrix model offers a natural medium for applying adaptive techniques.

3.6.3 Presentation Of Fuzzy Anchors

It had already been decided to adopt a very graphical presentation for the prototype, using the familiar point-and-click metaphor. The previous section described the internal representation of fuzzy anchors, and explained the move to view fuzzy anchors in terms of physical, graphical objects rather than logical, descriptive abstractions. It seemed natural therefore, to carry this graphical element through into

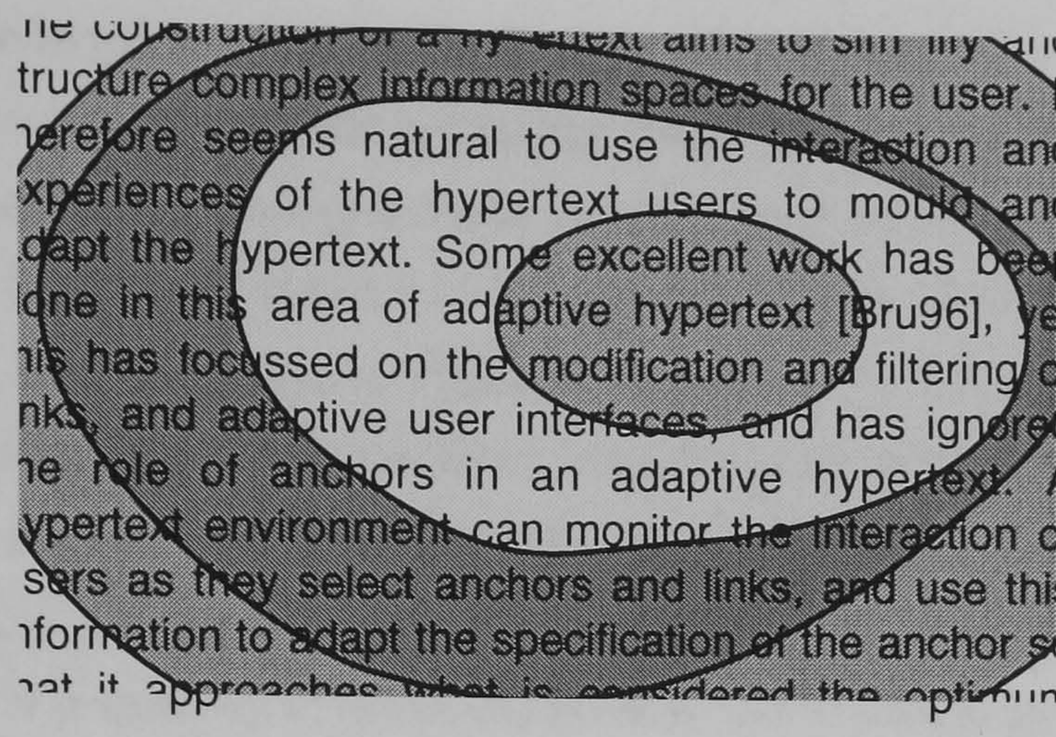


Figure 3.12: Using colour to represent anchors

the presentation of fuzzy anchors and incorporate it into the design of the user interface.

It was decided to use *colour* as a means of presenting fuzzy anchor values, by using different shades and hues of colour to indicate each fuzzy membership value in an anchor. Colour is an intuitive medium for many users, and helps to express the true nature of a fuzzy anchor, as a form of *contour map*. The use of graphical markings also enabled the user interface to incorporate many of the tools and abstractions which are more commonly seen in painting and drawing applications. For example, a user can *draw* a fuzzy anchor using a type of *paintbrush*, and treat the document as a *canvas*. Users can *erase* parts of an anchor by selecting the appropriate tools, then brushing over the existing anchor (figure 3.12). Indeed, this painting metaphor encourages the development of a whole palette of tools and brushes which can be used to create more complex and expressive fuzzy anchors. Chapter 7 includes a discussion of some more interesting utilities which could be developed in future work.

3.6.4 File Format Of Anchorbases

As the prototype implementation was developed, the information used to define each fuzzy anchor and to describe each anchorbase became ever more complex. Eventually, it was decided that a formal grammar would be developed to define the format of fuzzy anchor files, rather than using informal, ad-hoc approaches. The grammar was implemented using the *flex* and *bison* language tools, which are implementations of the familiar *lex* and *yacc* utilities [Les75, Joh75]. A full specification for the lexical tokeniser and the anchorbase grammar are included in Appendix C. The anchorbase defines the following entries:

- **id** = <*number*>

The prototype implementation is designed to operate in a shared, distributed environment which supports multiple users. Users can access any of the anchors or anchorbases, and use these in their hypertext. Similarly, the adaptive server which is described later in the chapter takes input from all users in the environment, and uses this to modify and alter the anchor definitions to produce a more accurate set of anchors which better reflect the needs of the user. This requires that each anchor must be uniquely and unambiguously identifiable throughout the entire network domain. The problem of unique naming mechanisms is a common problem in any distributed environment, and the previous chapter discussed some of the approaches taken in the World Wide Web community (URLs [URL94], URNs [WWWb] etc). It was decided that a unique numeric identifier would be sufficient for this initial implementation which could be used to identify any anchor in hypertext domain. The current prototype can also use the filename of the anchorbase which contains the anchor, so each identifier needs only to be unique within the scope of each anchorbase file.

- **page** = *<number>*
  The PDF model used in the prototype implementation presents a page-based model of each document. The matrix approach discussed earlier, describes the shape of the fuzzy anchor, and the area of the page which it covers. However, a complete anchor definition must also include the page on which the anchor appears. The PDF model does not offer any interpretation of the document contents, so the page number refers to the *logical* page number, rather than any number which may appear on the rendered page.

- **dest** = *<url>*
  The focus of this work has been to develop the anchor to the status of first-class object, and to provide an anchoring model which better expresses the uncertainty inherent in a hypertext environment. The benefits of separating anchors from the hypertext links are numerous, and have been discussed previously. The notion of *anchorbases* has been suggested in which links no longer contain the details of anchors, but reference these indirectly. In this way, anchors can be maintained separately, shared between links, and extended arbitrarily without affecting the hypertext model. However, the main purpose of the prototype implementation was to evaluate the idea of fuzzy anchors, and explore how these could be used in a hypertext environment. For this reason, this implementation includes the linking information with the anchor definition, and is described by the *dest* token.

  The current application supports URL references, as used in the World Wide Web community. The URL naming scheme references a document using the

format:

`transfer-protocol://server-location/document-path`

The prototype simply passes each URL address to an appropriate WWW browser, such as Netscape Navigator [Net], and the browser is responsible for retrieving documents. The URL scheme has also been extended in this implementation, to support linking to other PDF documents. This URL syntax allows links to reference pages *within* a PDF document, and can be useful for defining references to other parts of the same document. In this case, the Acrobat plug-in is responsible for locating the appropriate document and moving to the correct page destination.

- **xres** = <*number*>, **yres** = <*number*>
  The decision to represent fuzzy anchors using a matrix has simplified the implementation of many aspects of the prototype. However, this model based on grids is heavily tied to the resolution of the rendering device, as each screen pixel must have a corresponding cell in the anchor. The relatively high resolution of most displays means that this can present an unacceptable overhead and unreasonable storage requirements. It was recognised that the resolution of the underlying grid, only needs to represent anchors at the granularity which is adequate for the document. For example, most fuzzy anchors need only to differentiate between letters and words in a textual passage, and do not need to be defined to the accuracy of a single pixel.Therefore, the format of the anchorbases was extended to allow the author to control the resolution of the anchor matrix, by defining the horizontal (*xres*) and vertical (*yres*) dimensions. The units are expressed in terms of number of matrix cells eg. **xres = 4** would divide the screen width into four cells.

- **range** <*number*>
  Each cell in the anchor definition contains a single numeric value, which reflects the *fuzzy membership* of the anchor at that point. This value is used to indicate the importance of the region, and its contribution to the overall anchor. Some situations demand a very high level of accuracy for specifying these fuzzy values, so that small differences can be expressed. Other scenarios do not require such degrees of accuracy, and may only use several different fuzzy values to define an anchor. The *range* token allows the author to specify the uppermost value which is used in the anchor definition.

- **data** <*matrix*>
  The previous tokens have been used to define the environment for the anchor, and parameterise the anchorbase. The remainder of the anchor entry contains the matrix which includes the actual data values. The matrix consists of space

```
id = 23
page = 12
dest = http://www.ep.cs.nott.ac.uk
xres = 20
yres = 40
range = 10
data
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 1 1 1 1 2 2 2 2 2 2 1 1 0 0 0 0 0
0 0 0 1 1 1 3 3 3 3 4 3 3 1 1 0 0 0 0 0
0 0 0 1 1 1 3 4 4 4 4 4 3 2 1 0 0 0 0 0
0 0 0 0 1 3 4 4 5 6 6 6 5 4 4 2 0 0 0 0
0 0 0 0 0 1 3 4 5 7 8 8 6 5 4 3 2 1 0 0
0 0 0 0 0 1 3 3 5 7 8 9 8 6 5 4 2 0 0 0
0 0 0 0 1 1 3 4 5 7 7 7 5 5 4 3 2 1 0 0
0 0 0 1 1 1 3 3 4 5 5 5 4 4 3 3 1 0 0 0
0 0 0 0 1 1 1 2 3 3 3 3 3 2 2 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 2 2 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
...
```

Figure 3.13: A complete fuzzy anchor definition

separated numeric values, and should contain a number of values which corresponds to the resolution specified earlier (eg. a 10×30 matrix should contain 300 entries). The parser will continue to read the matrix until it reaches a "–" token; this is used to denote the end of an anchor entry, and the start of the next. By using this token, the parser can also truncate the anchor definition, and assume that all remaining matrix cells are empty. This reduces the storage requirements of many anchor definitions which do not cover the entire page.

An example anchor definition is included in figure 3.13), which defines a simple anchor shape on a 20 times 40 grid.

3.6.5 Tools And Functionality

The previous discussion has explained the idea of fuzzy anchoring, and how this has been implemented in the HIPPO prototype implementation. The internal representation of anchors and the file parsing tools provide the basic support for fuzzy anchors. This section describes the tools and operations which are made available to the user by the application.

The Toolbar

The Acrobat software defines a toolbar of icons which run along the top of each document window, and these are used to invoke commonly used operations (figure 3.14). Each Acrobat plug-in can tailor this toolbar, and provide additional buttons

which are considered useful to the user. The fuzzy anchor prototype provides three key operations which are added to the toolbar, and are also appended to the main *Tools* menu.

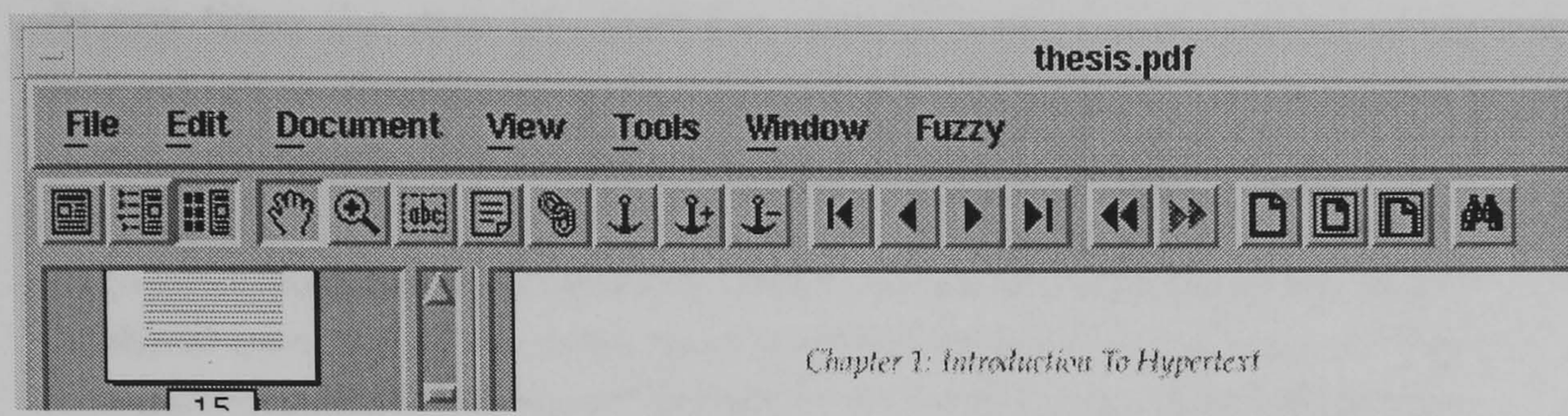


Figure 3.14: The HIPPO toolbar

- *Follow Anchor*

  This is the primary means by which the user interacts with the application, and is used to select anchors and traverse hypertext links. When the user clicks on a region of the document, the application pops up a dialog box containing all the anchors which have some presence at that point. Each entry in the dialog box indicates the fuzzy value at that point, and the target destination of the link. The user then selects the appropriate anchor (and link) to traverse (figure 3.15). This *click-and-query* cycle is similar to that seen in other open hypertext research ([FHHD90]), and this *resource-based* view of hypertext is explored later in the chapter.

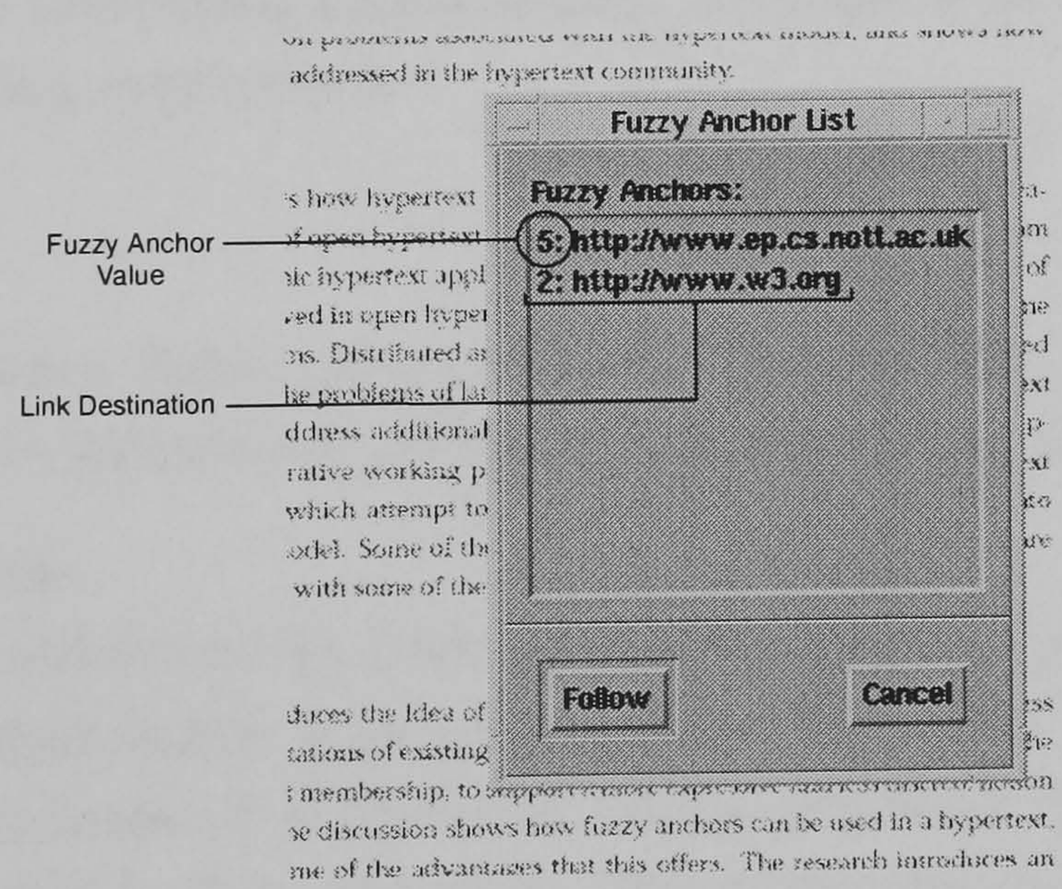


Figure 3.15: Selecting anchors in HIPPO

- *Add Anchor*

  Once a user has created a new anchor (see later), the author must define the

pattern used to represent the anchor. This dictates which words and regions contribute to the anchor, and their corresponding fuzzy membership values. The user defines this *contour map* by using a cursor to *paint* on the document. At any time, the user can alter the current fuzzy value setting using a dialog box which is displayed at the side of the document. An example of a user defining a fuzzy anchor pattern is shown below in figure 3.16.

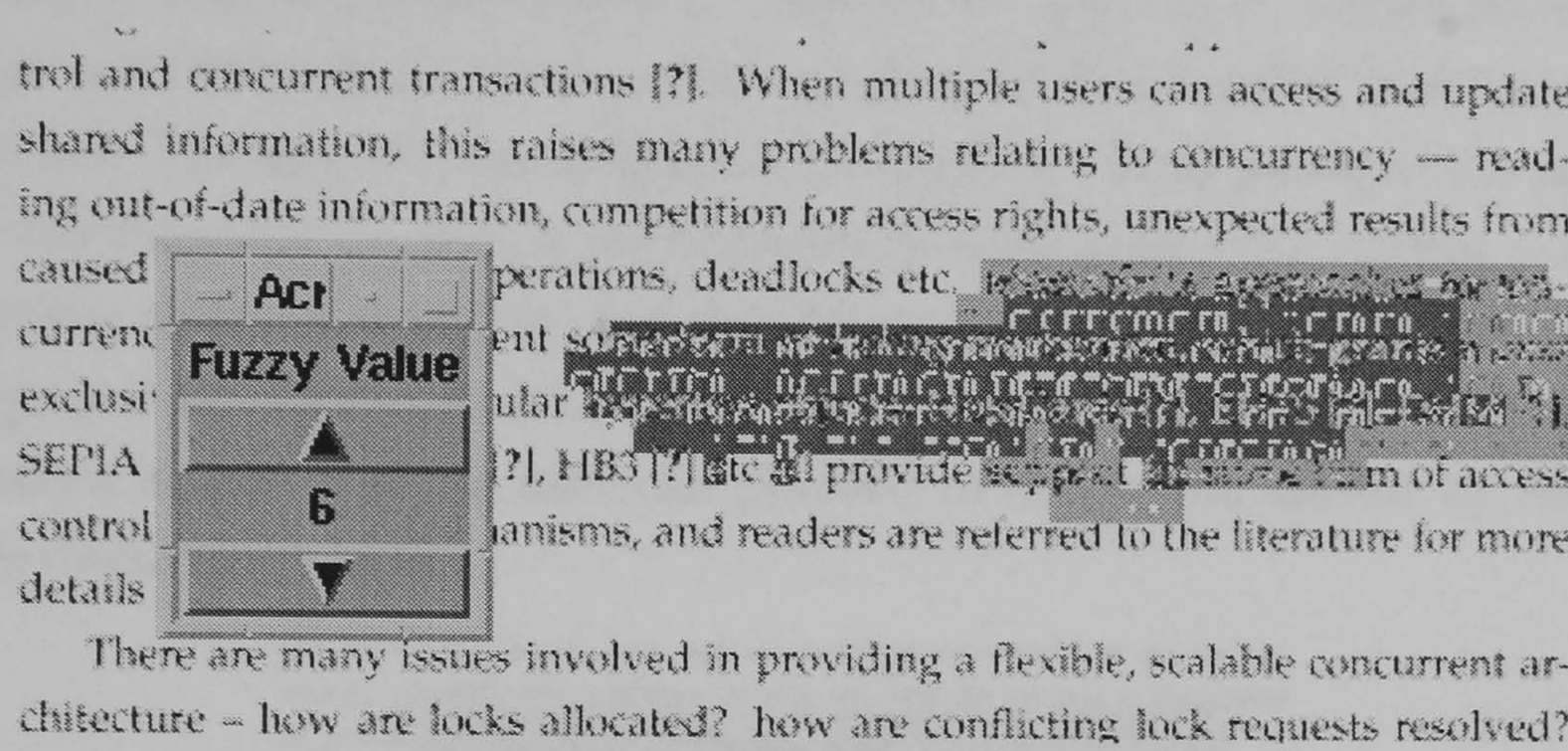


Figure 3.16: Defining new anchors in HIPPO

- *Remove Anchor*
  Once users have defined a fuzzy anchor pattern, they may wish to edit this to remove some of the matrix values. The user does this using the *delete* tool which allows the user to select a region of the document, which then removes this from the underlying anchor matrix. This tool is similar to an *eraser* tool seen in painting applications.

The Fuzzy Menu

The toolbar operations define commonly used tools and the plug-in also adds another *Fuzzy* menu to the browser which includes other useful operations.

- *Load Fuzzy Data*
  The idea of *anchorbases* has been discussed earlier in the chapter, which aim to group related anchor definitions together. This allows the author to define several representations of the same anchor, and for anchors to be treated as first-class objects in their own right. This menu operation allows the user to replace the current anchorbase with a new anchor file, and prompts the user with a file selection dialog box. The application then loads the file, and passes the data to the parser to build the internal data structures.

- *Save Fuzzy Data*
  Similarly, once the user has made any additions or changes to a set of anchor

definitions, these new anchor definitions can be written out to a new file. This allows an existing anchorbase to be updated, or a new, amended anchorbase to be created.

- *New Anchor*
  The earlier discussion of toolbar operations explained how a user can define a fuzzy anchor pattern using a *"paintbrush"* style tool. However, before the user can define these matrix values, they must first create a new instance of an anchor. The application displays a dialog box which prompts the user for all of the information which is necessary to fully define the anchor – link destination, matrix resolution, and the range of fuzzy values which are supported (figure 3.17). Once the details have been entered, the application creates a new anchor object, and the user can proceed to define the anchor by painting on the document canvas.

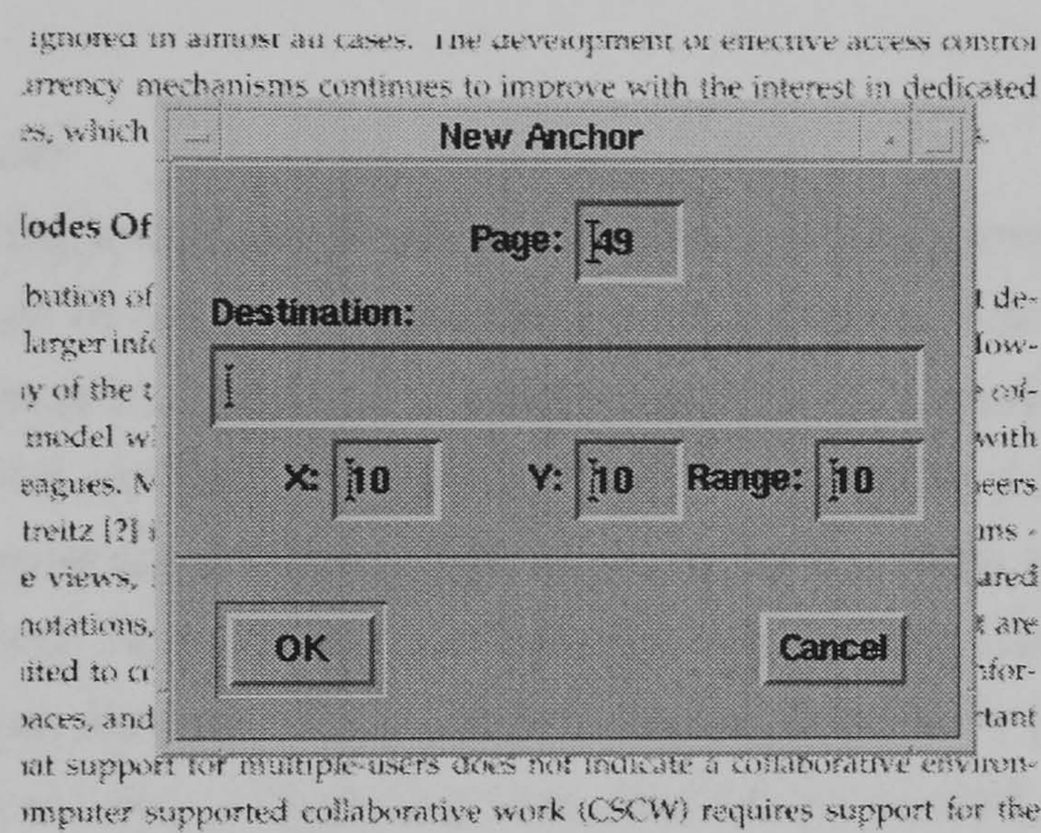


Figure 3.17: Creating a new anchor in HIPPO

- *Select Anchor*
  A document can contain many anchor definitions, many of which may overlap and occupy the same region of the page. If users wish to edit an anchor definition in any way – either to add new values, or to change existing values – then they must select the appropriate anchor before making any changes. This operation asks the user to select a point on the page, and then returns a list of matching anchors. An anchor can then be selected from the list, and this becomes the current selection.

- *Anchor Parameters*
  Each anchor definition contains a set of matrix values, along with a number of other attributes (range, resolution etc). This menu operation displays the attribute values for the currently selected anchor, and allows the user to edit

these (figure 3.18).

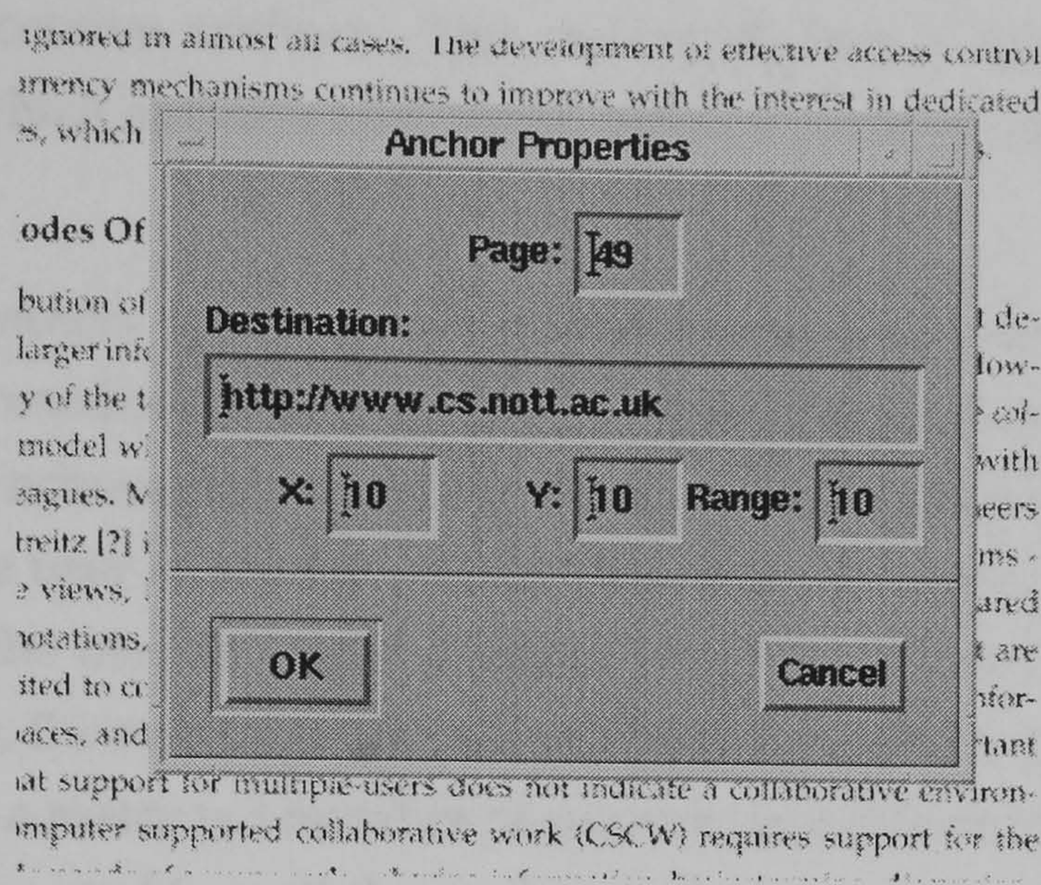


Figure 3.18: Editing anchor details

- *Display Anchors*
  The current implementation adopts a *resource-based* view of hypertext, and requires the user to *query* the application for anchors. For example, a user may show an interest in a particular piece of text, and can select a region using the tools described earlier. The application then returns a list of anchors which have some presence at this point, along with the corresponding fuzzy value for each anchor. The user then selects the appropriate anchor from the list before traversing the link. This can be a useful approach which encourages the user to focus on the underlying text, and assumes that a hypertext can have many anchors defined for any given node. However, it can sometimes be useful for the user to have a graphical indication of any anchors, which they can select directly. This tool displays each anchor using colour to represent fuzzy membership values, and paints over the top of the document. The tool uses an *exclusive-or* mode of painting so that the user can still read the document through the anchors. This tool has limited value if a node has many overlapping anchors, because it can be difficult to represent multiple anchors at any one place. Similarly, this tool has a restricted palette of colours, and the user may prefer to use the *anchor map* tool described later.

- *Change Threshold*
  Each page of a document may have many anchors defined over its surface, indeed, a highly interconnected hypertext is actively encouraged. However, this can be confusing if users are presented with many anchors. The application supports a *thresholding* function so that all anchor values below a certain limit are removed from view. In this way, the user is only presented with anchors which are strongly defined, with high fuzzy values. This is meant to discard

some of the less important anchor definitions, and also to reduce the overlap between successive anchors. The current threshold can be changed using a dialog box which appears next to the main document window.

- *Anchor Map*
  The *display anchors* option paints each fuzzy anchor over the top of the document, and provides the user with a direct representation of each anchor. The application encourages a *query-based* approach for selecting anchors, it can be useful for the user to see the precise definition of anchors. However, it can be difficult for the user to read the underlying text when using the display tool. Furthermore, the nature of the X11 windowing system means that the display tool only has a restricted number of colours with which to present each anchor. This can cause problems if the anchor includes a wide range of values, as the tool soon exhausts the available colours. This *anchor map* tool avoids these problems by creating a new window, which is used to display fuzzy anchors. The window has a private colour map, and has complete access to the entire palette of available colours. Also, the anchor map does not interfere with the main document window, so that users can still read the node contents easily.

- *Reusable Patterns*
  As authors begin to use fuzzy anchors in a hypertext application, they can find that certain patterns and shapes become more popular than others. Authors may find that a particular anchor shape is useful for expressing fuzzy anchors, and that they find themselves using it in a wide variety of situations. Different anchor shapes suggest slightly different semantics, and are more applicable to different content types. For example, plain text is a very linear representation, which is based around lines which scan across the page. Any fuzzy anchors which are used with this form of node content tend to have a symmetrical, rectangular nature (see figure 3.19). A graphical image does not have the same limitations for fuzzy anchors, and an author may find more complex patterns more useful – circles, ellipses etc (figure ?).

  The *patterns* option presents the user with a number of reusable patterns and shapes to define common fuzzy anchors. The user selects the dimensions of the pattern by dragging a bounding box across the page. This rectangular region is used to delimit the pattern – for example, an author could find it useful to select a rectangular region, which then generates an anchor with a blend of fuzzy membership values, which increase as they approach the centre of the rectangle. This could model the common case of a piece of prose which introduces a concept, before describing the most important points of the concept. The current implementation offers a limited number of shapes, although future work could extend this further (Chapter 7).

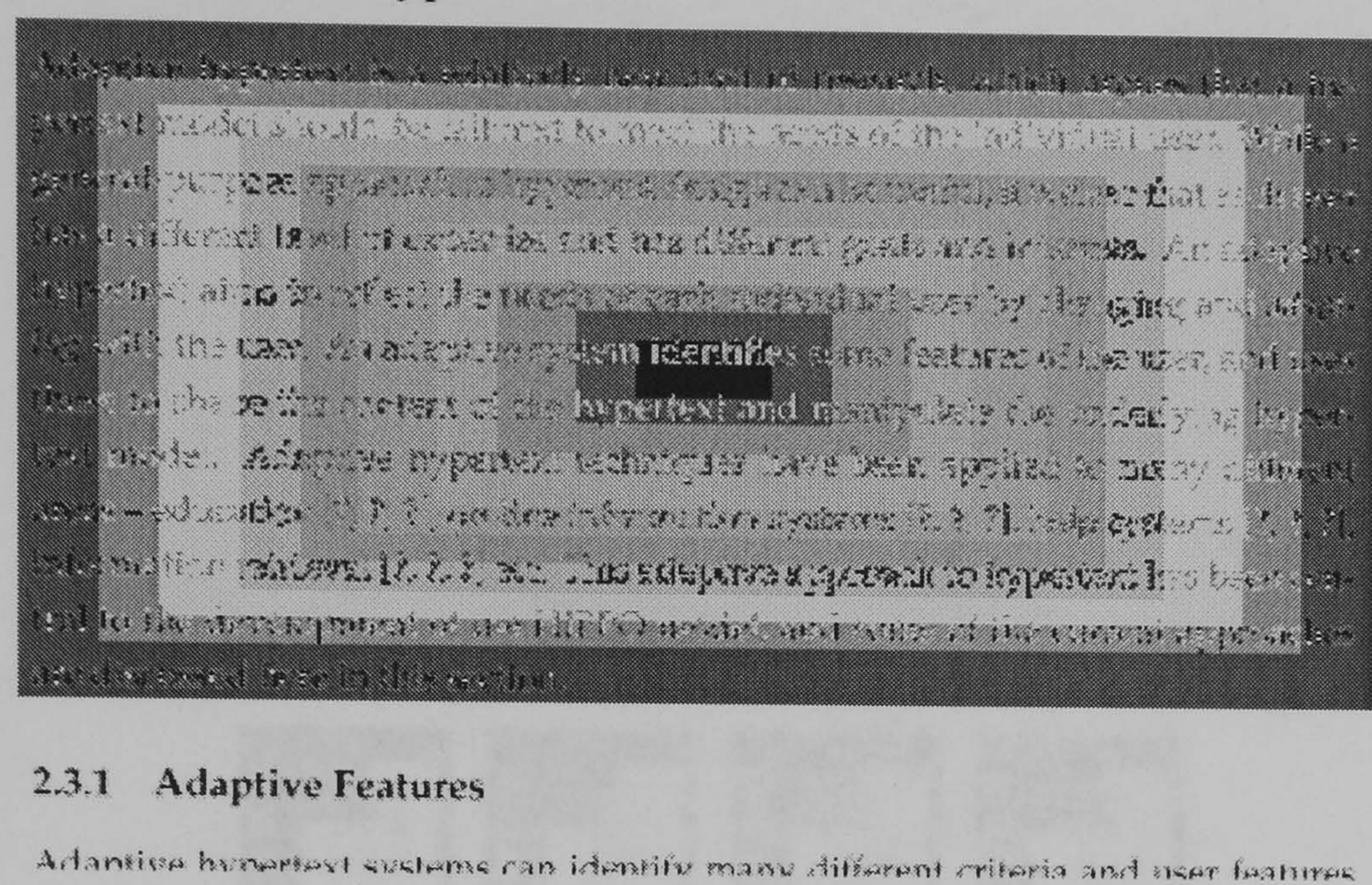


Figure 3.19: Reusable anchor patterns in HIPPO

3.6.6 Adaptive Server

Fuzzy anchors have been developed to provide a more ambiguous, flowing definition than conventional hypertext anchors, which accept the uncertainty involved in an anchor definition. An anchor concept cannot be encapsulated by a single, rigid span of text, but is an emergent idea which grows to a central focus before fading away. It may be unreasonable to expect the author to identify the ideal anchor. The true nature of a hypertext anchor cannot be ascertained until the hypertext has been made available to the user community. Only then can the author observe how users interact with the hypertext, and identify the accuracy of an anchor. The earlier discussion explained the need for more adaptive modelling techniques, and these have been implemented in an *adaptive server*. This section discusses the development of this server, and shows how this has been implemented in the current prototype application.

The purpose of adaptive modelling is to provide feedback from the user, which can then be used to modify and adapt the anchor definitions in the hypertext. However, this form of adaptation seemed to be a natural area to expand to a larger user community. The quality of the adaptive modelling is very dependent on the nature of feedback which is retrieved from the user. Therefore, it seemed advantageous to involve as many users as possible in this feedback loop, to gain a representative view of each anchor. For this reason, the adaptive implementation was collected into a centralised server, which receives information from all user clients (figure 3.20).

Each time a user selects a fuzzy anchor, the user client informs the server of the

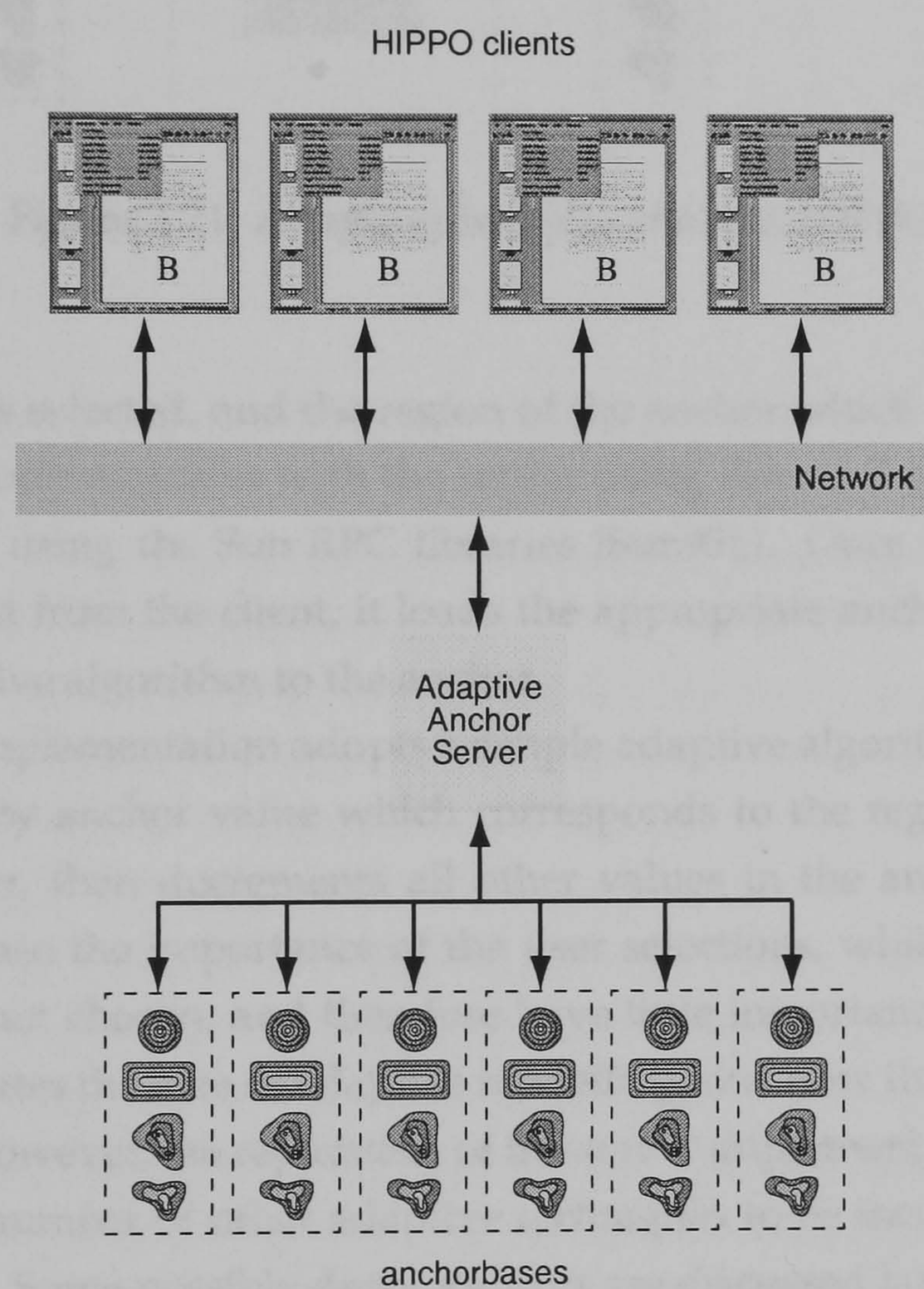


Figure 3.20: Remote adaptive server

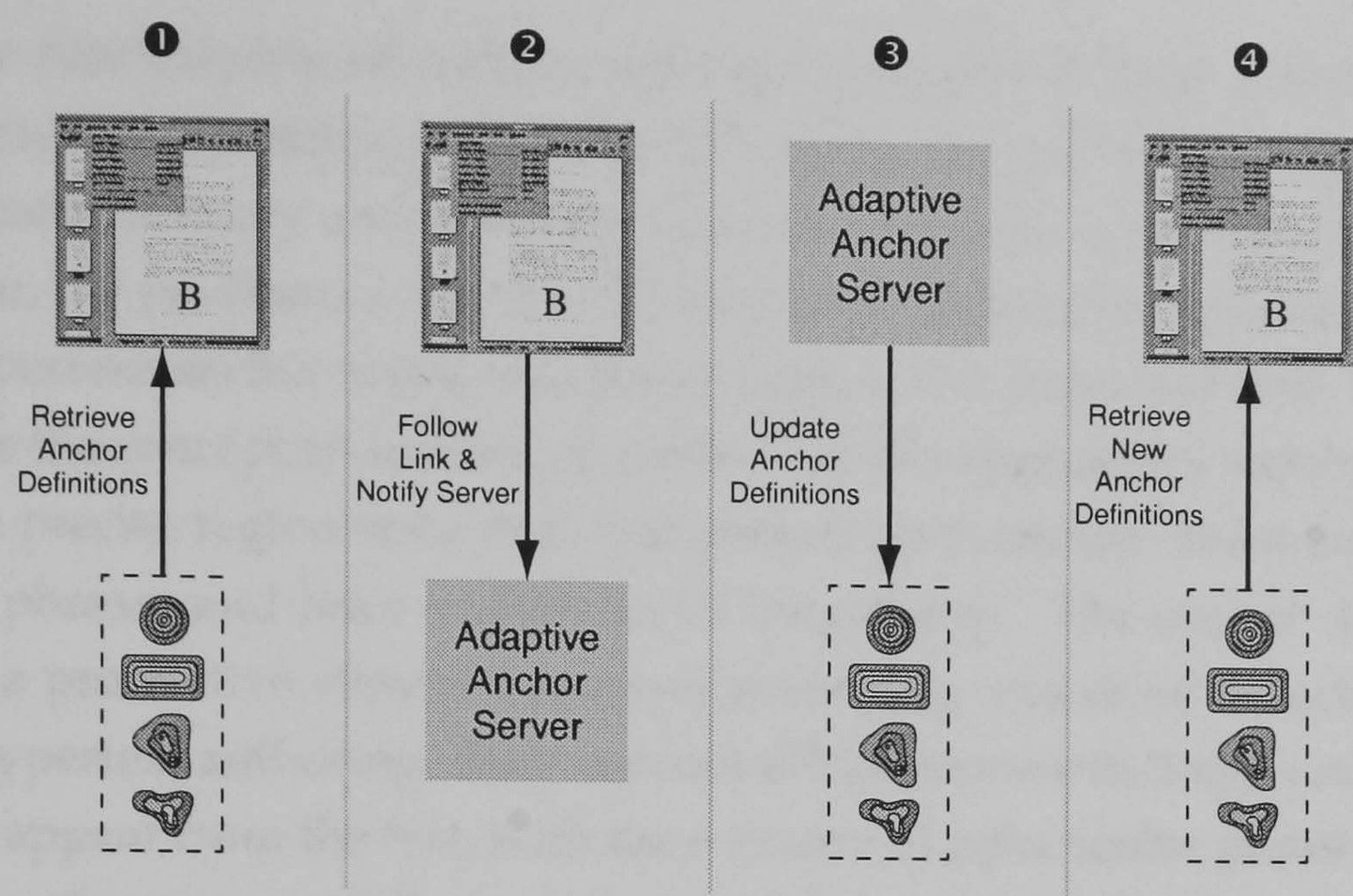


Figure 3.21: Adapting fuzzy anchors in HIPPO

anchor which was selected, and the region of the anchor which was clicked by the user. The client communicates with the server using *Remote Procedure Calls*, which are implemented using the Sun RPC libraries [Sun95a]. Once the server receives the packaged data from the client, it loads the appropriate anchor definition, then applies the adaptive algorithm to the anchor.

The current implementation adopts a simple adaptive algorithm – the server increments the fuzzy anchor value which corresponds to the region which was selected by the user, then decrements all other values in the anchor matrix. This attempts to increase the importance of the user selections, while reducing any regions which are not chosen, and therefore have little importance. This simple approach demonstrates the idea of adaptive modelling, and how this can be applied to fuzzy anchors. However, the separation of the server implementation from the user clients, allows a number of other adaptive techniques to be incorporated in future implementations. Some possible developments are discussed later in the chapter.

3.7 Summary

This chapter has attempted to identify some of the problems with current approaches to the hypertext anchor and anchoring mechanisms. The majority of hypertext applications implement the anchor as a simple data attribute, associated with the hypertext link. The development of open hypertext systems has led some designers to extend the notion of the anchor, and to implement the anchor as a first-class object which can be manipulated independently from other hypertext objects [KL91]. This approach has a number advantages, as the specific addressing mechanisms employed by anchors can be abstracted from the other hypertext linking mecha-

nisms. The functionality of anchors can be developed further, without affecting other hypertext abstractions, and an anchor can be shared between multiple links. However, contemporary anchoring mechanisms still have a number of limitations, in particular, the problem of *over-specific addressing mechanisms* and *static anchoring*.

The hypertext anchor represents the endpoint of a hypertext link, and aims to encapsulate the concept at each end of a link. Current approaches require the author to specify a precise region of content to represent each anchor. The anchor contains a word or phrase, and have a clear set of boundaries. The author does not feel that this is a productive view of hypertext anchoring, and does not reflect the true nature of hypertext authoring. Each concept which represents a link endpoint, does not simply appear from the text, with the mention of a particular phrase. A concept *emerges* from the prose, building on previous ideas, and reinforcing other concepts. The semantics of a node are much more flowing and ambiguous, and approaches to anchoring should provide support for this *fuzziness* rather than forcing the author to make unnatural choices. This chapter introduced the idea of the *fuzzy anchor* to help address this problem. The fuzzy anchor uses fuzzy membership values to reflect the contribution of each piece of text, image etc. A fuzzy anchor grows in strength as the concept becomes more prominent, and builds to a central focus, before fading away once the concept has been explored.

The static nature of hypertext anchors was also criticised, and it was argued that anchor definitions should reflect the needs of the users. Fuzzy anchors seem to be a natural medium for this form of adaptive, and an approach to adaptive anchor modelling was discussed. This allows the anchor patterns to shift and grow in response to the user browsing patterns. This should result in a more responsive form of anchoring, which can develop independently to approach the optimal definition. The chapter also introduces the idea of *anchorbases* which aim to collect related anchors together into a single set. The author can then provide multiple representations of each anchor, to suit particular user groups. Anchorbases have also been used to provide a more accurate, fine-grained form of adaptation, by limiting adaptive changes to a single, group-shared anchorbase.

Finally, a prototype implementation has been discussed, which supports this idea of fuzzy anchoring, and provides some adaptive support. The implementation is based on the Acrobat PDF browsing software, and extends this to support fuzzy anchors. A physical, *"painting"* metaphor was used to represent anchors, which are defined using different shades and hues of colour. The application provides authors and users with a number of tools and utilities to help create and manage fuzzy anchors. An adaptive server has also been implemented which receives feedback from user clients, and uses this to modify the anchor values. The prototype implementation has offered one approach to supporting fuzzy anchors, but has also raised a number of design issues. The development of the implementation has also sug-

gested a number of directions for future research, which are outlined in chapter 7. This work is also summarised in [New97b].

Chapter 4

Building Adaptive Trees Using Linkbases

The previous chapter showed how the hypertext anchor could be developed as a separate abstraction, independent of the conventional view of nodes and links. This separation promotes an open approach to hypertext, and allows the anchoring mechanism to be maintained and extended independently from the rest of the hypertext model. The concept of *fuzzy anchors* was introduced which offered a flexible addressing system for defining anchors. However, more importantly, it was shown how these fuzzy anchors could be used to implement adaptive modelling techniques. An application could incorporate feedback from the user to help modify and change anchor definitions, so that hypertext anchors become a more *living* abstraction which grow with the user.

This chapter applies many of the same techniques to the concept of hypertext linking. Many open systems have identified the link as a defining feature of a hypertext environment, and have attempted to abstract and separate this from the rest of the hypertext model. Many systems separate the hypertext linking information from the underlying node content, and introduce the idea of *linkbases*. These promote a more open and loosely-coupled hypertext system, which allow links to be maintained and updated independently of other applications. Users can then select sets of links which are tailored to a particular domain, or can use links with external, non-native data. However, current approaches to hypertext linking that use linkbases, do not fully develop this abstraction. Linkbases are currently viewed as individual, self-contained objects which are isolated from other sets of links. The relationships between different linkbases are ignored, and open systems do not explore the ways in which linkbases can support and reinforce each other.

This chapter develops the linkbase idea further, and shows how linkbases can be used as a building block for constructing more complex linkbase structures. The idea of *linkbase inheritance* is introduced to model more complex hierarchies and

dependencies, referred to here as *linkbase trees*. This approach to linkbases aims to provide an additional level of abstraction which promotes link re-use and sharing of existing linkbases. This model then introduces some adaptive techniques to show how linkbase trees can automatically develop and *grow* in response to the users' browsing patterns. Confidence values are incorporated into the inheritance model, to capture the idea of *weighted inheritance relationships* – these are then used to support a form of adaptive hierarchy. Finally, the prototype of the HIPPO system which was explored in Chapter 2, is extended to support these ideas of adaptive linkbase trees.

4.1 Separate Link Structure Using Linkbases

The hypertext link is fundamental to all hypertext systems, and provides the basis for non-linear writings (see Chapter 1). The *hyperlink* embodies the very idea of hypertext, and enables the author to express relationships and connections between disparate units of information. Chapter 1 outlined some of the different approaches which have been taken by hypertext researchers. Many early systems such as HyperTIES [Shn87], WE [SWF87] and ZOG [AM84b] provided a very simple notion of links, with limited anchoring support, and were often limited to uni-directional links. Other hypertext applications developed more powerful linking mechanisms, ranging from directional linking schemes and *n*-ary, multi-way links through to annotated links and *typed* linking schemes (eg. Intermedia [YHMD88], Neptune [DS86], NLS/Augment [Eng84a], Notecards [HMT87], TEXTNET [Tri86] etc).

While the hypertext link itself has been developed in many ways to help designers and users of hypertexts, researchers have also acknowledged the collective role that links play in a hypertext environment. A hypertext link should be viewed, not simply in isolation, by looking at its individual semantics, but also as a whole, as a part of the larger hypertext. Many hyperlinks reinforce other connections and offer alternative paths through the hypertext. These hyperlinks only have real meaning when used in conjunction with other links, and viewed in the larger context of the entire hypertext.

For example, a hyperlink often assumes that the reader has followed a particular reading path and has a certain knowledge of the subject. Similarly, many links rely on the presence of other links to offer the reader an alternative reading route. Hypertext links rarely exist in isolation; instead they play a more subtle, complex role in a hypertext. The true potential of hypertext and hyperlinking can only be fully realised when the hypertext is viewed as a whole, and all links are viewed in the context of other supporting links.

This collective view of hypertext links was explored in the PIE system [GB80] which arranged links (and other hypertext objects) into layers, called *contexts*. These layers could be combined together to form a single hypertext and could be used to select different sets of links or to implement some form of versioning. The reader could select appropriate layers and combine these together to form a single hypertext. The author could also provide several different sets of nodes and links, each tailored to a particular group of users. Intermedia [YHMD88] also supports a similar idea of *webs* which allow links and anchors to be separated into layers.

The idea of extracting links from the node contents and combining them together into discrete sets has also been developed in the field of open hypertext. Linkbases are a vital part of open hypertext research and have been used to great effect (see Chapter 2 for more information). Many open hypertext environments

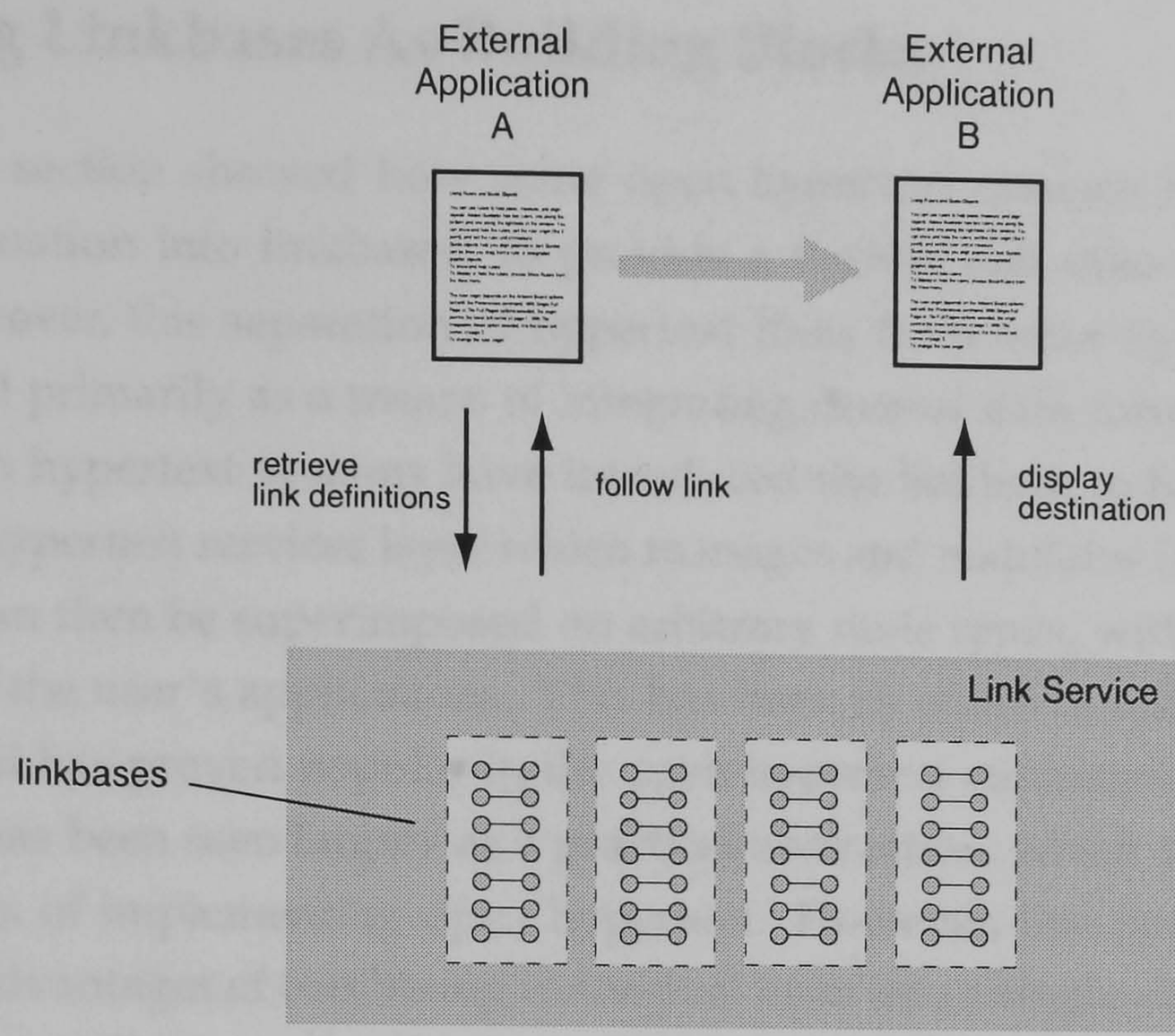


Figure 4.1: Using linkbases in open hypertext applications

address the problem of incorporating hypertext by providing a link services layer. The user's applications and tools sit on top of this linkbase layer, allowing them to access hypertext services. Open systems typically store hypertext links and linking information separately from the node contents – this allows the user's original applications to operate on the raw content, without having to understand hypertext-specific information [FHHD90, Pea89]. These hyperlinks are often stored together in separate files, referred to as *linkbases*, and can then be overlaid on the node contents to offer a seamless hypertext environment (figure 4.1).

The linkbase forms the basis of many open hypertext systems. Linkbases allow users to select the appropriate set of links for their application instead of being forced to adopt a single set which is permanently embedded in the node contents. Indeed, by leaving the original contents untouched, this allows the original applications to access the data without modification. Separate links allow users to access shared data repositories in a collaborative environment or to use read-only media or data which is not owned by the user. Linkbases also provide a way of supporting multiple views on data objects by merging selected link sets together to form a more coherent and complete set of links. A hypertext environment built on the idea of linkbases also allows dynamic link sets to be used so that the hypertext is constructed dynamically for the user, rather than as a static network of nodes.

4.2 Using Linkbases As Building Blocks

The previous section showed how some open hypertext systems have separated linking information into linkbases, to provide a flexible and tailorable hypertext system. However, this separation of hypertext links from other hypertext objects has been used primarily as a means of integrating diverse data formats and applications. Open hypertext systems have introduced the linkbase to help implement the idea of a hypertext services layer which manages and maintains hypertext links. These links can then be superimposed on arbitrary node types, with minimal customisation of the user's applications. This has been an effective approach to open hypertext, and has proved popular in the open hypertext community. In this way, the linkbase has been seen largely as a practical abstraction, which provides a convenient means of implementing open hypertext. However, this approach ignores many of the advantages of combining individual links into a single abstraction. This view of linkbases does not develop the *collective* view of hypertext links.

It is not simply the separation of hypertext structure which is important in a linkbase, but also the way in which links are combined together to form a collective view of hypertext links. Links are no longer considered in isolation, but are instead treated as members of larger sets and structures. The value of a link lies not only in its intrinsic definition, but also in the way it contributes to the larger set of hypertext links. We should be moving away from the idea of the link as a single entity, and should instead view the *linkbase* as the fundamental unit of discourse. Just as isolated links were developed and used to build larger structures, so the linkbase should be used as the basic building block for constructing increasingly complex structures.

The remainder of this chapter explores ways in which linkbases can be used to support a more scalable hypertext model. The author should focus on the linkbase as a fundamental hypertext object, and should concentrate on the many subtle and intricate relationships between these link collections. These inter-relationships should be encouraged and modelled explicitly. Linkbases should be seen as resources to be shared and reused between the hypertext community. The chapter introduces the idea of *linkbase trees* and shows how these promote a model for hypertext linking which scales to support larger hypertext structures. Linkbase trees encourage authors and readers to reuse existing linkbases and to refine and tailor these to meet their specific demands. The chapter also discusses some of the issues arising from this tree model, and explores some of the advantages of this approach.

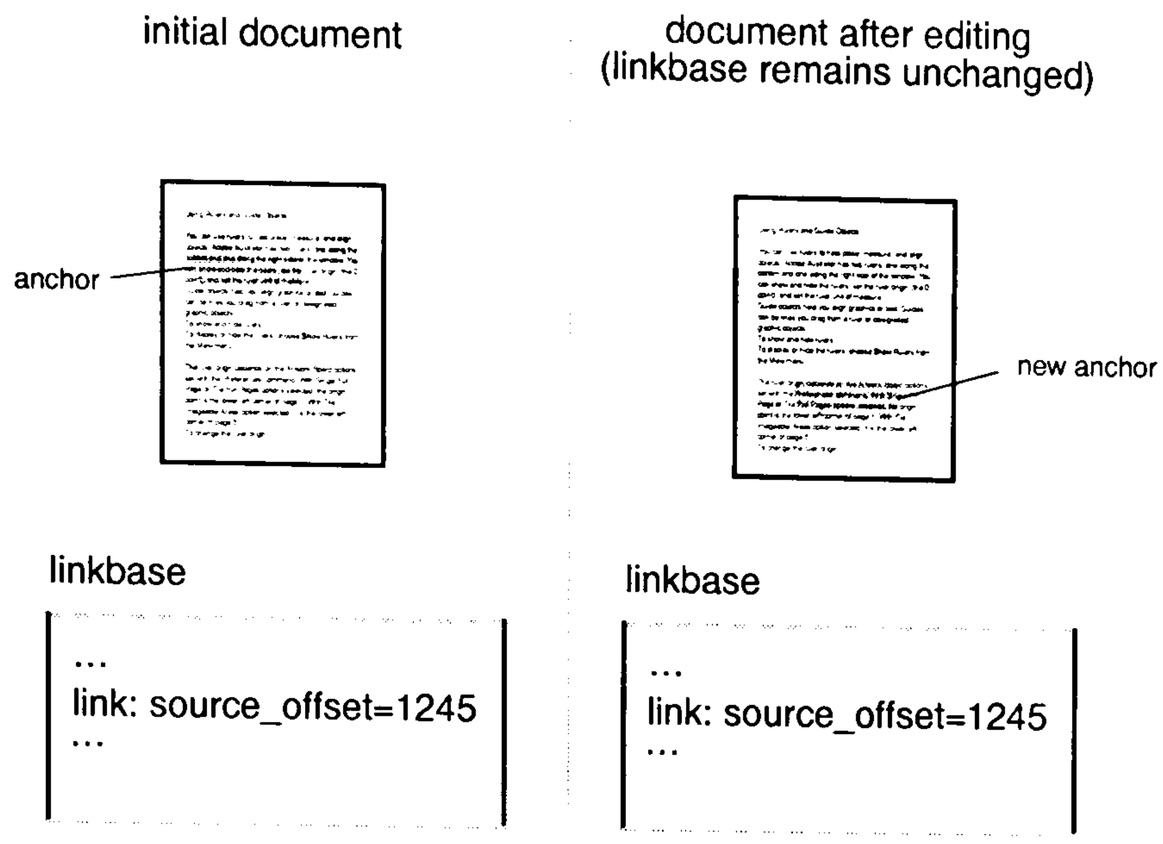


Figure 4.2: Update problems with linkbases

4.3 Limitations Of The Linkbase Approach

Applications which separate hypertext structure can experience some limitations compared to more integrated linking approaches. For example, it can be difficult to maintain link sets which are stored separately from the node contents, and are not managed by native applications. Changes which are made to the underlying node data must be reflected in the hypertext link structure – anchor positions may change; some links may reference nodes which no longer exist; a hypertext link may no longer make sense in the updated hypertext etc (see figure 4.2). Open hypertext systems have acknowledged many of these problems, and must provide additional support for ensuring data integrity etc. A hypertext application which adopts a linkbase approach must agree on some common format for specifying link entries, which is able to express the diverse semantics of hypertext links. The hypertext system must also provide some means of managing linkbases, controlling updates, merging link sets etc. However, these are all largely implementation problems, which many OHSs have attempted to address. This section focuses on the abstract problem of using linkbases, and how the user can maintain and reuse large numbers of links and link collections.

A linkbase represents a carefully crafted set of links, designed to structure a particular subject area or set of information. The creation of a linkbase involves a full understanding of the subject area and an awareness of the particular demands that hypertext makes on the user. When readers wish to explore a particular subject area, they should not be forced to conjure up hypertext links themselves or blindly use some kind of search engine. A linkbase may have been carefully tailored by some expert, and this expertise should be shared and reused by other users. A linkbase

represents very rich semantics and users should be able to incorporate these links into their hypertext. However, linkbases should not be considered in isolation. A linkbase is a rich hypertext resource which should be shared and reused by the user community. Each linkbase encapsulates information about a specialised subject area, or provides a collection of links targeted to a particular type of user. A linkbase can be useful in many situations, and a user will often combine several linkbases together to represent a more complete and rounded set of hypertext links.

This raises the problem of managing large collections of linkbases. How does a user decide which set of linkbases to use? Which linkbases should be used for a particular subject? Some linkbases will fit well with certain other linkbases, and some link collections will combine more naturally with particular link sets. A modern hypertext environment should provide ways of expressing these subtle interrelationships which exist between different linkbases. Users should be guided as to which linkbases to use and how they are incorporated into the hypertext. If a user finds a particular link collection to be of value, then perhaps other related linkbases could be suggested to the reader. When authors wish to build additional linkbases, they should be able to build on existing link collections, rather than starting from scratch. The role of an author should be, not only to provide sets of link definitions, but also to show how linkbases can be used together. An author should be able to show which linkbases can be used together, and how they should be combined. Linkbases should encourage reuse and promote a shared environment and users should be able to refine and tailor particular link collections to meet their own demands.

4.4 Linkbase Inheritance Trees

This method of using linkbases – reusing and sharing link collections; modelling relationships between linkbases; refining and tailoring link definitions – has much in common with the goals of Object-Oriented software design. O-O research looks at ways of constructing environments from components; reusing existing designs and implementations and expressing relationships between components. One of the key contributions of O-O work is the concept of *inheritance* which arranges components into hierarchies, to express relationships and commonality among objects [Lis87, Sny86b]. Booch [Boo94] describes inheritance as "a relationship among classes, wherein one class shares the structure or behaviour defined in one (single inheritance) or more (multiple inheritance) other classes". Inheritance allows classes to be shared and reused, and components to be extended and tailored to meet the specific needs of an application (see figure 4.3). These are also useful for managing and developing hypertext linkbases, so that dependencies between sets of links can be made explicit.

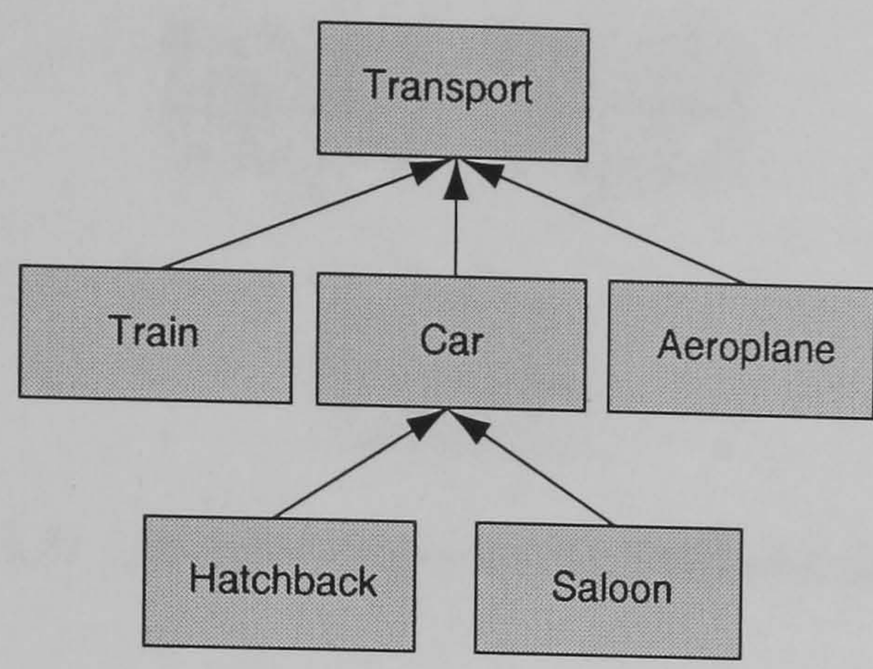


Figure 4.3: Object-Oriented inheritance hierarchies

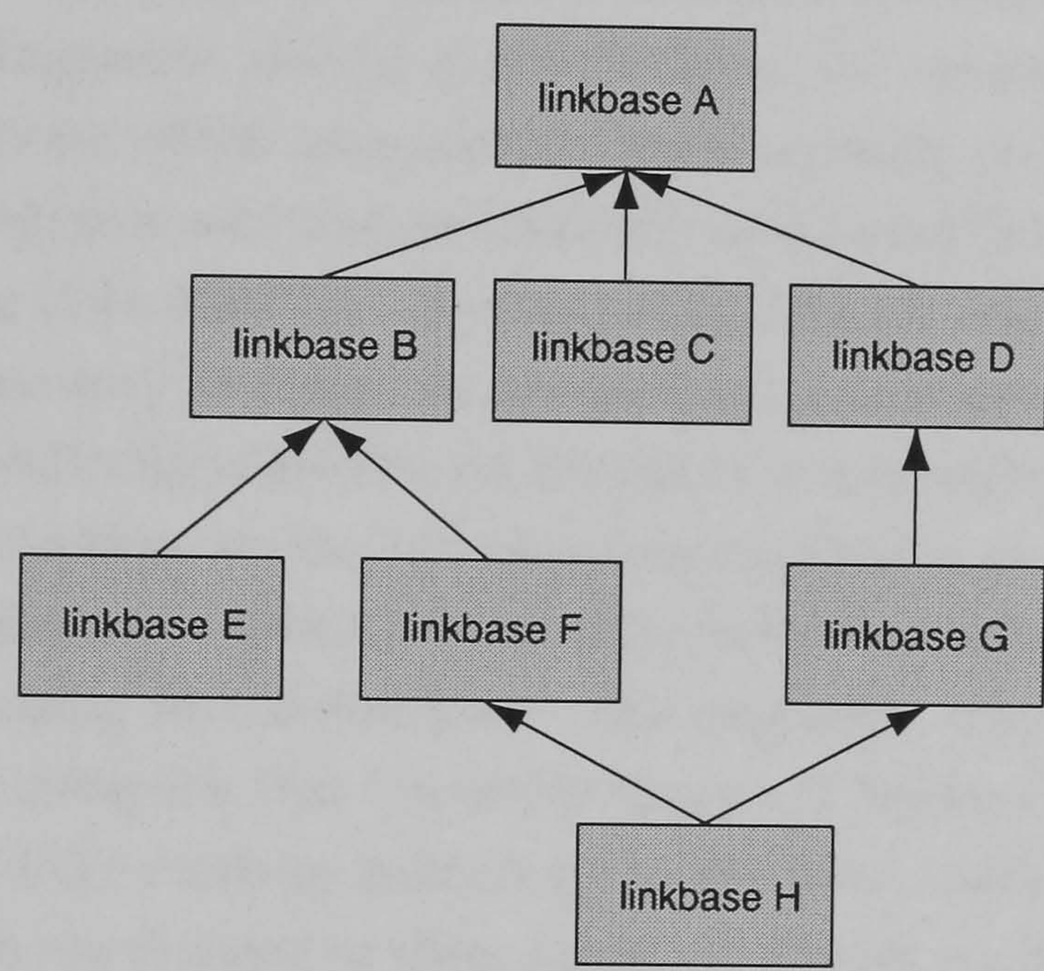


Figure 4.4: Linkbase inheritance trees

Linkbases can also be treated as components which can be arranged into some form of inheritance hierarchy. Relationships between link collections can be modelled as inheritance relationships, so that linkbases are shared and reused throughout the hierarchy. A linkbase no longer exists as a single, autonomous collection of links, but plays a role in a larger inheritance hierarchy. A linkbase can be defined not simply as a collection of individual links, but also as the combination of several existing linkbases. A linkbase that is derived in this way, encapsulates the contents of existing linkbases, but can also include additional link entries which are considered important by the author (figure 4.4). This linkbase hierarchy allows the author to express the relationships between linkbases, and show how these can be used to build larger, more accurate linkbases.

For example, consider the construction of an engineering linkbase which is to be used by engineers working on a large construction project. A large building project such as this requires the pooling of information from many diverse disciplines such

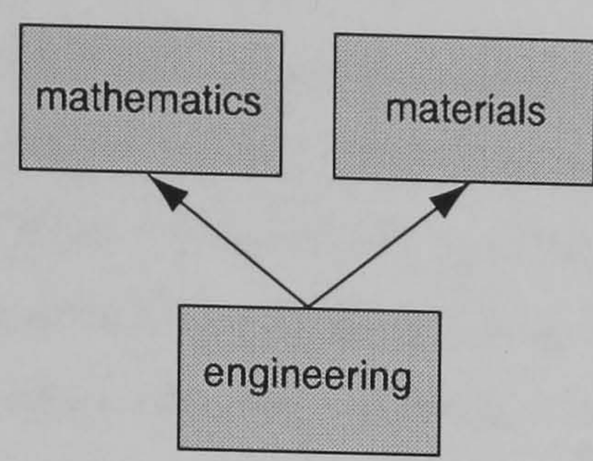


Figure 4.5: A simple engineering linkbase hierarchy

as mathematics, materials etc. However, a hypertext framework should encourage authors to reuse existing link collections, rather than creating the entire hypertext from scratch. Engineers should aim to build on the expertise of other authors, reusing link collections which have already been carefully constructed. Linkbases can be combined together, and augmented with additional links which are specific to the project. Some links may be refined or tailored by the engineering team, while other link definitions may be excluded altogether. Furthermore, the ways in which these existing link collections are reused should be made explicit so that future engineering projects can benefit from the experiences of the team.

An inheritance tree can express relationships between linkbases, and offer a useful approach to reusing link definitions. The engineers may decide to combine multiple linkbases using the tree hierarchy shown in figure 4.5. The engineering linkbase is derived from existing linkbases – in this case, mathematics and materials. The links which are defined in these parent linkbases are inherited and shared by the new linkbase, as if they were actually defined in the engineering linkbase itself [1].

This engineering linkbase which derived from existing collections can be extended with any additional links which are deemed appropriate to the subject area. Alternatively, the derived linkbase can *specialise* links by overriding and updating links which are defined higher up the tree. For example, the author may decide that a general purpose link which is defined in the mathematics linkbase should be revised and overridden with a more engineering-specific definition. Similarly, the inheritance model can support *exclusion* which is seen in some O-O languages [Sny86a]. Links which are defined in parent sets can be excluded from derived linkbases which appear further down the hierarchy. This allows those links which are considered inappropriate or irrelevant to the project to be removed from the final linkbase. The IGD system [Fei90] also incorporates some simple inheritance

[1]This scenario gives an example of multiple inheritance in which a derived class can inherit from more than one base class. O-O academics argue over the relative merits of single and multiple inheritance, although multiple inheritance seems to be a more natural choice when applied to sets of hypertext links. Multiple links for a given subject do not create contradictions, whereas multiple implementations of the same method in an object can provide ambiguities and conflicts.

ideas into the linking model, but this is used to filter links and navigation through the hypertext. Similarly, some systems arrange link *types* into a hierarchy to enforce a rigid link typing system. Other systems such as Neptune [DS86] and HyperPro [ON94] use link layers and *contexts* to support some form of hypertext versioning. These approaches all differ from the way inheritance has been used here, which allow relationships between linkbases to be modelled explicitly.

4.4.1 Advantages Of Linkbase Trees

This approach to managing linkbases using inheritance hierarchies has many advantages over the conventional approach to linkbases. Current hypertext applications view a linkbase simply as a collection of links, and do not attempt to develop the linkbase as a primary, independent object in the hypertext model. Linkbase trees allow authors to express the many relationships and dependencies between link collections, which are ignored in other systems. Link collections are expressed in terms of existing linkbases, and the relationships between them are made explicit. Authors can suggest ways in which linkbases can be combined most usefully, for future users and authors to use. Linkbases which are based on tree hierarchies also reduce the redundancy and maintenance overhead associated with conventional linkbase models. An inheritance relationship represents a *reference* to an existing linkbase, instead of forcing an author to make copies of the appropriate link definitions. Any changes which are made to a linkbase permeate throughout the inheritance hierarchy, and are automatically reflected in linkbases further down the tree. The authors of the parent linkbases are free to add, remove links, or update links, and these will be reflected further down the inheritance hierarchy, without needing to notify authors of the derived linkbases.

The application of inheritance to hypertext linkbases provides an intelligent way of reusing link contexts. Link inheritance promotes the idea of *programming-by-extension* – linkbases are constructed by reusing and specialising existing linkbases. An author is encouraged to think of a linkbase in terms of the wider hypertext and how it relates to existing link collections. Linkbases can be targeted towards specialist areas or particular type of users, in the knowledge that users will be augmenting each linkbase with definitions from other collections. A linkbase no longer has to provide a complete and exhaustive set of link definitions, but is just one building block in a larger hierarchy.

Authors are not limited to reusing individual linkbases, but can reuse entire sections of an inheritance tree. For example, if we expand the scenario above, so that an author wishes to construct a linkbase which caters for the design tasks of bridge construction. The links defined in the engineering linkbase are clearly of great value to the design staff, and should be included in the new design linkbase. Also, the design team may require some aspects of architecture and building design

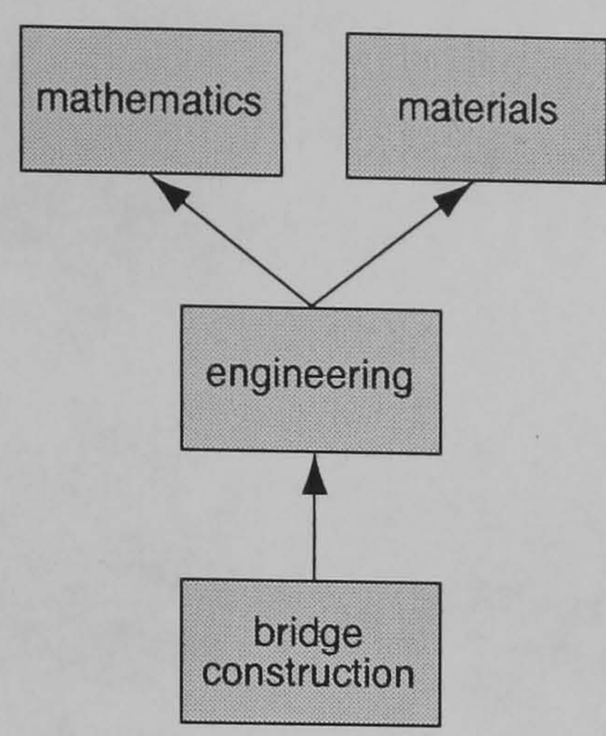


Figure 4.6: Deriving a new engineering linkbase

– this could also be incorporated into the inheritance tree (figure 4.6). However, the value of the inheritance model is that the author can inherit from the engineering linkbase, unaware that this engineering linkbase actually inherits from two further link sets. Authors can ignore the complexities of the inheritance tree further up the hierarchy, and can view the engineering linkbase as a single abstraction. In this way, authors are not only reusing links and linkbases, but are also sharing and reusing entire trees of linkbases.

4.4.2 Extending Trees Using Linkbase Visibility

The ideas of specialisation and exclusion have been incorporated into the linkbase inheritance model, to allow links to be overridden or removed from derived nodes. Many OO models also introduce the idea of *access control* to modify the semantics of the inheritance hierarchy. Access control allows the access and inheritance of certain properties of each node to be controlled at a finer level of granularity. For example, the C++ programming language [Str91] introduces the idea of *public*, *private* and *protected* protection modes which limit the access of object contents. Private members can only be accessed from within the same class, while public members can be accessed freely from external entities. Protected access however, offers a variation on this model in which only derived classes can access these protected members. Protected members can be accessed by any node which inherits from the class, but are hidden from any classes which appear outside of the inheritance hierarchy (figure 4.7).

These access control mechanisms have obvious benefits in the object world, but collections of hypertext links do not have the same properties as software components. In particular, hypertext links do not have the same notions of state and behaviour etc, and are largely implemented as static, passive node tuples. However these protection controls can be applied in a number of interesting ways which are discussed in more detail below.

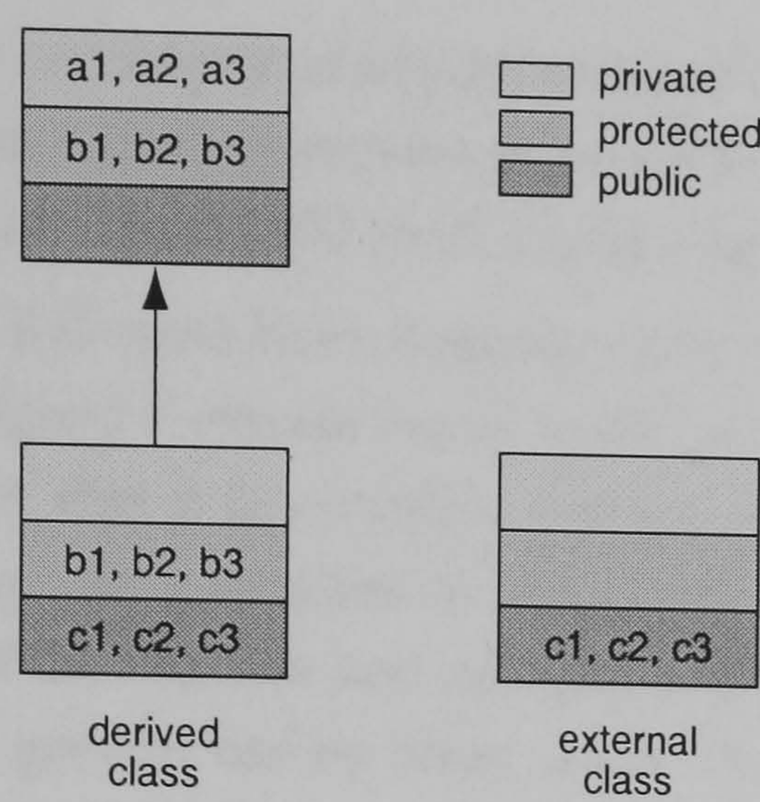


Figure 4.7: Access control in inheritance hierarchies

- A linkbase has been defined simply as a collection of hypertext links, however it can be useful to define additional information in a linkbase. For example, a linkbase could define symbols, authoring information, link ownership or other data members which could be then be reused in the link specifications in the remainder of the linkbase. Protection controls could be used to control access to this information in the hierarchy, or to hide some details from external linkbases etc.

- While the hypertext link is largely viewed as a simple node pairing, many hypertext environments have developed the concept of the link further to include more computational, behavioural aspects. Many hypertext applications allow scripts to be attached to links, which are executed when the link is traversed (see section 2.1.3). The World Wide Web [WWWa] in particular has proved particularly popular, providing computational functionality using CGI scripting and browser extensions such as JavaScript [JS]. The recent success of the Java [GM95] and ActiveX [Act] environments also develop this idea further. If we allow links to incorporate some behavioural aspects, then access control mechanisms can be used to limit access to data values and information used in the link computations. Furthermore, we can use the inheritance semantics to indicate that some of these data members must be overridden when other linkbases derive from this one.

- We could use the more literal definitions of the *public, private* and *protected* keywords to indicate in some way how relevant and important the links in the linkbase are to the subject. For example, a key link which is universally applicable may be given a *public* status, while a less important link which may not be so useful could be assigned a *private* value. While this idea of giving additional semantics to a link is useful, it seems more intuitive and

expressive to assign some form of affinity value to each link. Indeed, this idea of weighting hypertext links to express the confidence in the relationship has been incorporated into the HIPPO model, and is explored in section 4.5.1.

The idea of giving a link some kind of status within the linkbase can be refined. A link which is assigned a private value could be considered so esoteric and specific to a subject, that it is unusable outside of the linkbase in which it is defined. For example, a number of links may be defined by authors (to help them construct the linkbase and navigate through the hypertext), which are not suitable for general use by other users. Any attempt to use these in derived linkbases would not be as useful, and so a private assignment could prevent derived classes from accessing these links. This seems to be a more sensible and limited use of access control, and could be developed further. The current implementation of the HIPPO system does not support this idea of access control, and Chapter 7 suggests future directions for this research.

4.5 Adaptive Trees

The introduction of inheritance relationships into the linkbase model offers a new level of abstraction for constructing hypertexts. Linkbases are no longer considered as simple collections of links, but are viewed as fundamental objects in their own right. These linkbases can be used to build complex hypertexts, and promote a form of reuse and sharing. Users can amend and refine other linkbases, or incorporate other users' link sets into their hypertext. A linkbase hierarchy attempts to express the relationships between different linkbases, and model the dependencies between different link collections. These relationships can be very complex and subtle, and may only become apparent over a period of time. However, the current model requires the author to define a tree hierarchy at the first attempt – a hierarchy which remains for duration of the hypertext. This is a very static and permanent approach which does not seem to reflect the true nature of hypertext authoring.

The previous chapter showed how anchor definitions can also be very complex objects, and place significant demands on the author. The optimum definition of a hypertext anchor is never entirely clear, and can only be ascertained by observing the browsing patterns of the user. In time, a more accurate representation of an anchor becomes clear as users interact with the hypertext. This user feedback can then be used to adapt and modify the original anchor definition. Similarly, the construction of a linkbase hierarchy is also a complex task, and one which should be modified and changed over the duration of the hypertext. An author cannot be expected to build accurate and useful linkbase trees, without first observing the way the user interacts with the hypertext. It seems quite natural to incorporate some level of adaptation into the tree model, so that the hierarchy can respond to

the needs of the users. In this way, linkbase trees can automatically compensate for any inaccuracies or errors which appear during its initial construction.

For example, the previous scenario outlined the development of an engineering linkbase, and showed how this new link collection can be defined in terms of existing linkbases. The author may identify a number of useful *parent* linkbases, and might build a linkbase hierarchy resembling that in figure 4.5. This seems to be a reasonable hierarchy, yet it may transpire that some of the parent linkbases are more significant than others, or that some of the linkbases in the hierarchy remain unused. Conversely, some linkbases that were omitted from the tree may be able to make a useful contribution to the hierarchy. It is important to recognise the problems of constructing linkbase trees, and to show how initial attempts may prove inaccurate. The ideal tree hierarchy would have an emergent behaviour, which only becomes apparent as the users continue to explore and traverse links. The value of a particular linkbase can only be ascertained by observing the browsing patterns of user, and by allowing the tree hierarchy to evolve over time.

This section attempts to incorporate some adaptive modelling into the linkbase tree model. The notion of confidence values are used to augment hypertext links in the HIPPO model, to associate some degree of certainty with each link relation. These confidence values are also incorporated into the inheritance hierarchy to express *weighted inheritance relationships*. These can then be modified using feedback from the user, so that the tree *automatically* responds to the needs of the user.

4.5.1 Weighted Links

Chapter 1 outlined some different approaches to hypertext linking (typing, annotations etc) which attempt to introduce additional semantics into the linking scheme (see Section 1.1.2). These approaches provide more expressive hyperlinks which hope to better capture the true nature of a relationship. While this has been successful in many systems it can often be more useful to model, not only the semantics of a link, but also some measure of *certainty* in these link semantics. An author may create many links and relationships in a hypertext, yet some connections may be more useful than others. Some links may represent very strong relationships, while others indicate a looser coupling.

This idea of associating confidence values with hyperlinks has been explored by a number of researches. Furnas introduced the idea of fish-eye views [Fur86] which attached *affinity* weightings to links, and used these to aid navigation. Pausch *et al.* describe a system in which users are presented with node popularity weightings which can be used to help the user decide which links to explore [PD90]. The SaTellite system [PT90] uses link affinity values to compute views and dynamically lay out the hypertext, and Ziv *et al* [ZR97] uses Bayesian networks to model uncertainty in software systems. Also, Frisse used hypertext techniques in the medical field in

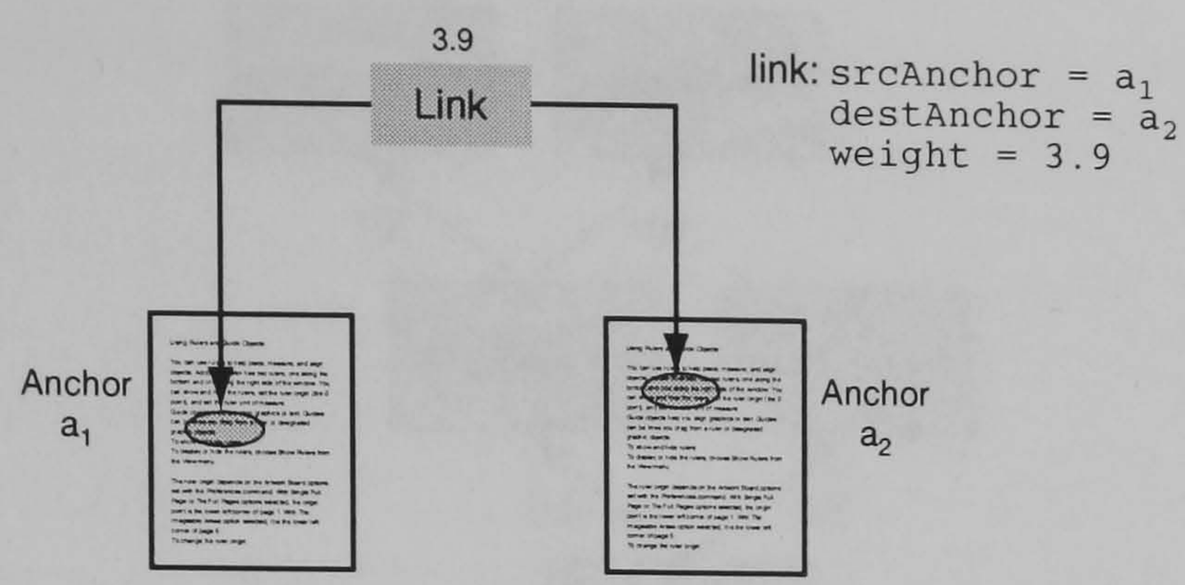


Figure 4.8: Weighted hypertext links

his Hypertext Medical Handbook [Fri87, FC89], which propagated weighting values throughout the hypertext to aid users in the retrieval of useful information.

This idea of introducing weights and confidence values into the hypertext linking system has also proved popular in the fields of information retrieval [Lel92] and adaptive hypertext [Bru96]. Link weightings introduce additional semantics into the hypertext system, which cannot be encapsulated using conventional link typing mechanisms. Weighted links allow authors to express more subtle relationships, not simply the type of relationship but also some measure of the *intensity* of the relation.

The HIPPO model supports this idea of *weighted links* by associating a confidence value with each link definition in the linkbase (see section 4.9.2). This value represents the certainty and importance of a particular hyperlink. The author can then express, not only the nature of a particular relationship, but also the importance that this has in the overall hypertext (figure 4.8). An author can emphasise certain links as being fundamental to the "reader experience", while other links are less important and are only to be followed by more advanced users. Users can filter out links which are less important, while focusing on those links with higher confidence values. Alternatively, a user may take the opposite approach and may be more interested in those links which the author considers obscure or less important. These weighted links also provide a useful means of supporting an adaptive hypertext model, and this approach has been seen in some adaptive hypertext systems. Section 4.7 explains how the HIPPO model combines these weighted links with the fuzzy anchors presented in the previous chapter, and with a form of weighted inheritance hierarchy.

4.5.2 Weighted Inheritance Relationships

Hypertext links can be assigned weightings which can be used to help the user navigate around the hypertext, and to decide which links are most useful. A link is no longer a discrete, static object, but becomes a fuzzy entity which exists in various states of intensity. Weighted links allow an author to express some confidence

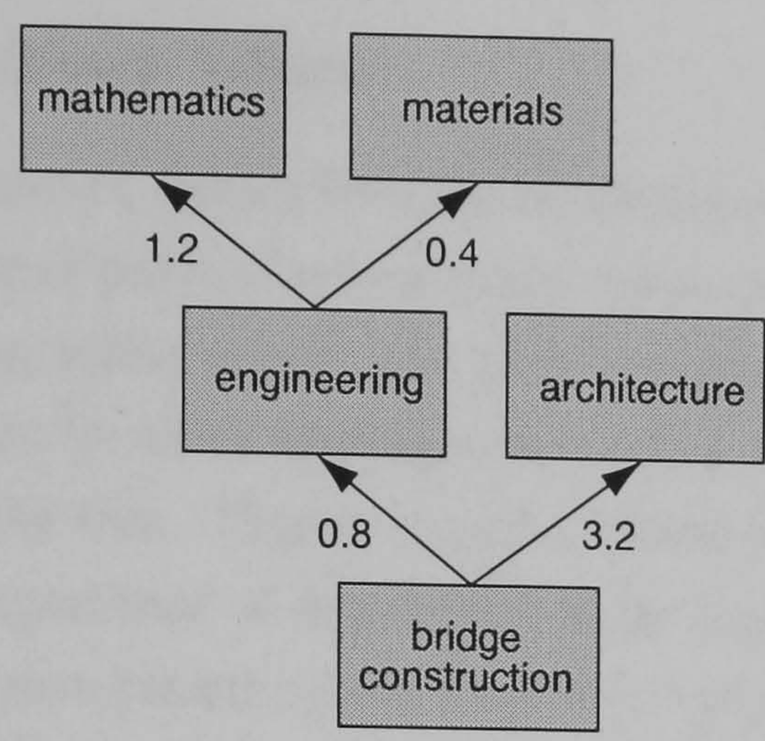


Figure 4.9: Weighted linkbase hierarchy

or certainty in each relation. This approach can also be applied to the inheritance model which has been proposed in this chapter. Linkbases can be arranged into tree hierarchies as before, but instead of treating all linkbases as equal, we can attach some form of fuzzy value to each relationship. An inheritance tree can assign weighted values to each inheritance relationship to indicate a particular importance to particular parent linkbases, or emphasise a particular dependency (figure 4.9). Linkbases which are important to the hierarchy and contain essential link definitions can be emphasised and given increased weightings.

For example, the design team for the bridge construction linkbase described in section 4.4 may feel that *architectural* concerns are of greater importance than conventional engineering constraints. The inheritance dependencies for the architectural linkbases could then be increased, while the other branches of the tree could be reduced. In this way, links defined in some sections of the inheritance hierarchy are emphasised and play a more important role than other links. These weightings can be used to emphasise the role of particular linkbases in the hierarchy, while suppressing less important link collections.

This idea of using confidence values allows us to model *degrees* of inheritance. Links and linkbases filter down the tree, some falling effortlessly while others grow weaker as they reach the bottom of the tree. A form of weighted inheritance relationships offers a natural means of supporting an adaptive model, and opens up new opportunities for developing linkbase trees. Chapter 2 discussed a number of alternative adaptive techniques which have been employed in some hypertext systems, and Chapter 3 showed how some of these approaches could be incorporated into fuzzy anchors. Similarly, these confidence values which augment the inheritance tree can also be modified using some adaptive policy. The following section introduces the adaptive model used in HIPPO, and section 4.9.5 shows how this has been implemented in the current prototype.

4.5.3 Adapting Inheritance Values

The introduction of confidence values into the inheritance hierarchy, allows authors to indicate the importance of particular linkbases. Some link sets can be emphasised by allocating large values, while others can play less important roles. The current approach requires authors to select appropriate values for each relationship when they construct the linkbase tree. This is a useful extension to the model which allows the expertise and experience of the author to be captured explicitly. However, it also seems natural to incorporate some feedback from the users who will actually be using the link collections. Confidence values can be modified in response to user browsing patterns, to better reflect the true needs of the user. This can highlight any tree weightings which are inappropriate or have been misjudged by the author. Linkbases which were considered important may turn out to be less useful than expected. Similarly, linkbases which were given low confidence values can be assigned more major roles where appropriate.

These degrees of inheritance provide a natural platform for supporting an adaptive model. The current HIPPO implementation uses a relatively simple adaptive model which monitors the browsing patterns of the user, to see which links are explored. Each time a link is explored, the system assumes that this link is of some value to the user. The system then locates the corresponding linkbase which contains this link definition, and increases the weighting value which is associated with that link collection. In this way, linkbases which contain useful link definitions are given high confidence values, and their role in the overall hierarchy is increased. Conversely, linkbases which remain unused and which do not seem to contain many links that are of use to the user, are considered less important. The confidence values associated with these linkbases are subsequently reduced, to reflect this lesser importance. This adaptive model has been implemented in the current HIPPO system, and is described in more detail later in section 4.9.5.

Figure 4.10 shows an example linkbase inheritance hierarchy, which models a collection of link definitions. Each of these inheritance relationships has been assigned a confidence value to reflect the assumed importance of each linkbase in the overall hierarchy. Linkbase A contains a particular link definition which is subsequently selected by the user. HIPPO locates the linkbase which contains this definition, and increments the associated confidence value. Similarly, the remaining weighted values associated with the other linkbases in the tree are reduced. This adapted tree hierarchy reflects the new importance of linkbase A in the overall tree definition.

The adaptive model which is outlined here uses a very simple adaptive strategy, by taking user selections and applying these directly to linkbase tree definitions. The most immediate limitation of this approach is the problem of identifying useful

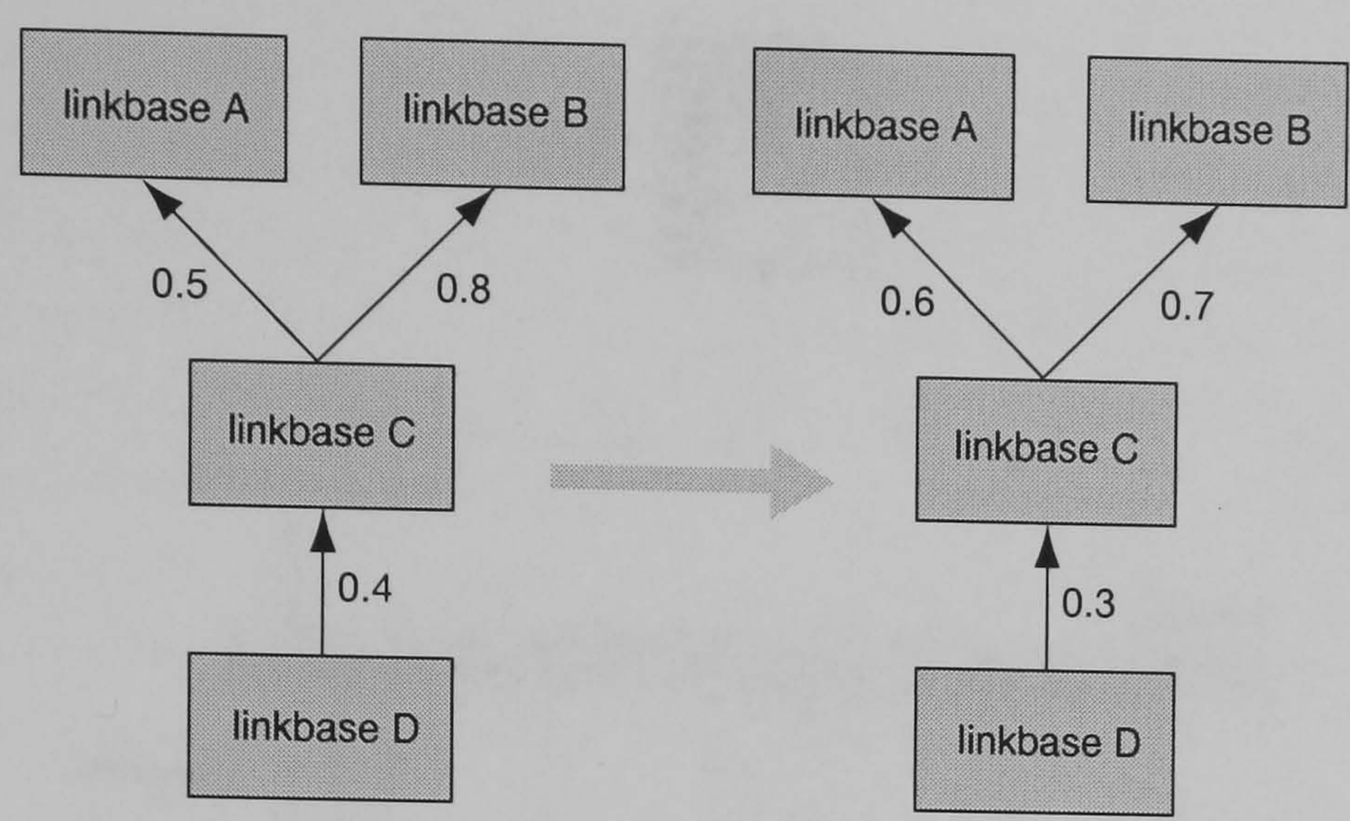


Figure 4.10: Adapting a linkbase tree

links from those hyperlinks which are less important. A system cannot assume that each link which is traversed is necessarily of benefit to the user. A reader of a hypertext typically explores many diverse branches, and reaches many dead-ends and useless paths. Indeed, this is one of the main problems of any hypertext model, and this *disorientation* problem is explored in some detail in Chapter 1. Chapter 2 showed how some adaptive hypertext systems have addressed this problem of identifying positive links. This is an area which could be developed further in future implementations, and Chapter 7 suggests some possible directions.

An implementation of linkbase tree hierarchies can adopt any combination of adaptive strategies to modify the hierarchy without affecting the overall approach to using linkbase trees. Chapter 7 includes some other adaptive ideas which have been considered for the implementation of linkbase trees in the HIPPO system. In particular, a simple user stereotyping model is discussed, which identifies groups of users and maps these on to collections of linkbase hierarchies. Groups of users are allocated particular tree definitions instead of using individual, global trees. This provides a finer level of granularity for the adaptive model, and could be easily added to the current HIPPO implementation. However, what is important is that confidence values are incorporated into the linkbase tree structure. A linkbase hierarchy is no longer a static, fixed tree, but has more responsive and adaptive features, which change over the lifetime of the hypertext.

4.6 Distributed Trees

The inheritance of linkbases promotes the idea of the linkbase as a fundamental hypertext abstraction, which can be treated as a first-class object. Linkbase trees encourage the intelligent reuse of link collections and provides a framework for ex-

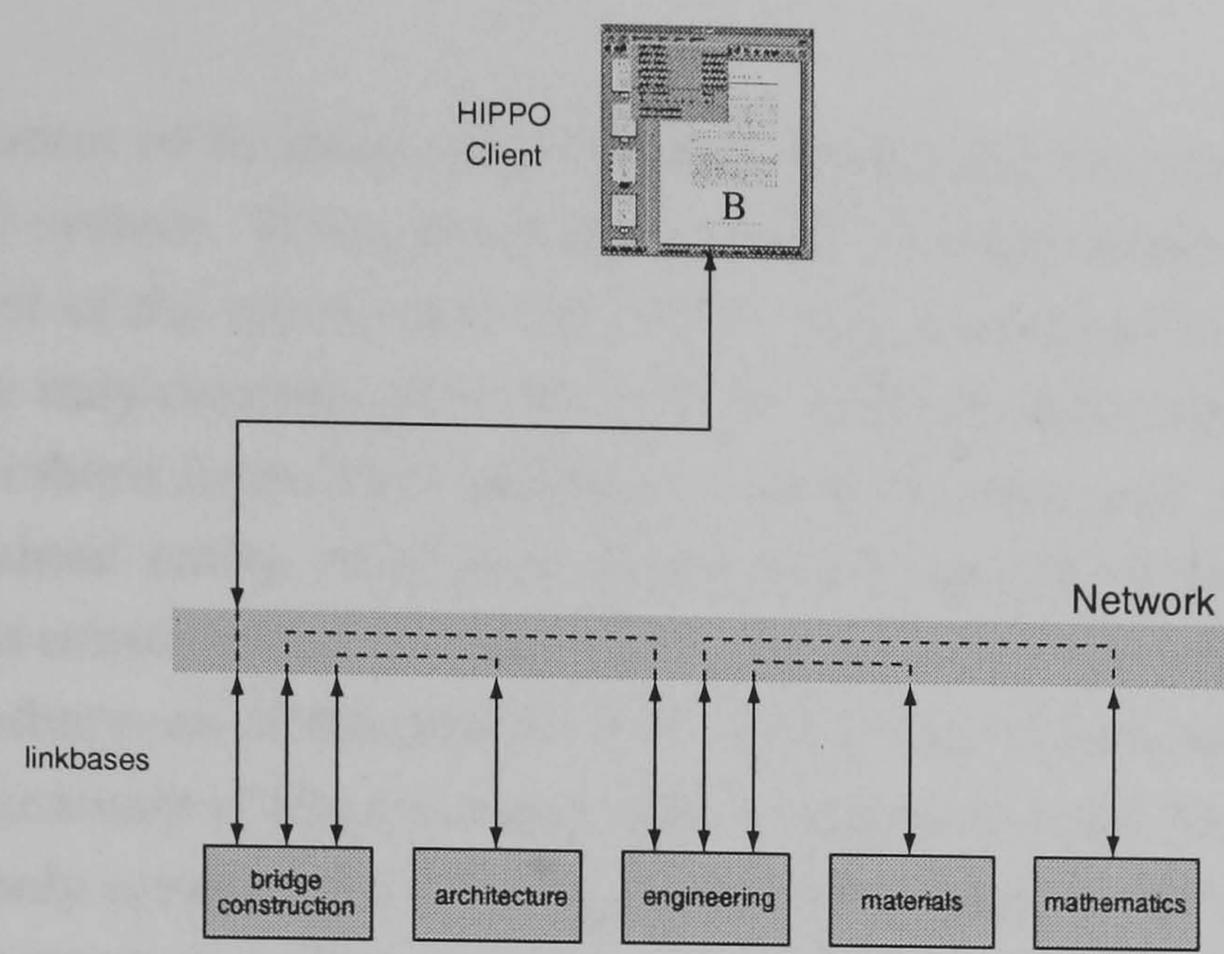


Figure 4.11: Distributing linkbase trees

pressing dependencies between link sets. More complex linkbases can be defined in terms of existing collections, and users are encouraged to reuse sections of a hierarchy. However, this model has only been discussed on a small-scale, using simple examples and shallow tree hierarchies. The real benefits of the model become clearer as the environment scales to include many hundreds and thousands of linkbases; carefully crafted by different authors, with many complex relationships and dependencies between linkbases.

This large-scale approach encourages enormous tree hierarchies, all reusing other collections of links – refining existing linkbases, overriding current links or excluding unnecessary link definitions. This is a natural opportunity to introduce a widely distributed topology. Linkbases no longer have to be located locally, but can be distributed linkbases throughout the network domain. Linkbases can be stored throughout the network and only retrieved on demand, when the tree hierarchy is evaluated (see figure 4.11).

There are numerous advantages to distributing the nodes of each linkbase tree. Linkbases can be located at the place where they are administered so that they can be updated without having to maintain redundant copies. The original authors of each linkbase maintain control of the collection, while at the same time, making this available to every other user in the network. The location of each linkbase can also be optimised, based on network topology or based on network traffic etc. Redundant copies of linkbases are removed because tree definitions always use *references* to linkbases rather than making physical, local copies of the link collections. This also avoids any problem of using out-of-date link definitions, because tree definitions always retrieve the current version when required (although long download times may require some caching technologies such as proxy servers or multiple mir-

rors).

The distribution of linkbase trees remains completely transparent to the user and the HIPPO system. When users incorporate a remote linkbase, they are unaware of the rest of the inheritance tree which may contribute to this linkbase. A remote linkbase may combine and inherit from multiple link collections, which in turn may each inherit from other linkbases. Users can treat each linkbase as a discrete, self-contained entity, regardless of the many complex dependencies which may go towards constructing the object. This provides a more scalable architecture, because it introduces an additional level of abstraction – users can build on existing linkbases, unaware of the linkbases which have been used further up the tree. The user need only consider the child linkbase, without any knowledge of the many parents which contribute to the overall hierarchy. This approach to distributed trees is discussed in the implementation of the HIPPO prototype, in section 4.9.4. Chapter 7 also outlines some of the areas where the distributed implementation could be improved further.

4.7 Combining Weighted Links, Weighted Trees and Fuzzy Anchors

The discussion so far has shown the value of separating hypertext linking information from the underlying node contents. It has been suggested that the *linkbase* should be seen, not as an end in itself, but as a building block which can be used to build complex structures. The idea of linkbase trees has been introduced to express relationships between linkbases and to build new link collections from existing linkbases. Furthermore, a number of weighting systems have been explored to model the importance and value of hypertext objects – weighted links, weighted linkbase trees and the fuzzy anchors which were introduced in chapter 3. This section show how these three approaches can be combined to provide a more expressive, coherent hypertext. These three approaches to weighted objects are summarised below:

- **Weighted links**
  Weighted links have been used, albeit in a slightly different form, in various adaptive hypertext systems and in some research into navigational systems [Fur86, PD90, PT90]. In the HIPPO model, weighted links provide the author with a way of emphasising particular links and showing the importance of certain relationships. In this way, a link captures the semantics of a particular relationship, but also a *belief* and *certainty* in those semantics.

- **Weighted linkbase trees**
  Linkbase trees offer a useful way of combining linkbases, and for reusing ex-

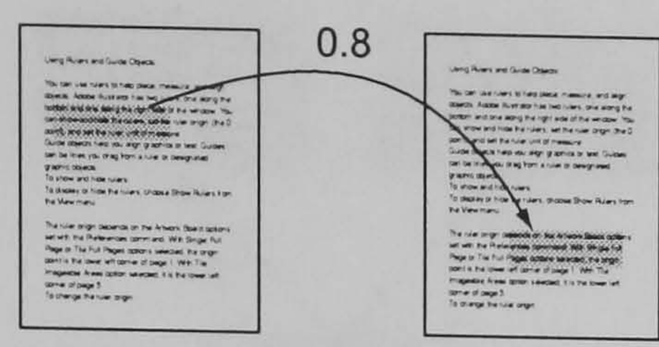
isting link collections. A linkbase should not expect to provide an exhaustive set of link definitions, but a more specific, finely-crafted collection. A linkbase typically supports a specific subject domain, but should also be considered in the wider context. A linkbase should be used in conjunction with other linkbases, reinforcing and supplementing these with new links. Linkbase trees allow the author to explicitly model these inter-relationships. They also encourage an approach to hypertext linking based on reuse and sharing – linkbases should be defined in terms of other existing linkbases, reusing some of the carefully chosen links which have been made by other experts and authors. The idea of a weighted hierarchy allows the author to attach confidence values to each inheritance relationship, to reflect the importance of each collection of links. Some linkbases will play increasingly important roles in a tree hierarchy, and others, while still useful, play a lesser role.

- **Fuzzy anchors**
  The previous chapter introduced an anchoring model based on fuzzy anchors. This argued that current approaches to hypertext anchoring do not reflect the uncertainty and ambiguity which are inherent in hypertext authoring. An anchor does not simply begin and end, but has emergent, flowing qualities. A fuzzy anchor uses the idea of fuzzy set membership, to provide an undulating anchor. Some elements can have a strong presence in a fuzzy anchor, and are considered vital to the anchor definition. Other elements which surround the main concept still have a role to play in this anchor definition, and help to set the central concept in some form of context. These supplementary elements can be assigned smaller fuzzy values to reflect this reduced importance.

These three approaches to weighted objects each model some form of ambiguity or uncertainty in hypertext and open hypertext in particular. These all use the idea of confidence values or fuzzy measurements to express the importance (or otherwise) of hypertext abstractions. However, each of these ideas has only been discussed in isolation, with little regard to how they affect each other. Each approach to weighting particular hypertext abstractions attempts to introduce additional, more expressive modelling capabilities. It seems natural to attempt to combine these approaches into a coherent overall model. This shows this has been achieved in the current implementation of the HIPPO model.

4.8 A Simple Example

A link in the HIPPO model expresses some semantics or relationship between objects in the hypertext (the current implementation only supports links between two objects, although there are no theoretical limitations to the number of anchors in



biology linkbase

```
...
link: src=a1; dest=a2; weight=0.8
...
```

Figure 4.12: Simple biology link definition

each link definition). The confidence value associated with this link is provided by the author, to express the importance or confidence in this relation. Important hyperlinks are assigned large confidence values, while less useful links are given correspondingly lower values. However, the *true* value of this link depends, not only on this initial value provided by the author, but also on the way it has been used in the linkbase tree hierarchy.

For example, consider a hypertext which explores some aspect of the science of *biology*. An author may create a link which combines two objects in the hypertext – perhaps a discussion with a definition of a scientific term. The author may be quite confident of the accuracy of this link, and believes it to be extremely useful to a wider audience. Subsequently, the author assigns the link with a high confidence value, say 0.8 (we assume weighted values are defined between zero and one). This scenario is shown in figure 4.12.

However, the real importance of this link depends very much on the domain in which it is being applied[2]. This link may well be of vital importance in a *biology* hypertext, but what if this link is being used in a more general medical situation? A different author may be defining a *medical* linkbase which can be used by medical students, for exploring a general medical hypertext. They may decide to incorporate the contents of the previous *biology* linkbase, along with many others – *genetics, physiology, neurology* etc. The author can construct a simple inheritance hierarchy to express these relationships, and can attach weightings to reflect the important of each constituent linkbase. A typical linkbase tree hierarchy might resemble figure 4.13.

In this case, the author has decided that although these biological link definitions are very useful, they are perhaps less important than a linkbase which supports *Clinical Procedures*. The author may feel that an introductory collection of

[2]In fact, the value of a link depends on many criteria – user knowledge, experience etc – and some of these areas are discussed further in Chapter 7

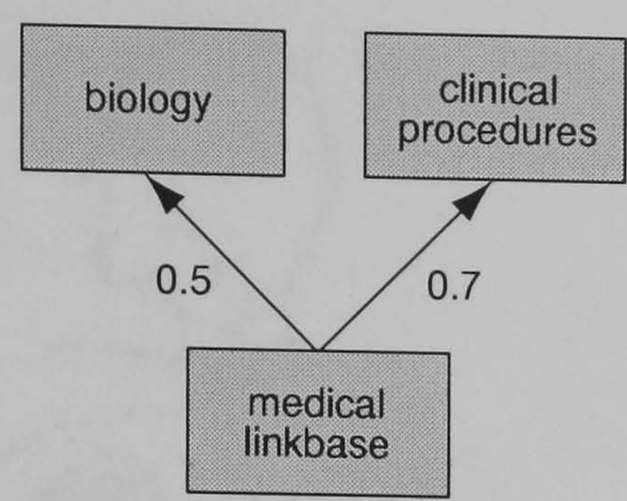


Figure 4.13: A medical linkbase tree

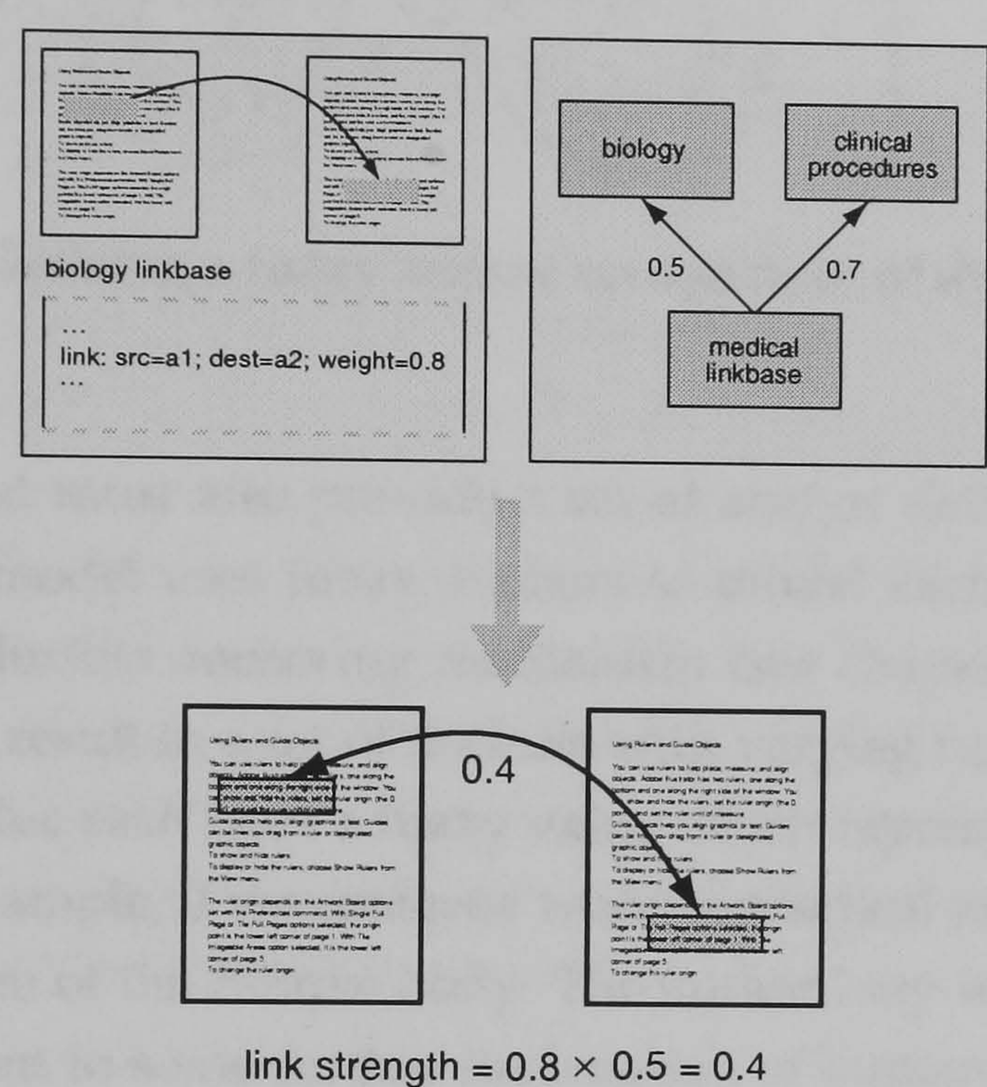


Figure 4.14: Combining weighted links with weighted trees

medical hypertext links should emphasise the practical side of the medical field, and that biological details are less important at this stage. Clearly, the value of the link we discussed in figure 4.12, while still as accurate as before, has less of a role to play in the new medical linkbase. As such, it seems sensible to combine the initial link confidence value assigned by the first author (0.8), with the weighted tree value chosen by the second author (0.5). A simple multiplication of the two values seems to express this idea, so that a new confidence value of (0.4) is generated (figure 4.14).

This newly calculated confidence value intends to represent, not simply the accuracy of the link, but also attempts to place the link in some form of wider context. The new link value combines the expertise of both authors – the original expert in biology, with the author who is building a new medical hypertext. This seems to be a very useful approach to hypertext construction, but we have yet to consider the role of the fuzzy anchor in this process.

The original author identified a particular relationship in the field of biology, and decided to make this connection explicit. They must assign some confidence

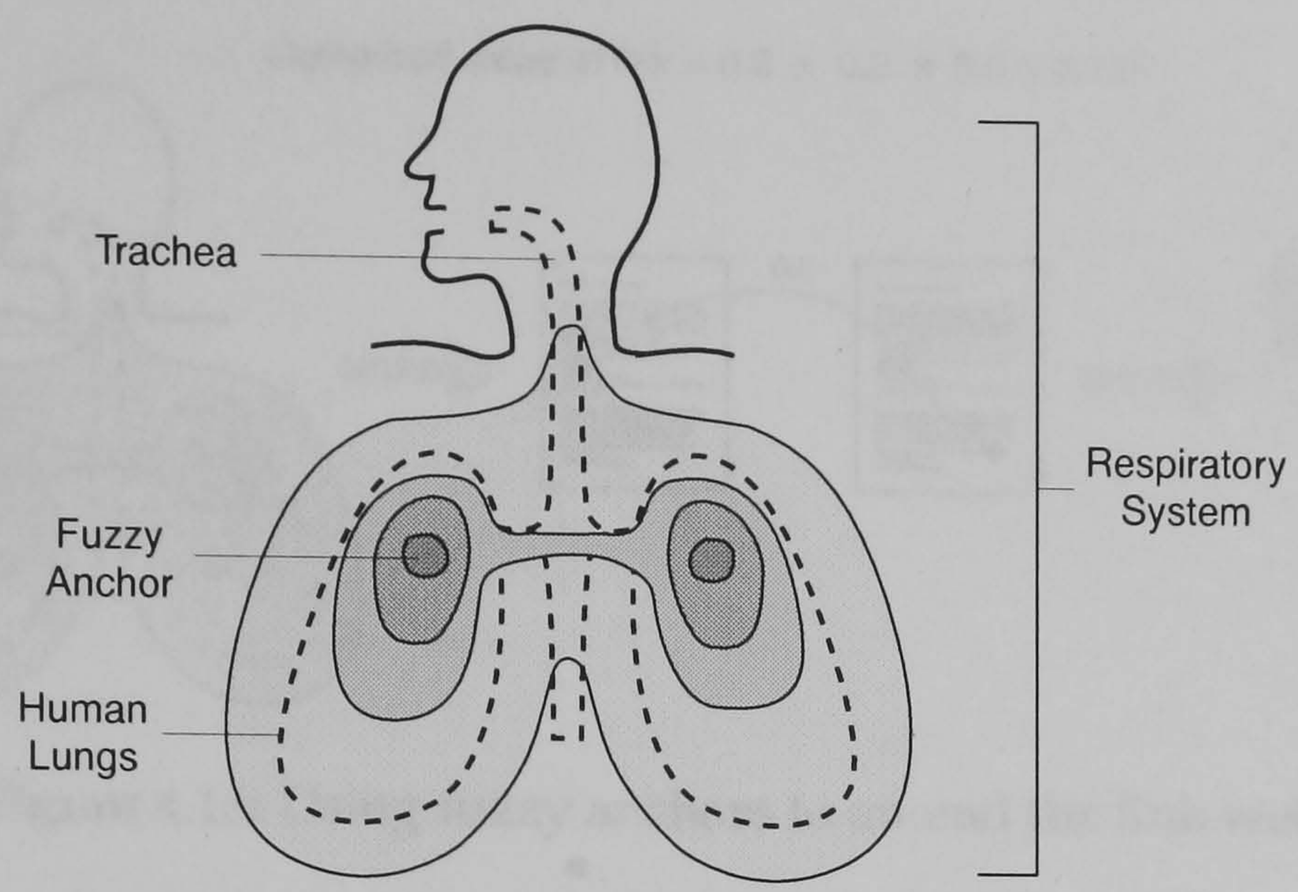


Figure 4.15: Defining a fuzzy anchor on a picture of the human body

value to the link, but must also provide a set of anchor definitions for each endpoint. The HIPPO model uses fuzzy anchors to model each link endpoint, in an effort to provide a flexible anchoring mechanism (see chapter 3 for further explanation). This would result in a set of anchors with varying fuzzy values – different areas of text or graphic each have a fuzzy value which represents their importance in the anchor. For example, if we continue with the medical example, the hypertext may contain a picture of the human body. The author may wish to link an area of the respiratory system to some further explanation of human respiration. The respiratory system has no clear boundaries – no definite start or end – and cannot be clearly delimited. However, fuzzy anchors allow the author to capture this ambiguity and express the uncertainty of the endpoint. The author decides to create a fuzzy anchor which focuses on the human lungs, but also includes some section of the *trachea* and the surrounding region (see figure 4.15). This results in a very rich and expressive anchor which seems to encapsulate the central concept while still capturing the wider context.

When the user encounters this picture of the human anatomy, they may decide to explore this picture of the human body in more detail. The user may have some interest in the physical problems associated with human respiration, in particular, perhaps at ways of avoiding choking or blockage of the windpipe. The user clicks on the *trachea*, in an effort to find more information. Figure 4.15 showed a fuzzy anchor which was defined to include some elements of the trachea, and would seem a natural choice. However, this anchor had a clear focus on the lungs, rather than the windpipe area, and it may be that other anchors, with a stronger presence over the windpipe would be more appropriate. The value of a hypertext relation lies not only in the accuracy of the link (or the way it is used in a linkbase tree), but also in the definition of the anchor. An anchor attempts to encapsulate fundamental

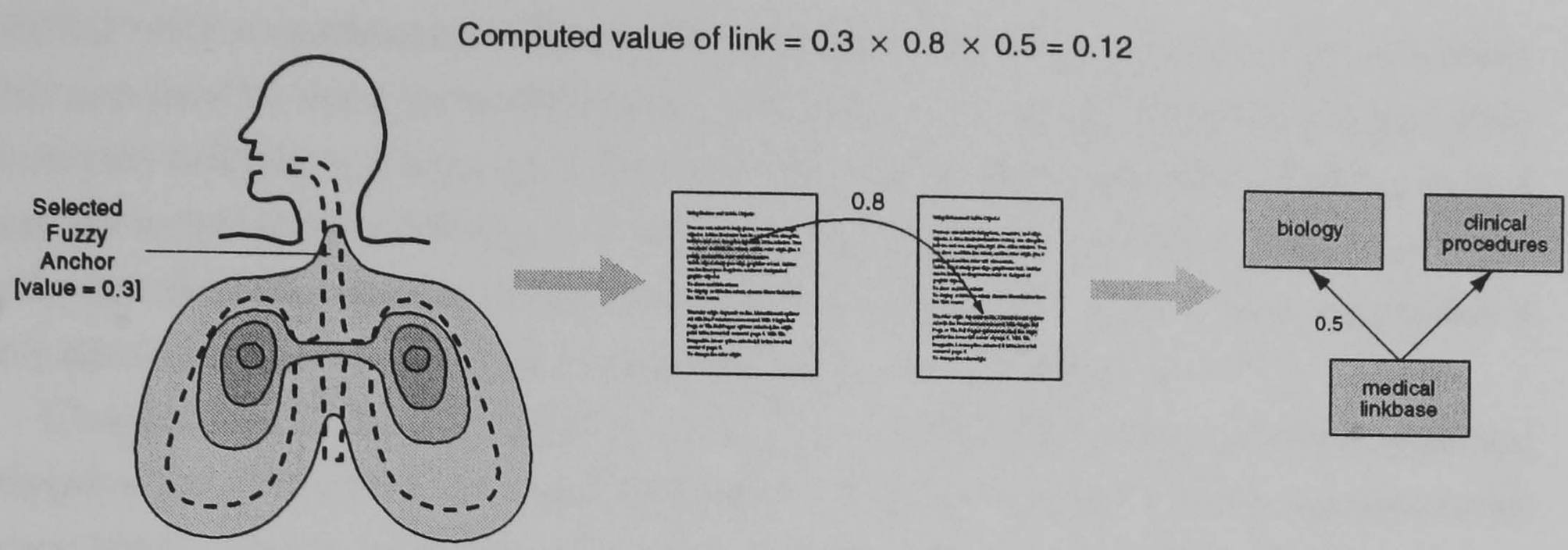


Figure 4.16: Using fuzzy anchors to amend the link weights

elements at each end of a link, elements which may not have clear boundaries. As such, it seems sensible to use the fuzzy definition of the anchor to modify the link weighting that we have calculated so far.

The region of the human windpipe that was selected by the user may have a rather low rating in the fuzzy anchor definition, say 0.4. This reflects the reduced importance of the windpipe in the respiratory system, and the greater role of the lungs etc. We could then generate a new confidence value by incorporating this anchor value into our current equation. The final value which represents the *true* confidence of the relationship becomes the product of the anchor value, link weighting and tree weightings ($0.3 \times 0.8 \times 0.5 = 0.12$). Figure 4.16 shows this final calculation in effect. This final value can then presented to the user, before deciding whether to traverse the link or choose another link. The user can then follow links which have a high confidence value, and ignore lower values.

This method of calculating the certainty values of hypertext relationships is applied universally to all relationships in the HIPPO system. It attempts to combine all of the ideas presented so far, and results in a link which reflects the rich semantics of each weighted object. The expertise of the author who identified the relationship is combined with the richness and expressive qualities of fuzzy anchors. This then combines the expertise of the author who constructed the inheritance hierarchy, and decided how the original link would be used in the overall hypertext. Section 4.9 shows how this overall model has been implemented in HIPPO.

4.9 Implementing Adaptive Linkbase Trees In The HIPPO System

This chapter has outlined the concept of *linkbase trees*. Linkbases are viewed as fundamental objects which can be arranged into inheritance hierarchies, to promote a scalable model based on reuse and sharing. Each inheritance dependency is aug-

mented with a confidence value, which represents the *strength* of the relationship. This can then be used to model *degrees* inheritance, to assign particular importance to certain linkbases. These confidence values were combined with a linking model base on *weighted links*. Finally, it was suggested that this model is suitable for large-scale distribution – linkbases can reside on remote machines to reduce redundancy and allow authors to maintain ownership of the link collections.

Chapter 3 outlined an initial prototype of the HIPPO system which supported adaptive fuzzy anchors, and was implemented using the Acrobat document framework [Acr]. This plug-in has been extended to support the ideas which are introduced in this chapter. The new application uses linkbases to separate linking information from the node and anchor definitions. The HIPPO prototype provides tools for defining linkbase hierarchies, which allows authors to define more complex inheritance relationships between separate linkbases. The system also implements a distributed model described in section 4.6, which allows linkbases to reside on remote platforms. Some aspects of this distributed model have not been implemented, and these are discussed in Chapter 7. Finally, the adaptive server described in Chapter 3, which was used to modify anchor definitions, has been extended to support adaptation of linkbase hierarchies. This section shows how the HIPPO system has been extended to support these ideas, and discusses the design and implementation of this new HIPPO system.

4.9.1 Linkbases In The HIPPO Prototype

The initial implementation of the HIPPO prototype which was described in chapter 3, discussed the idea of fuzzy anchors and *anchorbases*. A format and grammar specification for these anchor definitions was included, which contained all of the information which is required for a complete anchor description. In particular, each anchor definition included a *dest* entry, which indicated the target destination of each link. In this way, the user could select any anchor, then traverse the link to reach the target document. This approach combines the anchor definitions with the link specification, and was useful for demonstrating how fuzzy anchors fitted into an existing hypertext model. However, this chapter has shown the many advantages which can be achieved by separating linking information from the anchoring details (section 4.1).

The anchor definitions discussed in chapter 3 do not include any reference to a particular document, and focus entirely on addressing pieces of content within the node [3]. Instead, it is the link definition which defines the document which each anchor refers to – this allows a single anchor definition to be used in many different documents. The particular document which contains the anchor is no concern of

[3]The original prototype did include references to documents in the anchor definitions, but this was used to demonstrate the benefits of *fuzzy anchors*, and was removed in later implementations.

the anchor, and can change between subsequent link definitions.

For example, consider a document which contains the author's name at the top of each page. Other documents may use an identical layout, and also include a reference to the author at the head of each page. The author may wish to include a link from this name, to a node containing more information about the author. In this case, the author could create a link on each page of each document, but need only make a single anchor definition, which then be shared between each of the links. This reduces the storage requirements of anchor definitions because they only appear once (this is especially important when using fuzzy anchors, which can include large definitions). Furthermore, any changes to the anchor definition, need only be applied to the single, shared definition.

All of the hypertext links are separated into *linkbases*, which describe the hypertext relationships, and refer to anchors using their anchor ID numbers (figure 4.17). This approach allows the anchoring and linking mechanisms to be developed and extended independently of each other. For example, the fuzzy anchor can be replaced with a simpler, more traditional anchor model, without affecting any of the hypertext link definitions. Alternatively, any number of different link models could be used – typed links, weighted links, dynamic links – without affecting any of the existing anchoring details. Similarly, anchors can be shared between several links, so that any modifications to an anchor only need to be done once. Users share the same anchor definitions, while replacing a set of link definitions with another linkbase – perhaps links which are considered more suitable for the particular user.

The current HIPPO application implements linkbases as traditional files – each file contains a collection of links which can be loaded into the prototype. The plug-in provides additional menu operations to load a linkbase file and manipulate links. The application also defines a standard format for specifying each linkbase, which is discussed in the section 4.9.2. The author specifies tree hierarchies using a separate tree definition file – this defines which linkbases inherit from which existing collections. The implementation then provides tools to read these tree definitions, and generate the appropriate linkbases. The remainder of this section explains how linkbase trees and inheritance hierarchies are specified, and how the current implementation supports these. The adaptive server which updates and modifies anchor definitions (see Section 3.6.6) has been extended to handle adaptive links and link trees, and is discussed in section 4.9.5.

4.9.2 Linkbase Format

The first priority for extending the HIPPO application to support linkbase trees, was to agree on a standard format for linkbases. This specifies the syntax for defining each hypertext link in a linkbase file. As before, the *flex* and *bison* tools were used to construct a lexical analyser and parsing tools, which could then be used in the

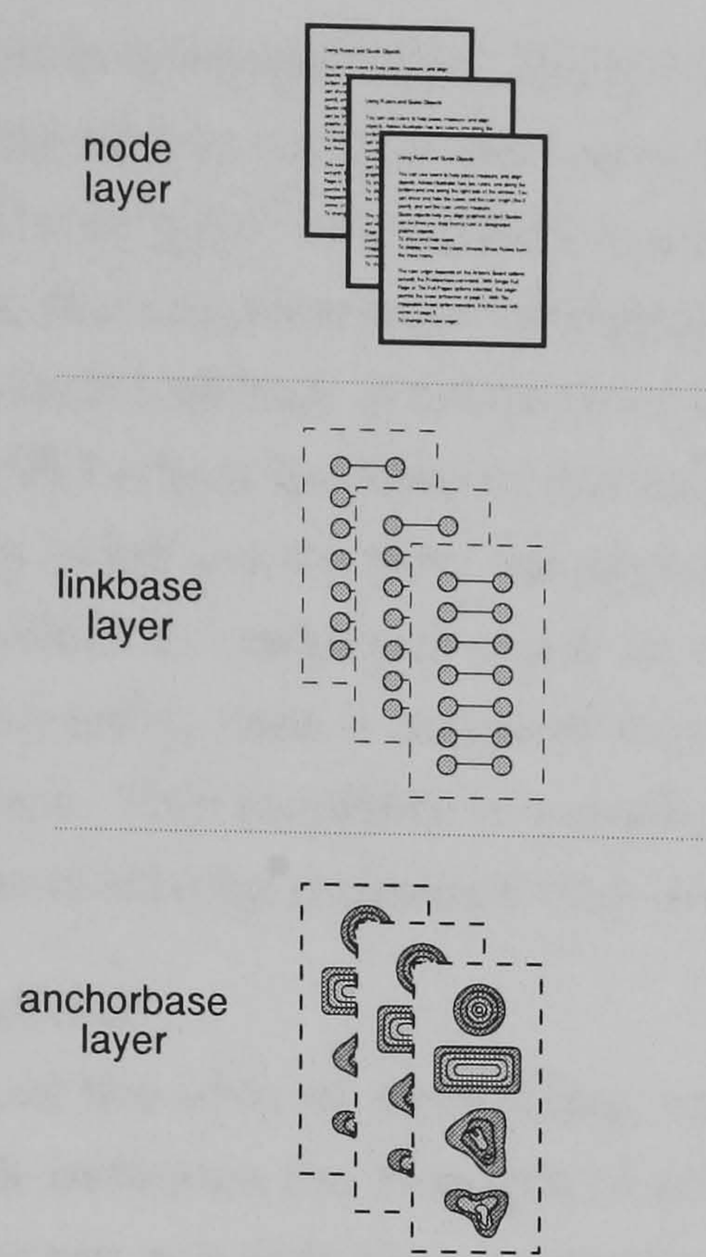


Figure 4.17: Anchor definitions maintained separately from links

```
--
linkbase = <linkbase ID>
strength = <strength of link>
srcDoc = <source document>
srcAnchor = <source anchor ID>
destDoc = <target document>
destAnchor = <destination anchor ID>
```

Figure 4.18: HIPPO linkbase format

HIPPO system. Readers are referred to [Foub, Foua] for more information on these language analysis tools. A linkbase contains a collection of link definitions, where each link definition is of the form:

Each of these fields is discussed here in more detail, and Appendix D includes a full specification of the lexical analyser and linkbase grammar. Entries which are marked *Optional* can be omitted from the link definition, and assume default values.

- **--**

  Each link definition must begin with this symbol. This simplifies the parsing of the linkbase, and allows each link definition to contain optional and missing elements.

- **linkbase** = *<linkbase ID>* *Optional*

  The HIPPO application allows authors to define linkbase hierarchies, by spec-

ifying a tree definition in a separate file. The prototype includes an adaptive server which monitors link traversals, and uses this information to modify weighted values in the original tree definition (see section 4.5.3 for more information). However, the adaptive implementation needs to know the origin of the link, that is, which linkbase actually contains the link definition. This *linkbase* field tells HIPPO which linkbase in the tree contained the original link definition. If the entry is left empty, then the application assumes that the link is a new definition, which is contained solely in the newly defined linkbase. If the field contains an entry, then it assumes that the link was originally defined higher up the tree. This identifier is a single string, which must appear somewhere in the tree hierarchy definition (see section 4.9.4).

- **strength** = *<strength of link>* *Optional*
  Section 4.5.1 discussed the idea of associating confidence values with each link definition, which indicates the strength of each hypertext link. This allows the author to express, not only the occurrence of a hypertext relationship, but also the strength and importance of this link. Other systems have shown how weighted links can be incorporated into an adaptive hypertext model. This *strength* entry in a link definition associates a weighted value with the link, and must contain a value between zero and one. Section 4.7 explains how this confidence value is combined with the weighted inheritance relationships, and how it is used in the HIPPO system. If the value is omitted, then the strength is assumed to be the maximum (ie. 1).

- **srcDoc** = *<source document>* *Optional*
  Each *srcDoc* entry must contain a valid *Universal Resource Locator* (URL) which indicates which document the source anchor refers to. This uses the standard URL syntax [URL94].

  If the entry is left empty, then the application assumes that the link is valid for every document. This allows Microcosm-style *generic* links (section B.3) to be supported in the HIPPO application. This can reduce the authoring effort involved in creating new hypertexts, by allowing a single link definition to be applied to each document node in the hypertext.

- **srcAnchor** = *<source anchor ID>*
  As discussed previously, the anchor specifications are maintained separately from the link definitions. Hypertext links refer to anchor definitions indirectly, by specifying an anchor identifier. This is then used by the application to locate the appropriate anchor definition in an *anchorbase*. Anchor identifiers can be arbitrary strings, which must match an anchor identifier in the current anchorbase (see Chapter 3).

- **destDoc** = *<target document>*
  This field defines the target destination for the current link. As before, the format uses the standard URL naming scheme to reference documents. The current implementation uses an appropriate World Wide Web browser to locate and retrieve the target document. This entry must be included in the link definition.

- **destAnchor** = *<destination anchor ID>* *Optional*
  This entry refers to a destination anchor, which defines the precise target location. As with the *srcAnchor*, the entry must include a valid anchor ID, and is used to look up an anchor from the current anchorbase. If the entry is omitted, then the link simply traverses to the node as a whole, and does not reference any content *within* the node. It is important to note that the current HIPPO implementation uses fuzzy anchors which are only supported for PDF documents. The prototype cannot support links *within* other content formats, however, authors can still link to other formats at the node level.

4.9.3 Linkbase Trees In The HIPPO Prototype

Linkbase trees are supported in the new HIPPO prototype, and allow authors to express inheritance relationships between linkbases. New linkbases can be defined in terms of existing linkbases, and can refine, extend or modify previous link definitions. The current implementation defines these inheritance hierarchies using a syntax derived from the C++ language which is used to express inheritance relationships between classes. This has been modified to support linkbases, and the syntax is discussed in section 4.9.4. Appendix E includes a full specification for the lexical analyser and grammar for linkbase trees.

The prototype provides a *build-tree* tool which reads a tree template, and builds an internal tree. This utility was defined using the *flex* and *bison* utilities to generate parsing functions. Once the tree definition has been parsed, the prototype then traverses the tree hierarchy, and combines the component linkbases into a single linkbase definition. This new linkbase contains link definitions from each linkbase that it inherits from (and in the order that each node appears in the hierarchy).

Each inheritance relationship which is specified in the tree definition, has a confidence value associated with it. This is used to indicate the importance of the linkbase in the overall hierarchy, and express its value in the linkbase tree (see Section 4.5.2). This confidence value is then used to modify the link strengths of each link definition in that linkbase. For example, consider part of the inheritance hierarchy in figure 4.19. This defines that *child* should inherit from *parent*, and attaches a weighting of *0.8* to this relationship.

Figure 4.20 includes an extract from the *parent* linkbase, which shows some of

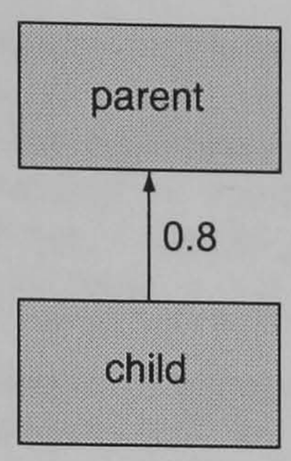


Figure 4.19: An example weighted inheritance hierarchy

```
linkbase = parent
strength = 0.8
srcDoc = index.pdf
srcAnchor = id-0001
destDoc = form.pdf
destAnchor = id-4895
--
linkbase = parent
strength = 0.5
srcDoc = index.pdf
srcAnchor = id-1003
destDoc = eprg.pdf
destAnchor = id-3845
--
linkbase = parent
strength = 0.2
srcDoc = eprg.pdf
srcAnchor = id-1923
destDoc = eprg.pdf
destAnchor = id-1924
```

Figure 4.20: Extract from the parent linkbase

the link definitions which might appear. Each link definition has a confidence value which expresses the value and importance of each link:

The weighting which was attached to the inheritance relationship (*0.8* in this example) can be used to modify these link definitions. These new link weightings are calculated as the product of the inheritance value and the original link value. The new link definitions which are inherited by the *child* linkbase are shown in figure 4.21. A full explanation of the way these weightings are combined is given in sections 4.7 and 4.8.

This approach still allows each link to have an independent weighting, which can express the importance of each link in the hypertext. However, the introduction of *weighted inheritance* values also allows different linkbases to be assigned a different importance, and for these to be reflected in the final link definitions. Section 4.9.5 shows how the prototype uses feedback from the user to modify these inheritance values.

```
linkbase = parent
strength = 0.64
srcDoc = index.pdf
srcAnchor = id-0001
destDoc = form.pdf
destAnchor = id-4895
--
linkbase = parent
strength = 0.4
srcDoc = index.pdf
srcAnchor = id-1003
destDoc = eprg.pdf
destAnchor = id-3845
--
linkbase = parent
strength = 0.16
srcDoc = eprg.pdf
srcAnchor = id-1923
destDoc = eprg.pdf
destAnchor = id-1924
```

Figure 4.21: New link weights inherited by child linkbase

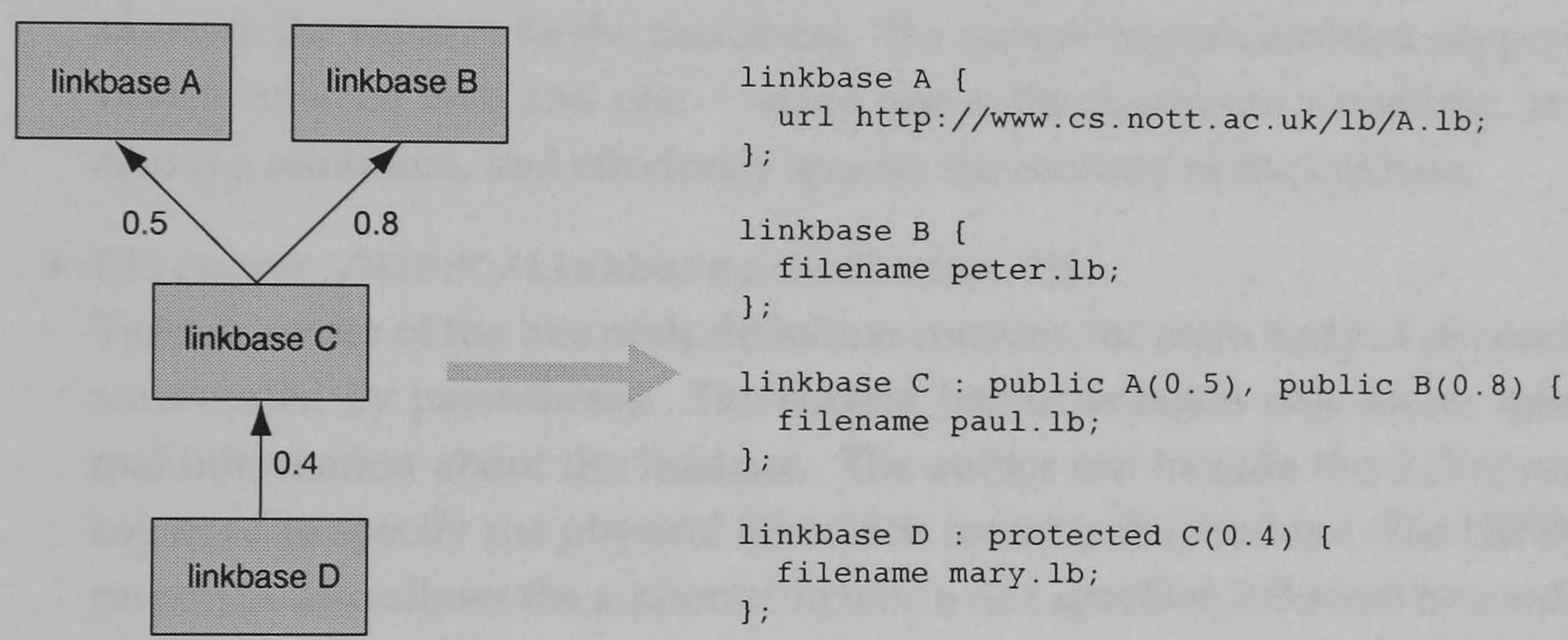


Figure 4.22: An example tree definition

4.9.4 Linkbase Tree Format

The HIPPO system defines a syntax for specifying linkbase inheritance hierarchies. An example tree definition is shown in figure 4.22 and a more complex example is included in Appendix F.

- `linkbase medicine`
  Each node in the inheritance hierarchy must begin with the `linkbase` keyword. This must be followed by a linkbase identifier which can be any sequence of letters. This identifier does *not* refer to the actual physical file containing the linkbase, but is used to refer to the linkbase throughout the tree definition. For example, this example uses the *medicine* identifier, which then appears later in the example when other nodes wish to inherit from this linkbase.

- `biology (0.5), genetics`
  Once a new tree node has been introduced, the definition must include a comma separated list of inheritance specifications. These define which existing linkbases the new node will inherit from, and provide additional information about the nature of this inheritance. The `public` and `private` keywords are used to indicate the level of access control. Access control is an important idea in Object-Oriented design, which controls the level of visibility of inherited elements. Possible applications of this access control were discussed in section 4.4.2, although the current implementation ignores these access specifiers.

  Each linkbase identifier can have an optional numeric value contained in parentheses. This value is used to indicate the *inheritance weighting*, which signifies the importance (or otherwise) of each inheritance relationship (section 4.5.2). This example uses a value of 0.5. If this numeric entry is omitted, then HIPPO assumes the value to be the maximum. The current implementation supports values between zero and one – where one is the maximum weighting, and zero is a minimum, and effectively ignores the contents of the linkbase.

- `filename /HIPPO/linkbases/medicine.lb;`
  The remainder of the tree node definition contains the main body of the node, surrounded by parentheses. The current implementation only stores minimal information about the linkbase. The author can include the `filename` keyword to specify the physical file which contains the linkbase. The HIPPO prototype also allows the author to include a `url` specifier, followed by a valid URL to refer to linkbases which are stored remotely. In this case, HIPPO will retrieve the linkbase from the remote machine using a set of Java classes. Future implementations can make additions and extensions to the linkbase body (linkbase author, comments, usage etc).

4.9.5 Extending The Adaptive Server

Section 4.5.3 showed how feedback from the user could be used to modify the weightings associated with a linkbase tree definition. The construction of a linkbase inheritance hierarchy can be a complex task, especially as the size of the tree increases. It can be useful to incorporate feedback from the user, to automatically modify and "tune" a linkbase hierarchy. Chapter 3 described the design and implementation of an adaptive server for modifying fuzzy anchor definitions. This server has been extended to support adaptive trees, and is explained in this section in more detail.

Chapter 3 showed how clients communicate with the server using remote procedure calls. This allows the server to be located centrally, and to service arbitrary

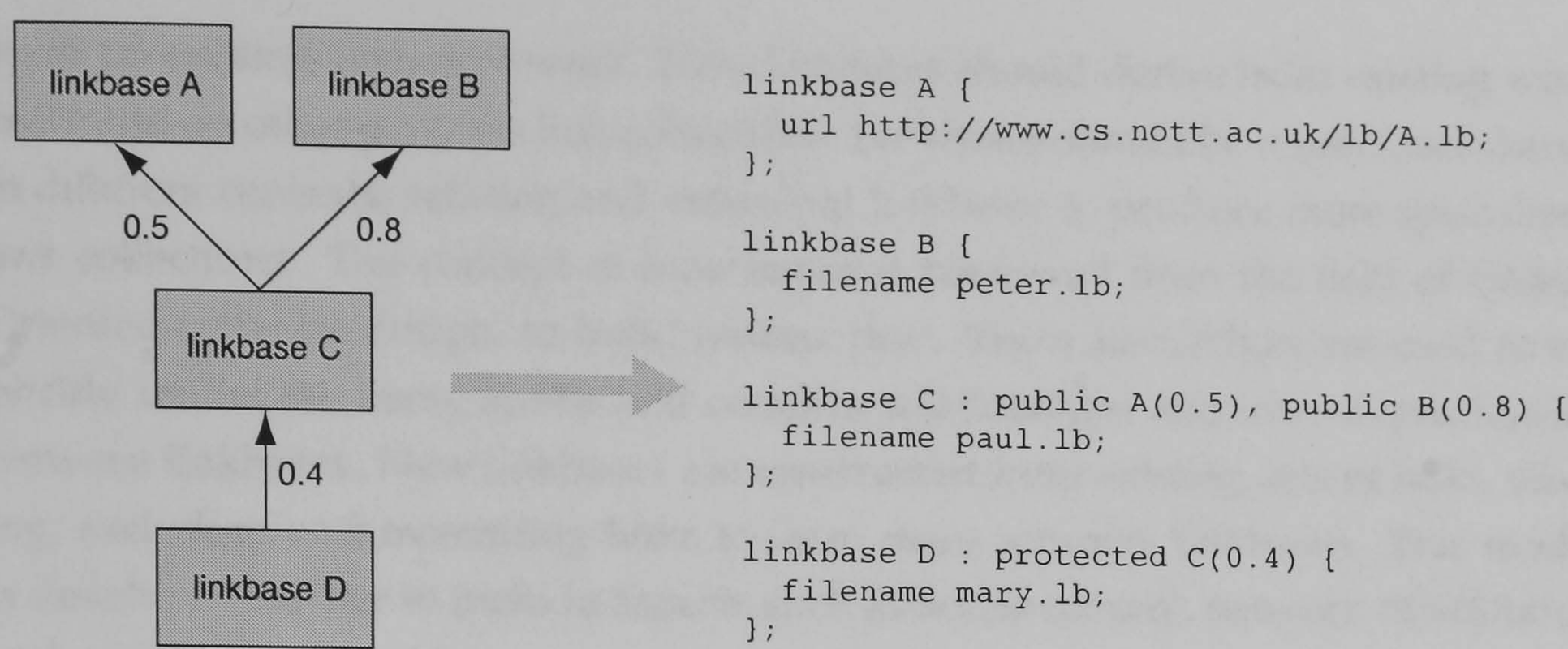


Figure 4.23: Using the adaptive server to modify a linkbase tree

numbers of clients. Section 4.5 outlined the adaptive model which has been used in the HIPPO system. Each time a user explores a link, this hyperlink is assumed to be of value to the user and should be emphasised in the tree hierarchy. Each HIPPO client notifies the server of each link selection, along with any information which is needed for adapting anchor definitions. The client sends a linkbase identifier, and the file containing the tree definition which is currently being used. The server retrieves this tree definition, then increments the value associated with the appropriate linkbase identifier. Similarly, the remaining inheritance weightings which refer to other linkbases are reduced (see figure 4.10). In this way, popular linkbases emerge with high confidence values, while less important linkbases tend towards zero. An inheritance relationship with a zero weighting will effectively ignore the contents of the linkbase.

4.10 Summary

This chapter has argued that modern hypertext systems should promote a level of abstraction when dealing with hypertext links, and adopt a constructive view based on linkbases. The conventional view of a link treats a hyperlink as a discrete entity, existing in isolation from the rest of the hypertext. However, hyperlinks have a very important collective role, reinforcing and supporting other links and connections in the hypertext. Future approaches to link management should borrow from the field of software design and analysis, moving towards a more maintainable, reusable view of hypertext links.

The concept of a linkbase – a collection of specific links – is discussed, and it is suggested that this collective view of links should be used as the basic building block to construct complex hypertexts. Hypertext research should be moving towards more reusable, maintainable practices, so that new hypertexts can share and

build on existing bodies of work. New linkbases should derive from existing work and build on other people's link collections. Linkbases should be reused and shared in different contexts; refining and extending linkbases to produce more specialised link collections. The concept of inheritance is borrowed from the field of Object-Oriented software design, to build *linkbase trees*. These hierarchies are used to explicitly model the many subtle and complex relationships and inter-dependencies between linkbases. New linkbases are constructed from existing sets of links, sharing, excluding and overriding links to form more complex linkbases. This model is developed further to include aspects such as access control, network distribution and weighted hierarchies.

Authors can use inheritance techniques such as *specialisation* and *exclusion* to construct more complex linkbases from existing linkbase objects. Some form of access control has been introduced to show the benefits of protecting elements in the linkbase. The inheritance model has also been developed to include weighting values which can be used to emphasise certain relationships and reduce the importance of other inheritance paths. These have been combined with the idea of *weighted links* which attach confidence values to links, and allow the author to express the importance or certainty of links. An adaptive model has been developed which incorporates feedback from the user to modify and update the confidence values in the inheritance tree. Finally, a distributed architecture is suggested in which linkbases are located throughout the network domain and accessed remotely. The HIPPO prototype described in Chapter 3 has been extended to support this idea of linkbase hierarchies and is described in detail. Chapter 7 suggests some areas which could be improved and developed further in future research.

Chapter 5

HIPPO+ – Distributing The HIPPO Model

The opening chapters showed how the hypertext model has been developed over the years to meet the demands of modern computing tasks. Initial monolithic applications have been replaced by open approaches to hypertext. These open hypertext systems attempt to generalise and develop many of the key abstractions which are common to all hypertext systems. Chapter 2 identified many of the issues arising from open hypertext research, and explored the idea of *link services*. The work presented so far in this thesis has attempted to develop the anchoring and linking models used in open hypertext, and to show how these have been used in the HIPPO system to support a flexible hypertext model.

The HIPPO model introduces the idea of *fuzzy anchors* and *linkbase trees* to provide an expressive hypertext model. These address many of the problems in OHSs – scalability, separation of hypertext information, reuse, distribution of information etc. In particular, the adaptive model which has been developed helps to address many of the tailorability and maintenance problems in a hypertext. These adaptive ideas help to mould and shape the system to match the precise demands of the user. This moves away from the traditional view of a hypertext as a fixed, static system, towards a more flowing and responsive environment.

However, the current prototype of the HIPPO system has been largely implemented as a monolithic application. The majority of the functionality and intelligence is embedded deep inside the application, and remains hidden from the user. Open hypertext systems demand a less tightly-coupled environment which allow services and operations to be shared and reused. An open hypertext environment should make this functionality available to other environments so that existing applications can incorporate this hypertext technology. Existing tools should be able to make use of the ideas presented in this thesis, without significant re-implementation and alteration. Users should be able to tailor and extend their hypertext environ-

ment by selecting those services and functionality which they find useful.

This chapter describes the HIPPO+ implementation which re-implements the HIPPO model using a distributed architecture. The HIPPO prototype has been broken down into key services and components which can be widely distributed throughout a network domain. These services can then be made available to all users, so that the ideas developed in the HIPPO research can be shared and reused between applications. A distributed model based on communicating services and components offers many advantages over the monolithic, closed HIPPO implementation, and these benefits are discussed in more detail. The HIPPO+ system is compared with existing distributed frameworks, and some of the problems which arise in a distributed environment are discussed.

5.1 Summary Of The HIPPO Prototype

The previous chapters have introduced some of the early research in the hypertext community, and shown how this has been developed in recent years. Early applications offered a variety of linking models and supported numerous navigational tools to help the user browse a hypertext (see Chapter 1). However, no matter how useful these additional tools were to the user, these early applications were implemented as monolithic applications. The hypertext tools and services were hidden from the user, and could not be used by other applications in the users' environment. If users wished to incorporate any hypertext functionality, then they were forced to give up their existing tools and applications, and adopt a single, all-inclusive hypertext system.

Chapter 2 showed how recent research has moved away from these closed, monolithic applications, towards open hypertext environments. These applications reject the view of hypertext as a single application, and provide hypertext *services* which can be used as an *integrating* technology. The chapter discussed some of the requirements of open hypertext systems, and showed how these have been achieved in modern applications. OHSs often incorporate the idea of the *linkbase* to separate linking information from underlying node contents. Open systems also have a strong notion of the hypertext anchor, which allows a clean separation between the linking and anchoring mechanisms.

The work presented in this thesis so far has attempted to develop some of the common abstractions which are used in open hypertext systems. These have been combined with adaptive modelling techniques to provide a powerful and expressive hypertext environment. These have been implemented in the HIPPO prototype which has been described in detail. Chapter 3 introduced fuzzy anchors which offer a more subtle and flexible anchoring model. Many open hypertext systems have acknowledged the importance of the anchor in a hypertext environment, and shown the benefits of treating anchors as first-class objects [SLH94, LS94]. Fuzzy anchors can better capture the emergent nature of link endpoints, and can incorporate feedback from the user to modify anchor definitions. Chapter 4 developed the notion of the linkbase and showed how this could be used to build more reusable, maintainable data structures. Linkbase trees incorporate the ideas of inheritance used in OO research, to encourage reuse and specialisation of link collections.

The HIPPO model addresses many of the requirements of open hypertext environments which were discussed in chapter 2. Linkbase trees address some of the scalability problems associated with large hypertext environments and promote a distributed topology. The separation of anchoring and linking information supports an open hypertext model which allows different link and anchor sets to be used by different users. Node contents remain untouched so they can continue to

be used with their native applications, and it is easier to integrate different linking models and formats. More importantly, the previous work has shown how adaptive modelling techniques can be incorporated into the open hypertext paradigm. This helps to address many of the tailorability and maintenance problems which are typically associated with hypertext applications. Section 2.1 identified a number of requirements for open hypertext systems, which are re-evaluated here in light of the current HIPPO implementation:

- *scalability*
  Chapter 4 discusses the idea of *linkbase inheritance trees* which address some of the scalability issues involved in hypertext design. Linkbase trees encourage an authoring model based on reuse and shared resources. Link collections are tailored and specialised to meet the demands of specific hypertexts, and linkbases are used to construct larger link sets. Link trees allow authors to explicitly model the relationships between diverse link collections, which supports a scalable and maintainable hypertext model.

- *distribution*
  The current implementation of the HIPPO system incorporates some distributed techniques into the hypertext model. The adaptive server which supports the adaptation of fuzzy anchors and link trees has been implemented as a remote, centralised server. This allows multiple clients to provide feedback to a central server, and provides some level of distribution. Section 4.6 also showed how each node in a linkbase tree can be located remotely. Linkbases can be distributed throughout the network and retrieved on demand.

- *heterogeneity*
  The separation of linking information into linkbases has been widely used in OHSs to provide some heterogeneous support. Linkbases can be replaced and tailored to the user, or merged with existing collections. A linkbase can be used regardless of the underlying node formats, and multiple node contents can be serviced by the same linkbase. Chapter 3 also showed how anchor definitions can be maintained separately from the hyperlink information, which provides a clean separation between the two layers. This separates any linking concerns from the specific details of the node addressing mechanisms. Widely differing node formats can be incorporated into the HIPPO model by providing a suitable anchoring implementation – this can be done without affecting any linking model which is used elsewhere in the HIPPO system.

- *extensibility*
  The separation of linking and anchoring information from the node contents also promotes some degree of extensibility. The underlying linking model can

be extended without affecting the nodes, and the format of link definitions can be extended arbitrarily. Additional anchoring mechanisms can be introduced independently of the linking model, without having to alter existing link definitions.

Much of the work in this thesis has also explored ways in which adaptive modelling can be incorporated into an open hypertext model. Adaptive models have been presented which use feedback from the user to modify anchor definitions and linkbase trees. This is a very powerful development which allows the open hypertext system to respond to the needs of the user community. This helps to address some of the tailorability issues involved in hypertext design, by helping the author construct a coherent hypertext. While many open hypertext systems may support some level of extensibility, an adaptive model attempts to automatically manage these changes. The adaptive model in the HIPPO system guides the author and user, and tries to suggest useful modifications to the underlying hypertext.

The HIPPO model addresses some of these requirements which have emerged from open hypertext research. In particular, the HIPPO model develops the anchoring and linking abstractions, and offers an interesting application of adaptive hypertext techniques. However, the current model ignores some of the key requirements of open systems, most notably, *openness*, *interoperability* and *computation*.

The HIPPO system does provide some separation of linking and anchor definitions from the node information. Some aspects of the HIPPO model are implemented using separate tools and utilities, such as the tools for manipulating linkbase trees. However, the prototype is still largely implemented as a monolithic application. Much of the hypertext functionality is embedded deep in the application itself, and cannot be used by external applications. HIPPO does not appear to be a truly *open* system, and does not offer any means of integrating with the external environment. The user cannot use HIPPO with their existing tools, and HIPPO does not have any notion of *link services*. Similarly, HIPPO does not have any significant support for *computation*, or offer any means of extending and customising the application.

This chapter shows how the HIPPO prototype has been re-implemented using a widely distributed model. This new implementation – HIPPO+ – views the hypertext system as a collection of communicating processes and services, which can be located throughout the network domain. The advantages of this approach are discussed along with a number of issues which arise from this new design. A number of existing distributed architectures are also explored, which have been used to influence the design of the HIPPO+ system.

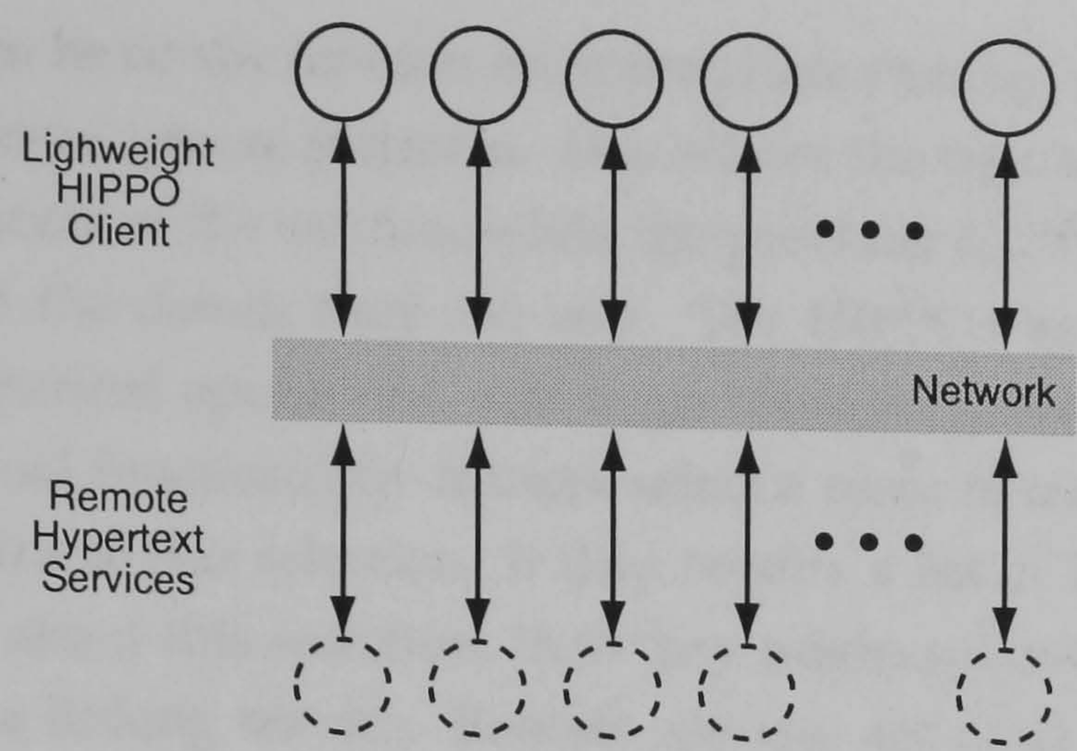


Figure 5.1: The HIPPO+ distributed model

5.2 The Distributed HIPPO+ Model

With the exception of some key applications (eg. Augment [Eng84a]), the majority of early hypertext projects focused on small-scale, localised implementations [App87, HMT87]. Hypertext applications were seen as tools to be used by the single user, to create small hypertexts and manage personal information spaces. The popularity of the personal computer encouraged the development of tools aimed at the individual working in isolation. However, these monolithic tools are unable to meet the demands of modern computing tasks [Mal91]. The next generation of hypertext applications need to cater for much larger hypertexts with many users and millions of documents. A hypertext must span many networks and platforms, seamlessly embracing users and their tools. Leggett refers to this new vision of hypertext as *hypermedia-in-the-large* [LS94].

The HIPPO+ system attempts to re-implement the ideas which have been presented in previous chapters using a distributed model. The hypertext system is viewed as a collection of small components and services which can be distributed throughout the network (figure 5.1). A simple, lightweight client acts as the main interface to the system, but the real intelligence of the system is widely distributed throughout the domain. Each component implements some hypertext abstraction or service which can be invoked by the client when required. Typical remote services might include *view node, traverse link, select anchor* or *load linkbase tree* etc. These were all operations which were implemented in the previous HIPPO prototype, but were embedded deep inside the application.

For example, as users move through a hypertext, they may wish to explore a particular node. The previous prototype would have implicitly loaded this node from a previous file, while the new HIPPO+ model requires the user to invoke a remote service to perform this task. This service may load the node from a local file, or perhaps retrieve the node from remote storage engine etc. If the user wishes to

view the node, then he or she invokes an appropriate *viewing service* which may be implemented on some remote platform. This allows the user to choose the appropriate implementation of the service, while the previous HIPPO system fixed this behaviour and hid the details from the user. The HIPPO+ system also identifies other common hypertext operations, and maps these on to remote services which implement the actual functionality. If users select a piece of text, they invoke a remote service to retrieve this selection. If they require a list of links which provide more information about this selection, then they might submit the selection to an appropriate remote linking service. Remote services are used for all operations – creating new links, deleting nodes etc.

The HIPPO+ model distributes the entire functionality throughout the network domain instead of implementing each operation locally in a single application. This provides a robust architecture, and allows services to be located at the most appropriate location. Each of these services can be invoked by other existing applications, which allows the HIPPO ideas to be reused in other tools and utilities. The user is also not limited to the set of operations which are initially made available by the developer. New services can be implemented in the network domain, and the user can invoke these if and when required. This provides a level of extensibility which was not available in the previous HIPPO system. Section 5.3 discusses the advantages of this HIPPO+ implementation in more detail. Chapter 6 discusses a proposed adaptive model which can use feedback from the user to modify the functionality of the system.

5.3 Advantages Of A Distributed HIPPO Model

A distributed architecture offers many advantages over conventional software designs. Distributed systems spread the computational load throughout a system, to reduce the load on any one node. Users are encouraged to share information between distributed users, and have access to a larger information domain. Many of the redundancy problems associated with local systems are also avoided, as the user has access to the original version at all times. Distributed systems offer a durable and robust architecture which can handle local system failures – the system has no single point of failure. Also, a widely-distributed model can make use of remote resources – users are no longer forced to make copies of large databases. Computations can be executed on high-speed platforms and services can be maintained at the point of origin.

A distributed model also has particular implications for the hypertext community. Section 2.2 discussed some of the hypertext projects which have incorporated distributed techniques. Nodes no longer need to be maintained locally and can be accessed remotely over a network. Queries and computations can be executed re-

motely, and have access to much larger resources. Hypertexts are no longer limited by the constraints of the single machine, but can span large networks with many hundreds and thousands of platforms. The distributed model used in HIPPO+ also attempts to address some of the open limitations which were identified in section 5.1. These are summarised below:

5.3.1 Scalability

Distributed systems are better suited to meet the scalability problems associated with large-scale hypertext systems. Distributed models avoid many of the storage limitations associated with localised systems, by distributing node and link information throughout the domain. Distributed models also resolve many of the redundancy problems which emerge when users are forced to make local copies of information. However, the most significant advantage of a distributed model, is that the system *itself* is distributed. The HIPPO+ system distributes the functionality throughout the network, so has no single point of control. This avoids many of the bottlenecks associated with conventional implementations, and allows the system to grow at will. New components can be added easily and efficiently, which can then be merged into the overall system. This idea of extending the system is discussed later in section 5.7.

5.3.2 Openness

The previous implementation of the HIPPO model used a single, monolithic Acrobat plug-in to implement the ideas in this thesis. While this demonstrated the usefulness of the HIPPO research, it was not implemented using an open architecture. Other users could not incorporate these HIPPO ideas into their existing environment, and the HIPPO prototype did not provide a means of integrating with other applications. The HIPPO+ system implements the ideas in HIPPO using a collection of distributed components. These each implement some abstraction or provide a particular service, and are made available to the entire network domain. Other applications can then make requests to these services, and incorporate the HIPPO+ functionality into the existing environment. Existing tools can invoke remote HIPPO+ services by supporting a minimal RPC protocol, so can make use of HIPPO ideas (eg. linkbase trees, fuzzy anchor tools etc).

5.3.3 Distribution

Chapter 4 showed how the *linkbase tree* model could be extended with a simple distributed architecture. Each node in a linkbase tree definition can be located remotely, and retrieved on demand. The original HIPPO implementation provided a

collection of Java classes which supported this remote access. Similarly, the adaptive model was implemented as a centralised server which could be accessed by remote clients. However, while this level of distribution can be useful it does not constitute a truly distributed system. Section 2.2.1 discussed the different degrees of distribution which have been adopted by hypertext applications. The HIPPO+ system aims to distribute *every* service in the current HIPPO system, so that the entire functionality is distributed throughout the network. Each operation in a hypertext is implemented as a small, lightweight component which is invoked remotely. Services can be replaced; users can choose alternate implementations; the system can continue to operate even if some services fail etc. This should provide a robust and scalable hypertext environment.

5.3.4 Heterogeneity

One of the main problems confronting hypertext developers is the difficulty of supporting diverse formats and heterogeneous node contents. The HIPPO+ system uses remote services and components to support key hypertext operations (*view node, create link* etc). This means that new formats and content types can be integrated into HIPPO+ quite simply, by providing an appropriate remote service. For example, new node types can be viewed if the developer provides a suitable browsing tool, or perhaps implements some of the *fuzzy anchoring* ideas for this new content type. The HIPPO+ client can then simply invoke these new services as if they were part of the existing hypertext application. The HIPPO+ model does not restrict the user to particular node formats or *hard-wire* any services into the system. Section 5.7 discusses how the HIPPO+ system is used, and how the system can grow to include new services.

5.3.5 Interoperability

Open hypertext research has developed the notion of a hypertext *link services layer* which replaces the traditional idea of a monolithic application. The OHS defines a protocol which external applications must support to integrate with the hypertext service. The distributed HIPPO+ model defines an interface for each remote service, which dictates how the component is invoked. The HIPPO+ client can then use this interface to access remote services. This idea of interfaces between clients and services is fundamental to many distributed architectures and is discussed further in section 5.5.2. The RPC interface used in HIPPO+ is shown in section 5.6.4.

5.3.6 Extensibility

As discussed previously, the new HIPPO+ implementation adopts a model based on remote services. A number of common operations are identified, and these are

mapped on to remote services which implement these functions. However, developers can make additional services available in the network domain. The user can then invoke these services if required, to augment the existing set of hypertext services. New data formats can be incorporated into HIPPO+ by making suitable services available, and multiple linking models can be supported by choosing remote services with different link semantics. The user is not limited to the initial set of remote services which were suggested by the author, but can augment them with additional hypertext operations. This computational model is developed further in section 5.7, which allows user to select arbitrary services. This provides an extensible environment which allows the user to determine the functionality of the system.

5.3.7 Computation

Section 2.1.3 discussed the importance of supporting computation in an open hypertext system. Links should not always be implemented as simple navigational relations, but may need to support more computational elements. HIPPO+ implements each hypertext operation as a remote service, so that *every* hypertext operation resolves to some remote computation. Computation is central to the hypertext model. Each operation – *view a node, load a linkbase, follow a link* – results in the execution of some remote process. In this way, computation is not only supported, but is fundamental to realising the ideas in the HIPPO model.

5.3.8 Tailorability

The previous chapters have shown how an adaptive model has been incorporated into the HIPPO system, and has been used to modify anchor and link tree definitions. Many open hypertext systems support some level of tailorability, and allow the user or author to alter the hypertext in some way. However, these systems do not *support* and *guide* these changes, or suggest how the system should adapt. There is a clear difference between *adaptive* and *adaptable* systems; for example, the Hyperform system [WL92] provides support for extensibility, but does not attempt to decide *how* the system will be adapted. An OHS should not only *allow* tailorability, but should help manage these changes and automatically adapt the hypertext environment. Chapter 6 describes how an adaptive model has been incorporated into this new distributed HIPPO+ system. The inclusion of an adaptive model represents a fundamental development, which allows the system to *automatically* develop and adapt to changes. The functionality of the system is no longer fixed, and the boundaries of the system can change to reflect the needs of the user.

5.4 Existing Distributed Architectures

The HIPPO+ system attempts to re-implement the HIPPO ideas presented in this thesis, using a distributed model. The previous section discussed the benefits of a distributed approach to software design, and explained the advantages of the HIPPO+ prototype over the existing HIPPO system. Similarly, software developers have long recognised the advantages that a distributed model can offer, and have used distributed techniques in the implementation of applications, database systems etc. This has led to the development of a number of general-purpose distributed architectures and frameworks which can be used for developing distributed systems. These have received particular emphasis with the success of the World Wide Web and the move towards component-based software (eg. JavaBeans [JB], ActiveX [Act] etc). This section describes some of the common approaches used to develop distributed software applications. This discussion begins with some of the low-level mechanisms for inter-process communication (IPC) such as sockets, RPC calls etc. Section 5.4.2 introduces the idea of *distributed frameworks* which provide more abstract services for building large-scale, distributed applications. Finally, Section 5.4.3 closes with a discussion of *compound documents* which provide container models for building component-based software systems.

5.4.1 Inter-Process Communication

The functionality of distributed systems is located throughout a network. Function calls and operations are no longer (necessarily) executed in the same address space or on the same platform. This means that all distributed models must provide some means of invoking remote services, and collecting results. This support for inter-process communication (IPC) can be implemented using low-level mechanisms such as socket interfaces, shared memory etc. However, these communication methods do not shield the developer from the details of each remote platform, and can be inadequate for developing distributed applications. This has led to the introduction of the *Remote Procedure Call* (RPC) which has been discussed briefly in Chapters 3 and 4. The RPC model is central to many distributed environments and has been used in the HIPPO+ system.

The remote procedure call offers an abstract, higher-level mechanism for inter-process communication. RPC hides the details of remote execution, and allows the developer to treat remote functions as if they were implemented locally. When a remote function is encountered, control is passed seamlessly to the RPC environment, which invokes the remote service, and waits for a result (figure 5.2). RPC implementations provide reliable and robust support for IPC, and handle complex issues such as error-handling and addressing in a portable fashion. RPC libraries also provide a portable representation for exchanging data between machines, and

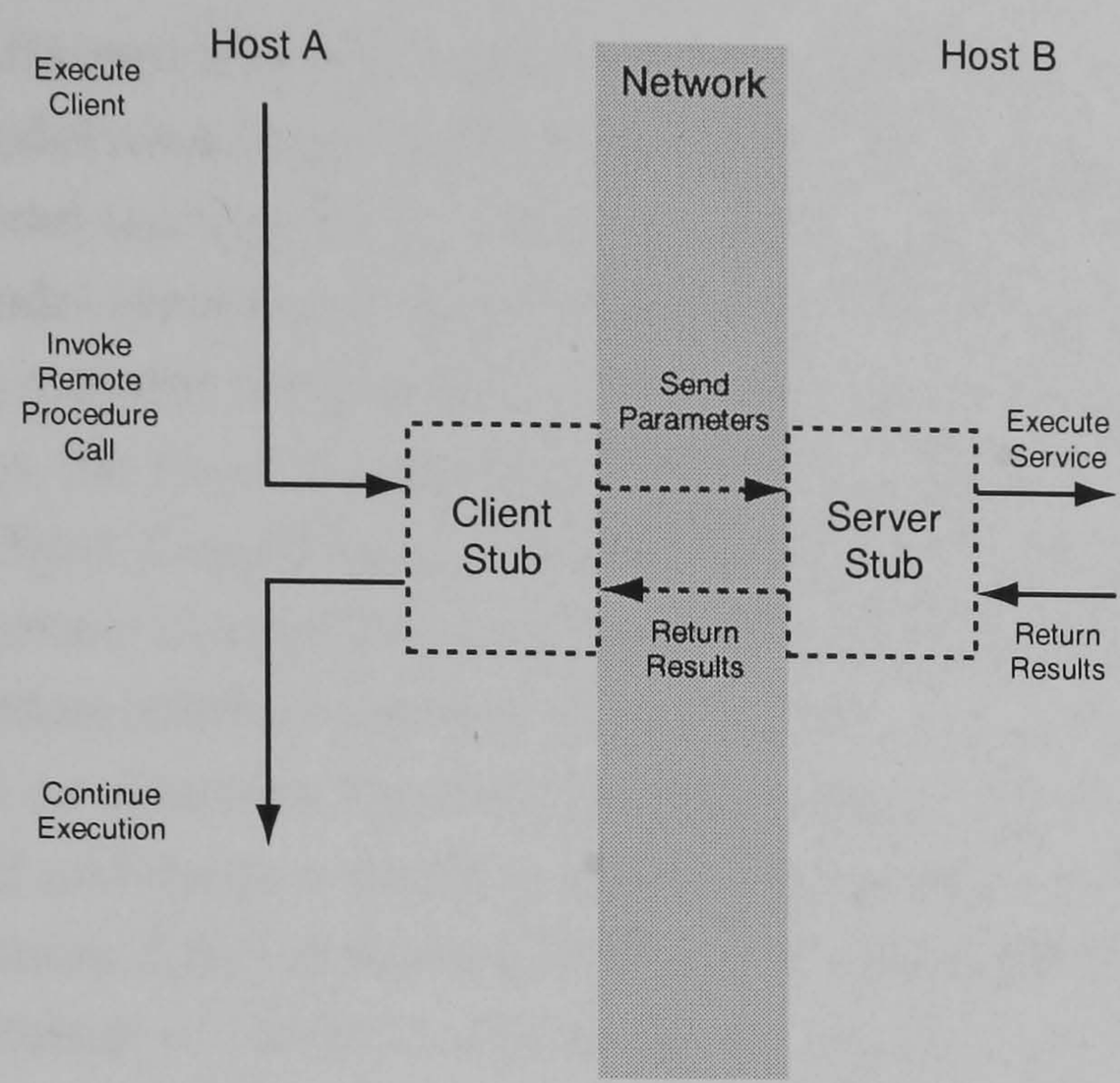


Figure 5.2: The Remote Procedure Call (RPC)

may support additional services such as authentication (eg. DES [DES93], Kerberos [KN93] etc).

RPC mechanisms provide an effective model of inter-process communication, and form the basis of many distributed environments. For example the DCE environment (see section 5.4.2) defines a secure RPC transport layer, and this has been incorporated into the DCOM component framework (section 5.4.2). Separate implementations of RPC libraries are also widely available (eg. ONC-RPC [Sun95a] which is used in HIPPO+ and RMI [SM97] which supports remote invocation for the Java environment).

5.4.2 Distributed Frameworks

While RPC mechanisms can be useful for implementing distributed inter-process communication, they do not offer many of the services which are necessary for supporting distributed software development. A distributed system needs directory services for managing remote resources; transaction management; concurrency control; security; licensing etc. This has led to the development of general-purpose distributed frameworks which can be used to develop distributed systems.

The *Distributed Computing Environment* (DCE) [DCE] developed by the *Open Software Foundation* (OSF) supports a secure RPC model and provides a transparent distributed file system. DCE offers directory services based on the X.500 [Uni93] and DNS [SP82] standards, which can be used for managing collections of remote resources. The DCE RPC layer has also been incorporated into the Microsoft *Distributed Component Framework* (DCOM). The DCOM model emerged from the OLE

and COM frameworks which form the basis of the Windows operating system. The DCOM model offers some distributed services, but also provides a compound document model (section 5.4.3) which is used to support ActiveX components. The DCOM model represents a move towards an object-based approach to software design using reusable components, and is discussed further in section 5.4.3.

Perhaps the most developed and flexible distributed model is offered by the *Common Object Request Broker Architecture* (CORBA) [Objc], developed by the *Object Management Group* (OMG) consortium. CORBA provides uses the idea of *Object Request Brokers* (ORBs) to service client requests, and provides additional object services such as directory services, security support etc [Objb]. CORBA is an open distributed architecture which is useful as a standard reference model, and is discussed in more detail in section 5.5. Many of the CORBA ideas have influenced the implementation of HIPPO+ and these are compared with the implementation of HIPPO+ in section 5.6.

5.4.3 Compound Documents

The distributed frameworks described previously allow the development of widely-distributed systems based on remote services. Each service supports an interface which can be used by any client to invoke requests. This client/server model has been developed by the Object-Oriented community, to support the idea of reusable software *components*. An application can be constructed from existing components, which can be reused instead of writing applications from scratch. Each component not only provides data, but also encapsulates some intelligent behaviour which can be reused in other systems.

The *compound document* has been developed to support this method of software design, and provides a means of organising and managing collections of components. The *document* itself is seen as the metaphor for integrating components. The compound document contains data and components which act on this data, and must provide storage facilities, layout services, data transfer operations etc (figure 5.3). The compound document becomes the container for each document, each application, even an entire desktop. More importantly, compound documents can also sit on top of distributed transport models such as CORBA, DCOM etc. These compound documents can then be used as the basis for all client/server, distributed computing.

Compound documents are just beginning to appear, and are starting to play an increasingly important role in software development. The OpenDoc model was developed by the CLI consortium [Ope], which integrated with the CORBA model. Similarly, Microsoft has developed compound document technology as part of their OLE and DCOM architectures [Act]. The JavaBeans [JB] framework also uses components to build software systems, and may incorporate compound document tech-

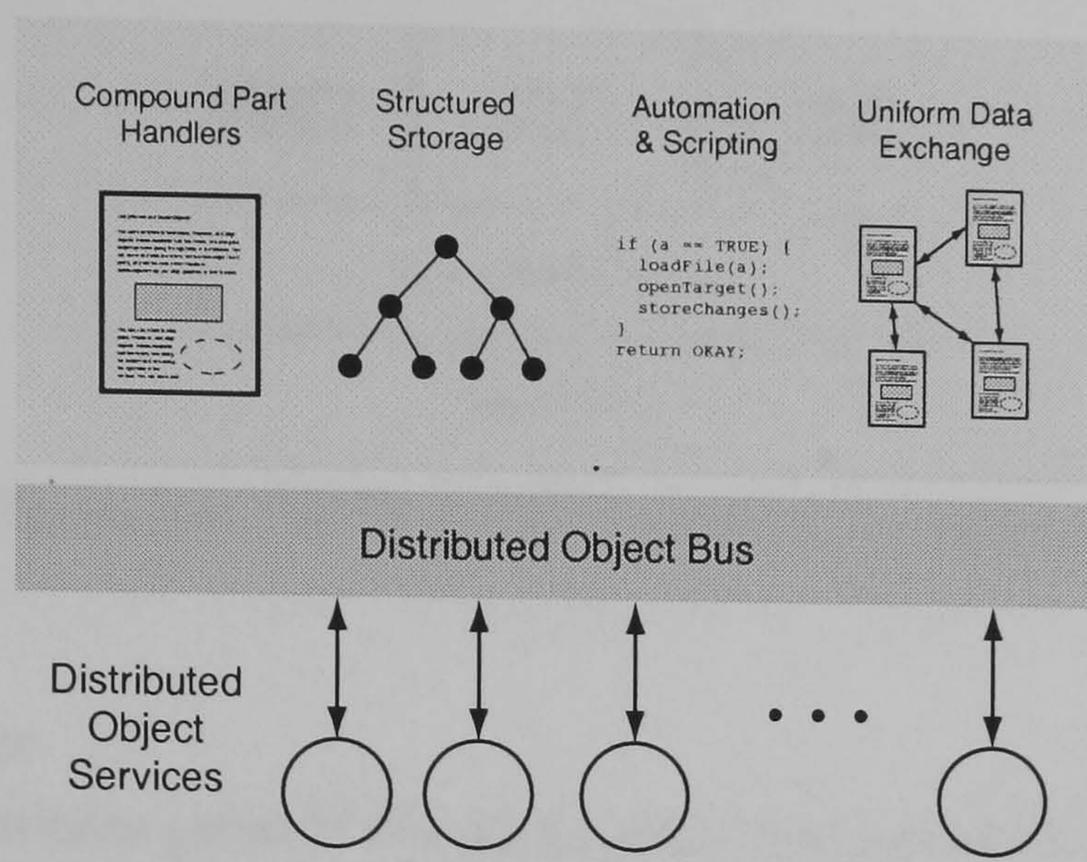


Figure 5.3: Compound Document Frameworks

nology. Also, the World Wide Web community is developing a *Document Object Model* (DOM) for managing web documents [WWW98b]. The compound document offers significant advantages, and Chapter 7 discusses how compound documents could be incorporated into future generations of the HIPPO+ model.

5.5 The CORBA Model

The previous section introduced the CORBA model which provides a framework for developing distributed applications. This model is becoming increasingly important and has influenced some aspects of the HIPPO+ system. For this reason, the model and the CORBA model is explored here in more detail. CORBA supports the large-scale distribution of components and objects throughout a network domain, and makes these services available to clients. Client-server requests are handled seamlessly, and the user remains unaware of the distributed topology. CORBA also supports a rich layer of object services which can be used to augment the distributed environment (naming services, transaction management, concurrency control etc) [Objb].

CORBA provides the distributed communications infrastructure of a distributed system, and forms part of the larger *Object Management Architecture* [Obj97]. The OMA defines five layers (figure 5.4):

- *Object Request Broker*
  This layer provides the actual distributed infrastructure of the OMA model, and is responsible for routing client-server requests. The ORB allows remote objects to be invoked in a transparent and portable fashion, and is discussed further in section 5.5.1.

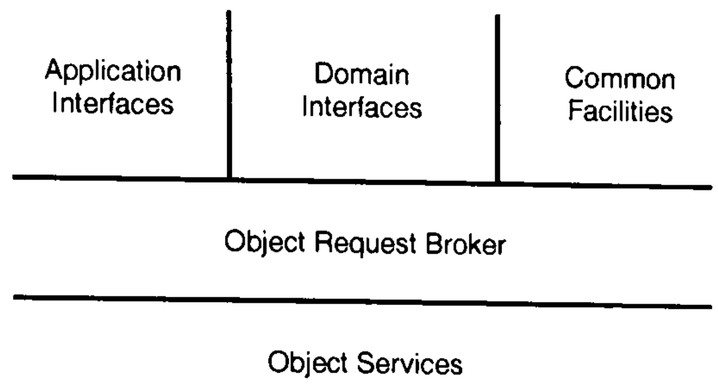


Figure 5.4: The Open Management Architecture

- *Object Services*
  These components provide the additional services which are needed to support a usable distributed environment [Objb]. These include directory services for managing collections of remote objects, trading services for locating services etc (see Section 5.5.3.

- *Common Facilities*
  The Common Facilities layer includes those application services which can be tailored to meet the specific needs of the user. These represent tangible, *"real-world"* services such as document printing, database queries etc [Obja].

- *Domain Interfaces*
  These services are specific to particular areas and application domains. They may combine object services and common facilities, but are intended for specific markets and domains.

- *Application Interfaces*
  This layer represents the highest abstraction, and includes intelligent, reusable component applications. These components are constructed from large numbers of components – object services, domain interfaces, common facilities – to build entire applications.

Readers are referred to the documentation of the OMG CORBA project [Obj97] which discusses each of these layers in more detail. However, the following sections identify some key areas in the CORBA model which are important for an understanding of the model, and have influenced the development of the HIPPO+ system.

5.5.1 Object Request Broker (ORB)

The *Object Request Broker* or ORB forms the basis for the CORBA model, and is responsible for managing all client/server communication in the CORBA system. The ORB provides seamless and transparent access to the distributed services, and hides the complexities from the client applications. Figure 5.5 shows how the ORB is used

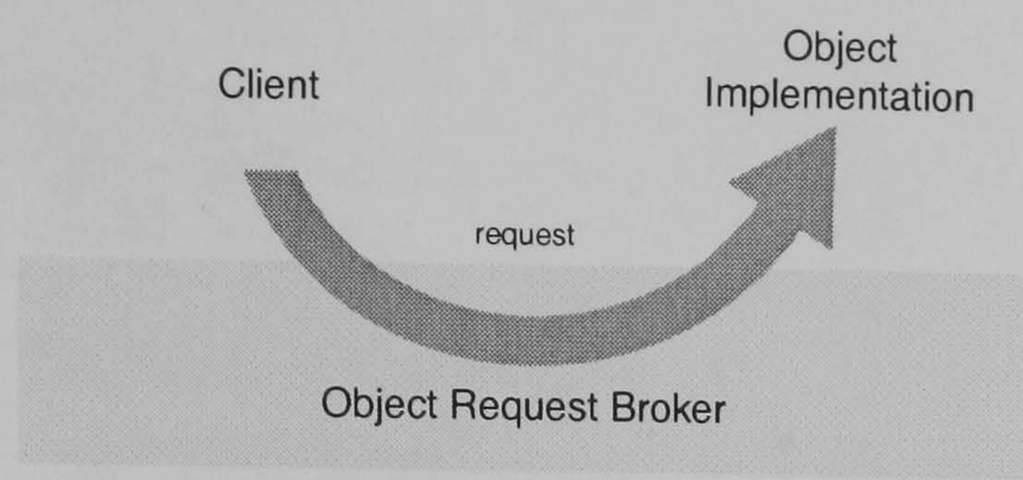


Figure 5.5: The ORB in the a client-server transaction

in a typical transaction. A client makes a request to a remote service, which is then passed to the ORB. The ORB is responsible for finding the appropriate object which can implement this request, and passes the operation parameters to the remote object. The method is invoked, and the ORB waits for the results. In this way, the client is unaware of the location of the service, the operating system or even the language which is used to implement it. CORBA systems can also support multiple ORBs and the standard defines the *Internet Inter-ORB Protocol* (IIOP) [Objc] for communicating between them.

5.5.2 Interface Definition Language (IDL)

Each remote object in a distributed CORBA system implements a collection of methods to support a particular service. The CORBA standards is an *open* model which allows each component to be implemented using *any* language, running on *any* platform. CORBA developers are not limited to a particular implementation platform, and can choose the appropriate language for the particular domain. This means that each service must have a platform-independent, language-independent method of describing the operations which it supports. This is the role of the *CORBA Interface Definition Language* (IDL) [Objc].

An example IDL definition is included in figure 5.6, which defines an interface for a simple library service. The IDL definition provides an abstract definition for each operation, using standard CORBA data types and semantics. IDL isolates the CORBA environment from the details of each service implementation, so that each client need only be aware of the operations that a service provides without needing to know any implementation details. Each language which is to be incorporated into the CORBA system must provide a set of mappings between the specific language and the language-independent IDL.

The conventional invocation model requires each client to know about the services which it is going to use. These static interfaces are generated automatically from IDL interfaces, and compiled directly with each client. Any type checking and integrity checks are performed at compile time, and the method calls are more effi-

```
module library {
  interface book {
    long getPages();
    string getTitle();
    string getISBN();
  };

  interface dictionary : book {
    attribute string language;

    string lookupWord(string word);
    void setLanguage(string lang);
  };
};
```

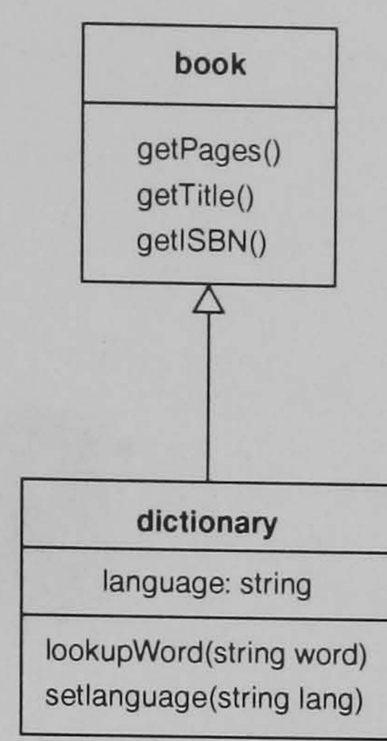


Figure 5.6: Interface Definition Language (IDL) example

cient. However, CORBA also allows services to store IDL interfaces in an *Interface Repository*, so that clients can query and invoke services dynamically. Clients can retrieve IDL definitions at runtime, and dynamically invoke *arbitrary* services without having to know the details beforehand. Dynamic invocation is more flexible, but does incur some efficiency overheads. The ORB model can now be refined to support dynamic invocation, as shown in figure 5.7.

5.5.3 Object Services

The ORB and IDL models described previously implement the basic communication between client and server objects. This provides a portable and transparent distributed environment, but ignores many of the more complex operations which are needed in a usable distributed system. These services are addressed in the *Object Services* layer[Objb], and the HIPPO+ model does attempt to implement some of these (see later in chapter). The current *CORBAservices* specification defines the following services:

Naming Service	*Event Service*
Persistent Object Service	*Life Cycle Service*
Concurrency Control Service	*Externalisation Service*
Relationship Service	*Transaction Service*
Query Service	*Licensing Service*
Property Service	*Time Service*
Security Service	*Trading Service*
Collections Service	

The *Naming* and *Trading* services have been especially influential in the development of the HIPPO+ implementation. Distributed systems can grow to support enormous numbers of components and remote services. The environment must

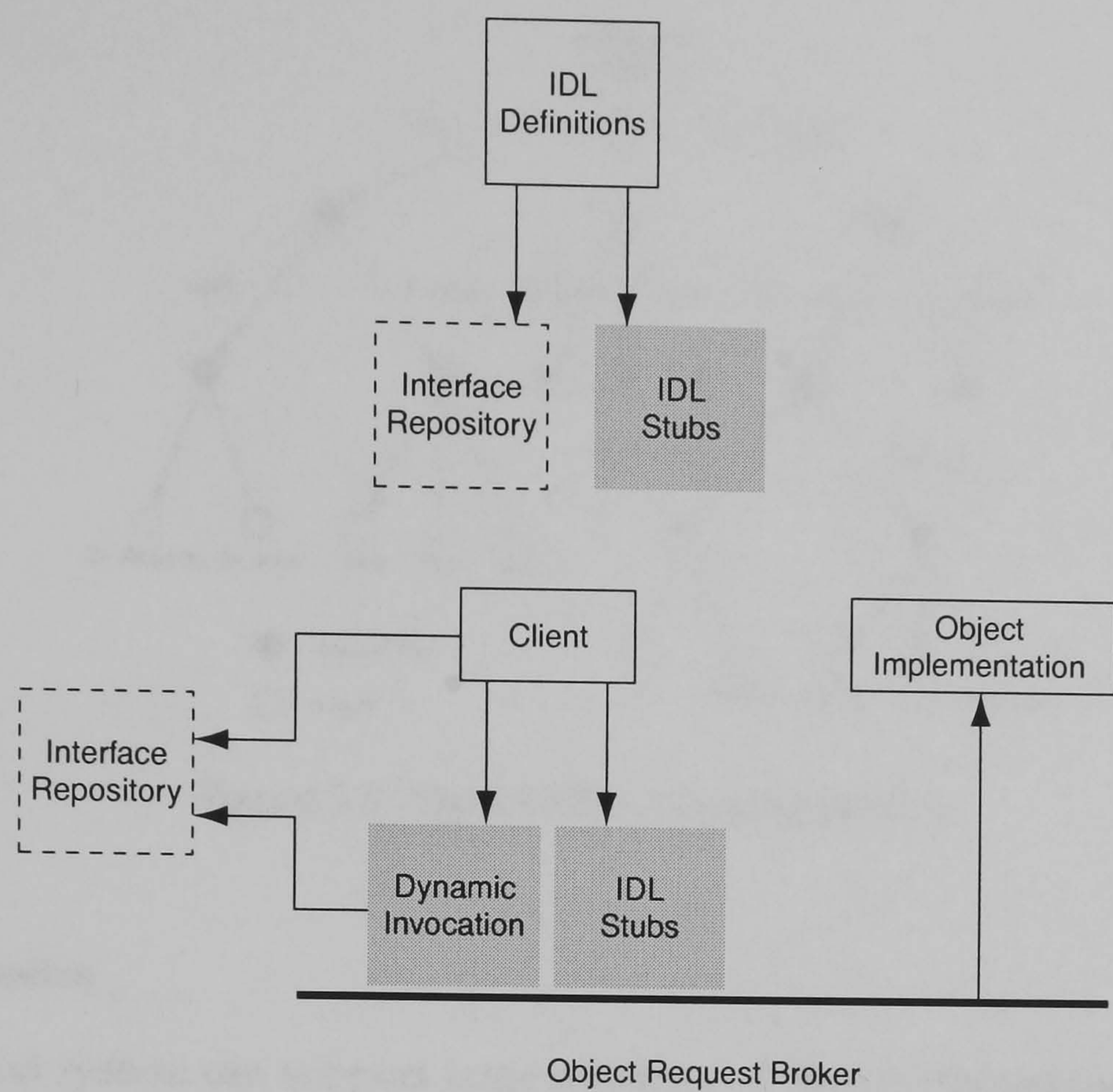


Figure 5.7: Static and dynamic invocation in the ORB

provide some means of managing this complexity, and allow users to locate desired services. The *Naming* and *Trading* object services are described here in more detail, and the OMG CORBA documentation also contains further information [Objb].

Naming Service

Each object in the CORBA domain is assigned a unique identifier, which provides an unambiguous reference to that service. Each client must include an object identifier in every request that it makes to the ORB. However, a large-scale distributed environment can contain many hundreds of components and services, and the user cannot be expected to keep track of all these identifiers etc. The *Naming* service provides a service for managing large collections of services, so that components can be located more easily. The *Naming* service allows object references to be arranged into hierarchical graphs, and supports a *name-to-object* binding mechanism (figure 5.8). These graphs can be widely-distributed throughout the network domain and combined with other naming contexts to produce *federated* graphs. Clients can then issue requests to the *Naming* service, to find objects with particular names, aliases, properties etc. This service can also be combined with other object services such as the *query* and *relationship* services, to provide a more flexible service.

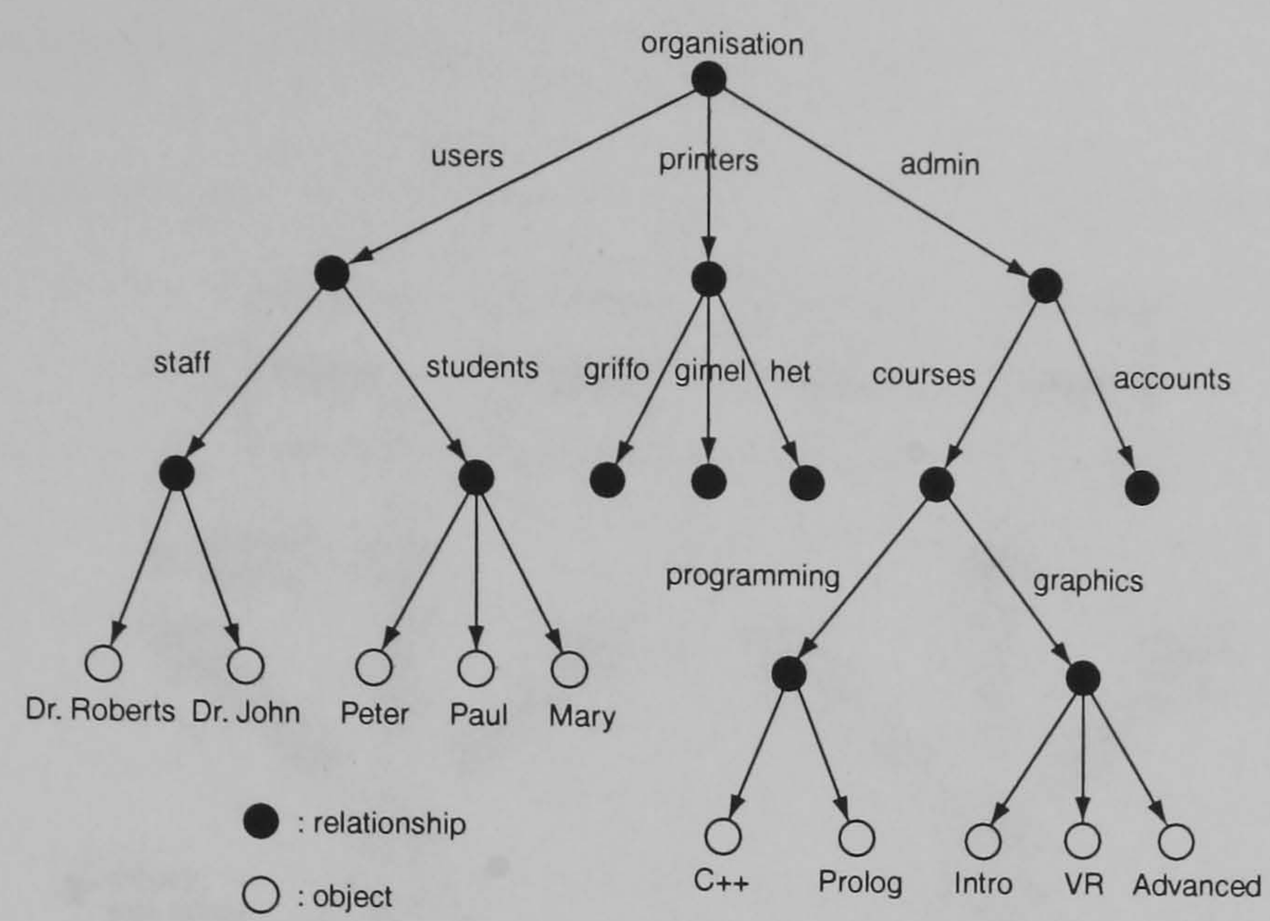


Figure 5.8: The CORBA Naming Service

Trading Service

A distributed system can support large numbers of objects throughout a network, supporting a diverse range of services. A scalable distributed model must provide some support for managing and locating these objects. The earlier discussion introduced the *Naming* service which allows objects to be arranged using directed graphs. This supports a *name-object* binding system, so that objects can be located using aliases. However, this simple service may not be sufficient for the effective management of large systems, which contain ever increasing numbers of objects and components.

The *Trading* service provides a more flexible system for locating objects. The trader object allows objects to advertise their capabilities and describe the services they offer. Clients can then ask the trader for those objects which match a particular service description. Figure 5.9 shows how these service descriptions can be *imported/exported*.

The trading service is more flexible than the naming service, and can support diverse trader implementations, each with different trading policies. A trader object can recommend particular service entries based on location, network traffic, implementation etc. A distributed environment can also support multiple traders, which each manage a particular domain. Traders can then combine service domains together, to provide a *federated* trading space (figure 5.10). The *Trading* service offers a flexible means of managing object services, and is vital for large-scale distributed systems.

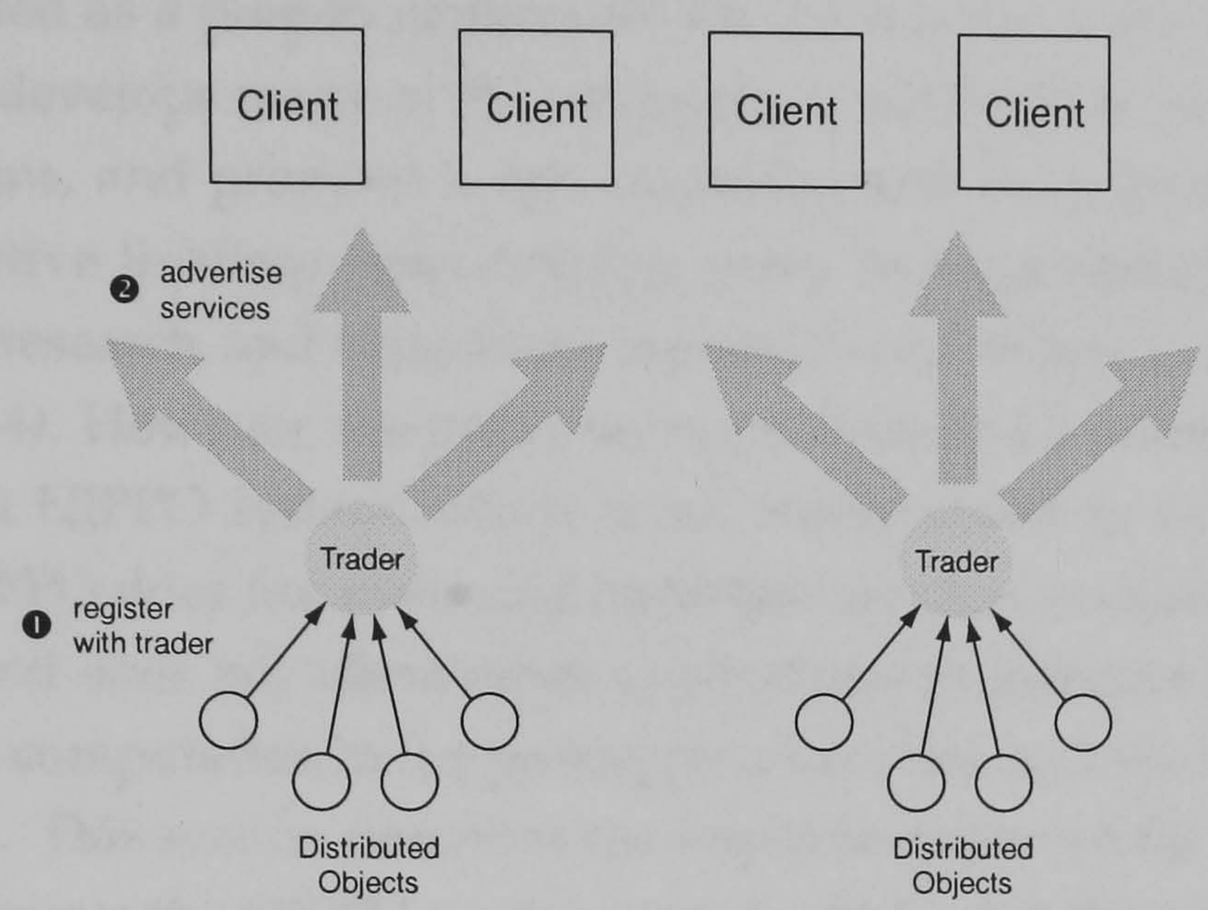


Figure 5.9: Advertising services using trader objects

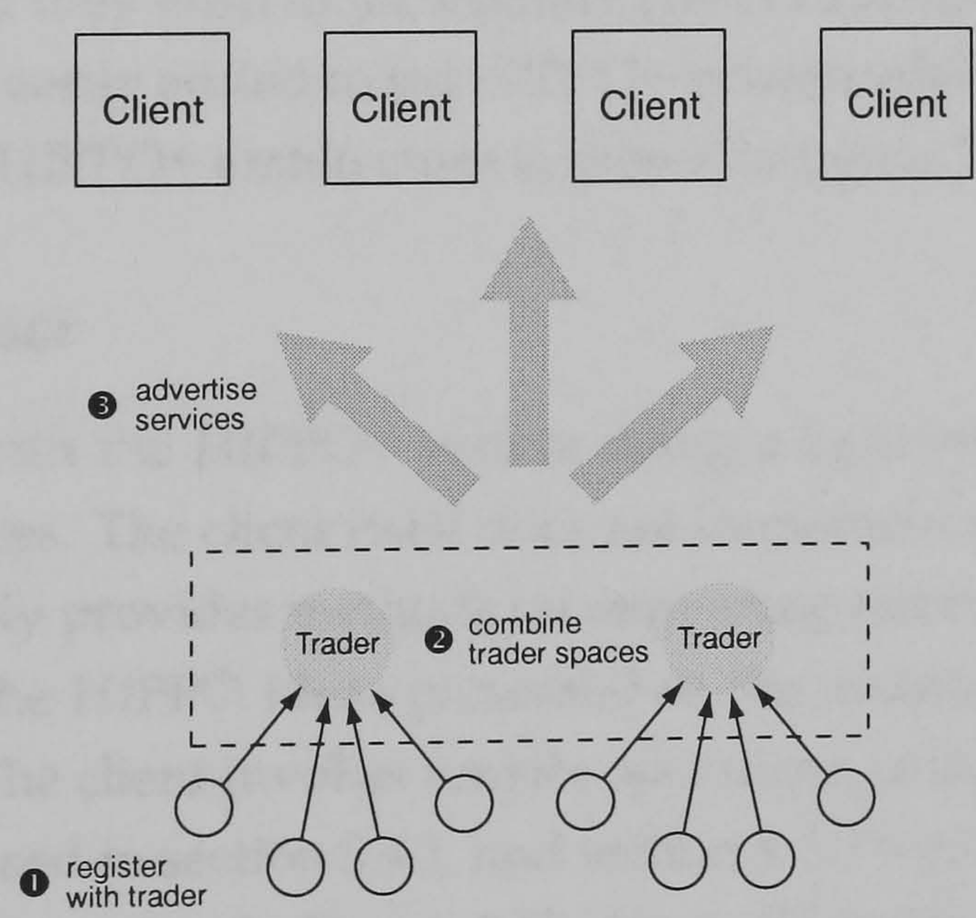


Figure 5.10: Federated traders

5.6 Implementing The HIPPO+ System

Section 5.1 discussed the current implementation of the HIPPO system which has been implemented as a plug-in component for the Acrobat software environment. This prototype develops many of the abstractions which have been used in open hypertext systems, and provides a rich anchoring and linking model. Fuzzy anchors and adaptive linkbase trees develop many of the abstractions common in open hypertext research, and support an expressive and adaptive environment (see Chapters 3 and 4). However, the discussion also identified a number of limitations with the current HIPPO system, which is not implemented as a truly *open* hypertext system. HIPPO does not make any hypertext services available to the external environment, and does not allow other applications to incorporate HIPPO ideas. The support for computation in the prototype is minimal, and the system cannot be easily extended. This section describes the implementation of the HIPPO+ system which re-implements the HIPPO model using a widely-distributed model.

HIPPO+ defines the hypertext system as a collection of widely-distributed, communicating services. The system identifies each key hypertext operation and abstraction in the HIPPO model, and implements each of these using a remote service instead of embedding the functionality deep inside a monolithic application. The users interact with HIPPO+ using a lightweight client which can be used to invoke these remote services. In this way, the functionality of HIPPO+ is no longer embedded in a single monolithic application, but is distributed throughout the network (see section 5.2). This is an open architecture which allows other applications to invoke these services if they wish to incorporate HIPPO abstractions and operations. New services can be easily added to the HIPPO+ system which allows some degree of extensibility. The HIPPO+ architecture is shown in figure 5.11.

5.6.1 Node Browser

The user interacts with the HIPPO+ system using a lightweight client which can invoke remote services. The client itself does not implement any specific hypertext operations, but simply provides methods for requesting remote services. The actual implementation of the HIPPO ideas presented in this chapter is supported by remote components. The client invokes remote operations using RPC calls – the RPC approach was discussed in section 5.4.1, and section 5.6.3 explores the HIPPO+ RPC implementation in more detail. Figure 5.12 shows the node browser client which has been implemented using C++ and the X11/Motif windowing system.

The client shows some details of the current hypertext node, and allows the user to invoke common hypertext operations – *view node, edit node* etc. The X11 windowing environment provides a simple means of distributing visual applications, so that remote viewers can display nodes on the user's display. The client supports a

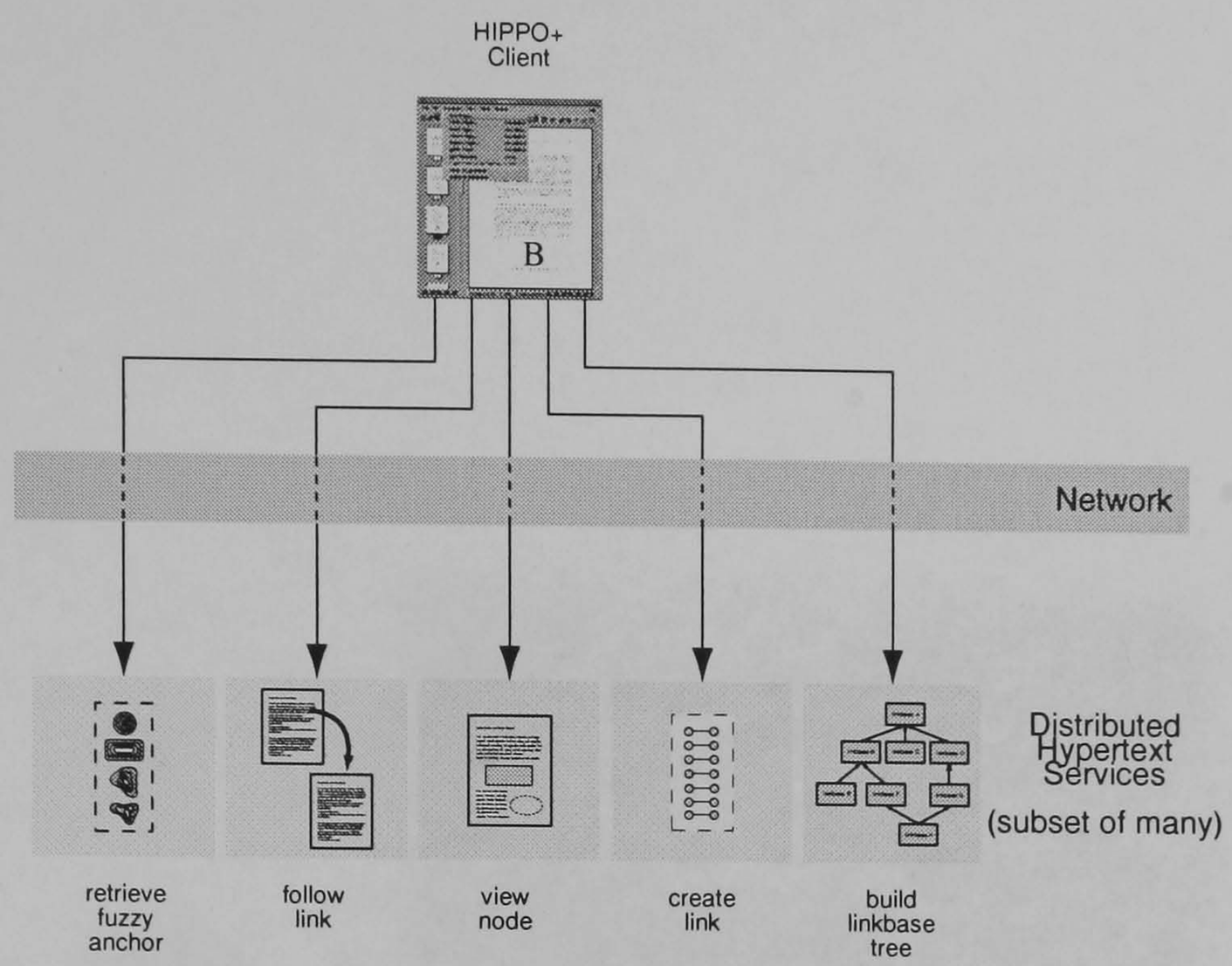


Figure 5.11: The HIPPO+ Architecture

rich set of other operations which are commonly used in a hypertext environment – *get current selection, make link, traverse link* etc, as well as services which are specific to the work in this thesis (fuzzy anchors, linkbase trees etc). These are all bound to buttons which invoke the appropriate remote service when clicked. Note that the actual node contents and link definitions etc, are all stored in separate buffers, and can be manipulated using the buffer tools in section 5.6.2. The execution of remote services is also explored in more detail in sections 5.6.3 and 5.6.5. The lower half of the browser shows the *Hypertext Component Hierarchy* (HCH) browser, which supports a hierarchical classification system for remote HIPPO services. This HCH directory service is used for maintaining large collections of services, and allows users to augment the *"button"* operations with other remote services. The HCH hierarchy and extended computational model is introduced in section 5.7.

5.6.2 HIPPO+ Buffers

Each time a user invokes a remote service, this performs some operation in the HIPPO model. Typically, each request will return some data associated with the hypertext or node – perhaps a collection of link definitions, the contents of a node, a linkbase tree definition etc. The HIPPO+ client stores the result of each service invocation in dedicated, special-purpose buffers. Each of these buffers represents a storage place for specific hypertext information (eg. node linkbase, the currently selected piece of text etc). Figure 5.13 shows an example of the browser which is used to select and manage buffers in the HIPPO+ system.

Each buffer is implemented as a local file, which is loaded on demand to reduce

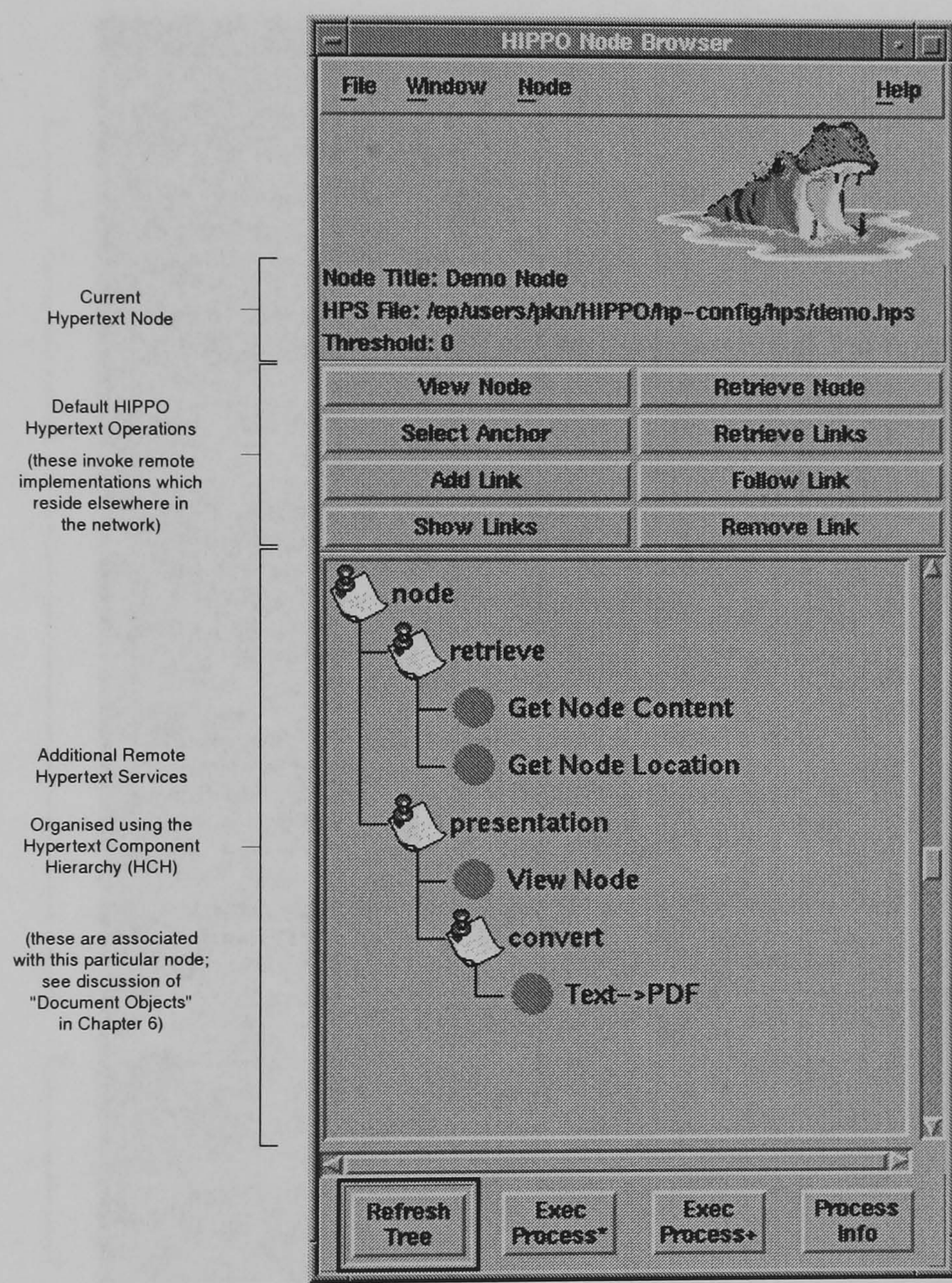


Figure 5.12: The HIPPO+ Node Browser

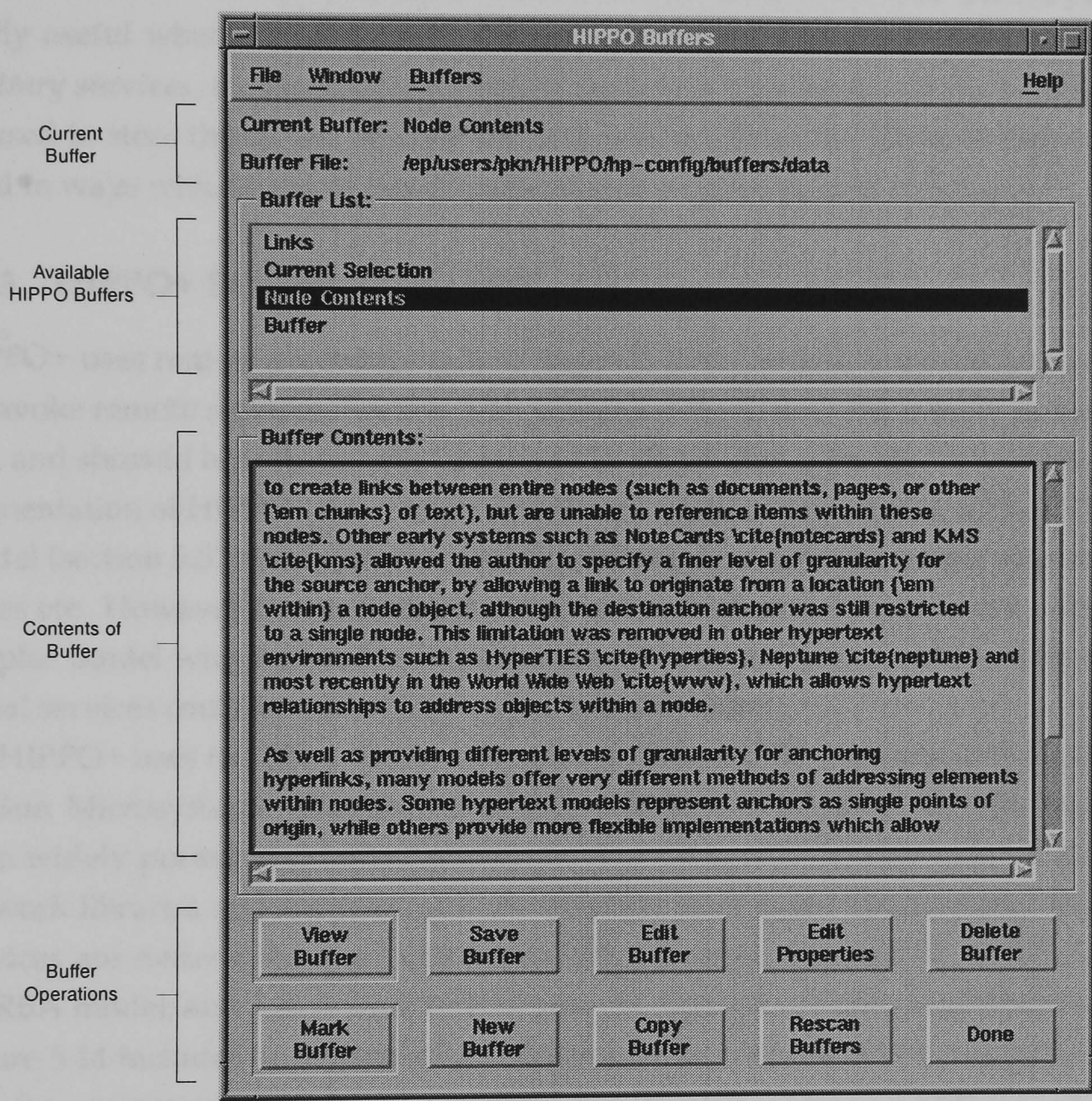


Figure 5.13: The HIPPO+ buffer browsing tool

the storage overhead in the client application. The buffer tool supports a diverse range of operations on each buffer – viewing, editing, copying etc. Some operations in the node browser (section 5.6.1) are automatically associated with particular HIPPO+ buffers. For example, when the user invokes the *retrieve node* operation, the client will invoke the corresponding remote service, and automatically place the results in the *Node Contents* buffer. Similarly, HIPPO+ has buffers corresponding to other common operations – *Retrieve links, Get Current Selection* etc. Users can also create their own buffers for specific tasks or to store temporary data. This is particularly useful when the computational model is extended to allow users to invoke *arbitrary* services, to augment the existing hypertext operations. These buffers can be used to store the results of these services, which allows the HIPPO+ client to be used in ways which were not anticipated by the original system developer.

5.6.3 HIPPO+ Services Using ONC-RPC

HIPPO+ uses remote procedure calls to support inter-process communication, and to invoke remote services. Section 5.4.1 discussed the idea of the remote procedure call, and showed how these have been used in distributed systems. The original implementation of HIPPO+ used a rich distributed environment similar to the CORBA model (section 5.5) which provided additional services such as trading objects, factories etc. However, it was decided that a system based on RPC libraries offered a simpler model with a finer-level of control over data exchange. In this way, additional services could be added to the system as required.

HIPPO+ uses the *Open Network Computing* (ONC) RPC implementation offered by Sun Microsystems [Sun95a]. This is available in the public domain, and has been widely ported to different platforms. The ONC RPC model includes a set of network libraries and support tools for building distributed applications. Remote services are defined using an interface definition language similar to IDL in the CORBA model, and this is used to generate the appropriate stub implementations. Figure 5.14 includes an example RPC definition for a service which supports three interface operations. The developer then uses the *rpcgen* tool to automatically generate the skeleton code for the remote object. Figure 5.15 shows the steps involved in developing client-server applications using ONC RPC tools.

Once the server applications have been implemented, these RPC services can be made available to the larger community. When each service starts up, it must register with an *rpcbind* daemon – this informs the daemon that the server is ready to receive requests, and explains which unique RPC identifier it will use. An *rpcbind* daemon runs on each machine and maintains a list of all currently running RPC services. Each client invokes a service by contacting *rpcbind* on the appropriate machine, which then passes the request to the corresponding service. In this way, each *rpcbind* daemon implements a very simple trading service which maintains objects

```
program MY_RPC_PROG {                    interface name
                                         (used to identify service)
  version MY_VERSION {                   version name
                                         (used to support multiple versions)
    int getMonth();
    string getDay();                     remote methods
    void printMessage();                 supported by this interface
  } = 1;                                 version number

} = 0x20000004;                          unique interface ID
```

Figure 5.14: An example RPC interface

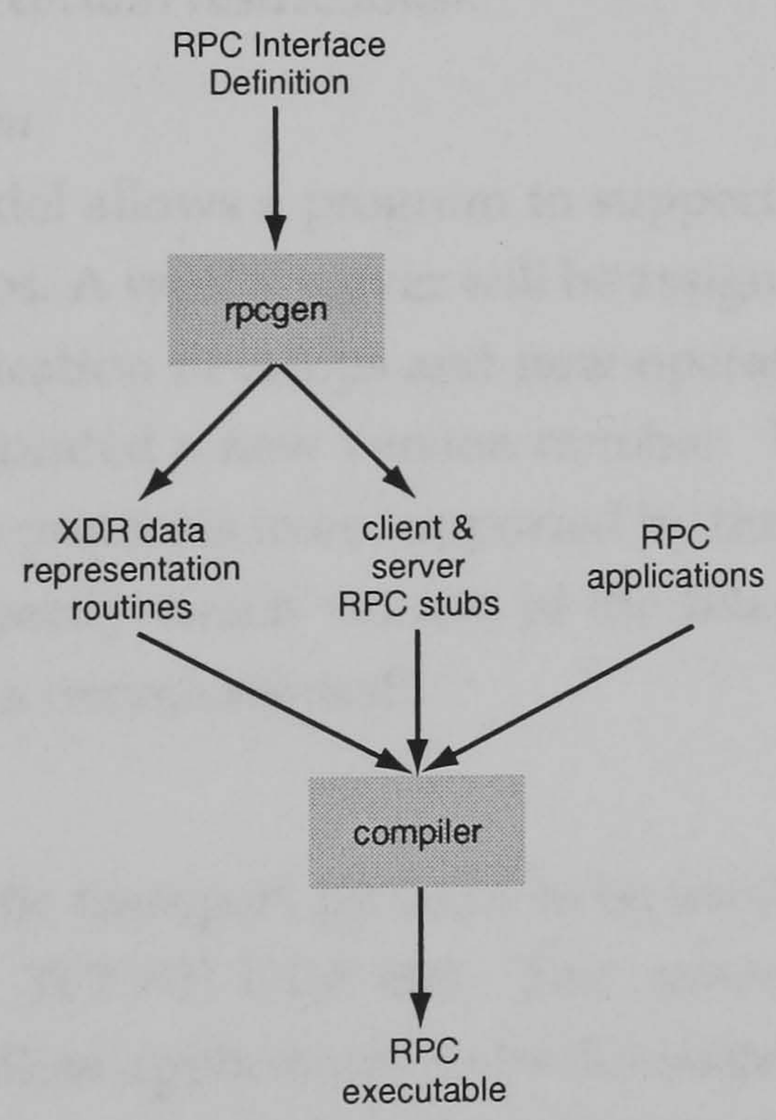


Figure 5.15: The RPC development cycle

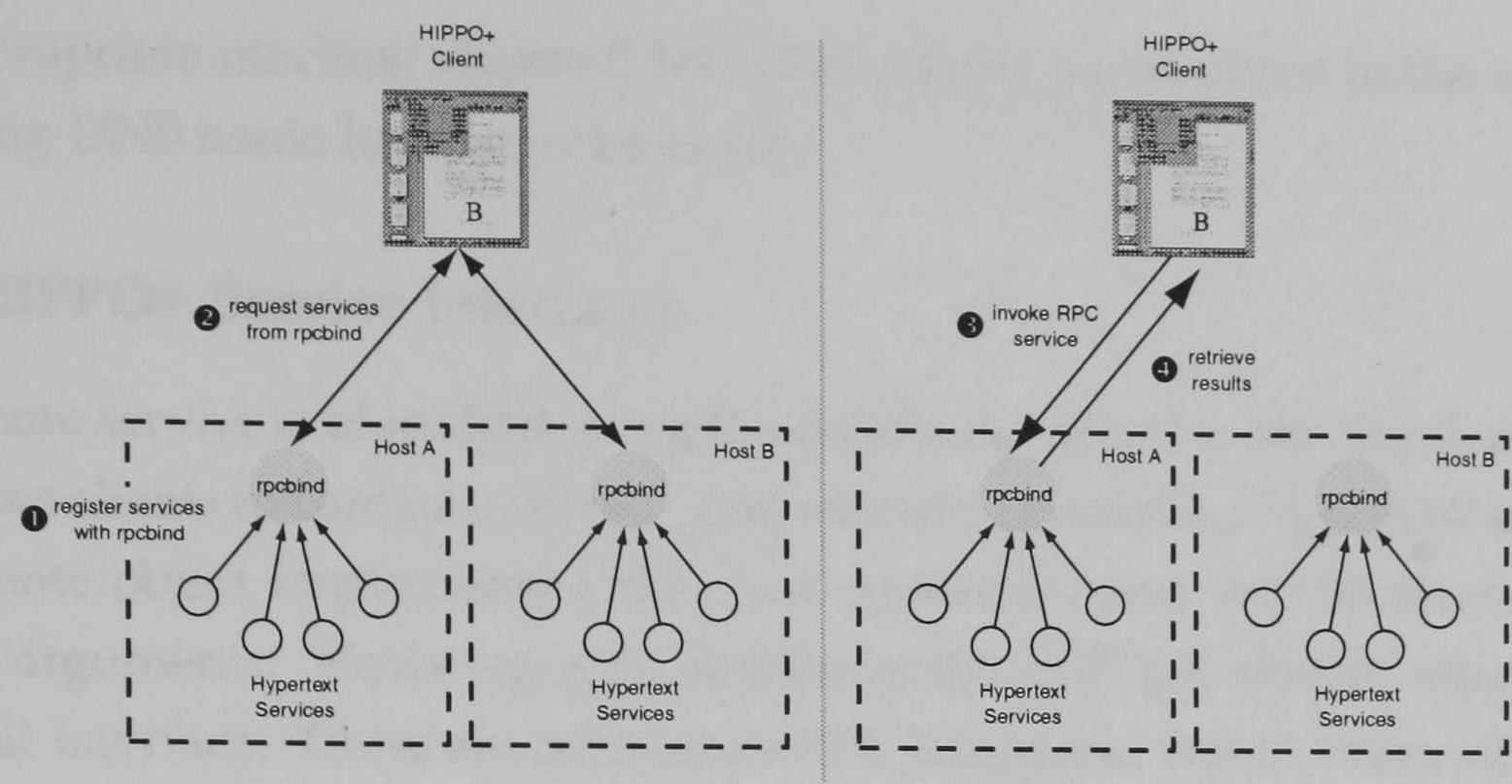


Figure 5.16: Using *rpcbind* in the ONC RPC model

on a single machine, and acts as an intermediary between clients and servers. Figure 5.16 shows how *rpcbind* fits into the client/server model.

Remote HIPPO+ services can be uniquely identified using the following entries:

- *RPC Program Number*
  RPC programs are identified using a uniquely assigned program number. These can be administered centrally by Sun Microsystems or can be allocated by the user (within certain restrictions).

- *RPC Program Version*
  The ONC RPC model allows a program to support multiple versions, as the application develops. A typical server will be assigned an initial version number 1. As the application develops and new operations are added, this new interface can be allocated a new version number. This idea of version numbers allow multiple protocols to be supported by the same server process. The client must then specify which version of the interface it wishes to invoke, each time it makes a remote request.

- *Transport*
  This allows a specific transport protocol to be used to communicate with remote services (eg. TCP/IP, UDP etc). The current implementations of the ONC RPC model allow applications to be developed *independently* of the transport type, although some types of applications may require a particular transport type to be used (eg. *connection-oriented* etc).

- *Host*
  Each client needs to know the name of the machine which is currently hosting the remote service. The client can then contact the *rpcbind* daemon on the

appropriate machine (figure 5.16). Host aliases are resolved in the usual way using DNS name lookup services etc.

5.6.4 HIPPO+ Service Interfaces

Each remote service is identified using the details described in the previous section. This allows clients to contact a service, and execute a particular hypertext operation. Each remote object implements a different operation, and will be invoked using different arguments. However, each service in the HIPPO+ system must support a minimal interface. These are defined as RPC functions, which every object must support.

Note: All operations return a status flag indicating success/failure, and a string containing any results

- `EXEC(string)`
  This is used to invoke the remote service, and the single parameter contains the arguments which are required by the remote object. Section 5.6.5 describes how this is combined with the HIPPO+ registry to support a simple form of dynamic invocation.

- `GET_INFO(void)`
  Returns a textual description of the service (usage, related services, additional information etc). Section 5.7.1 describes this query interface in more detail.

- `GET_NAME(void)`
  Returns an *alias* which can be used to refer to the service. This is defined by the author, and is used in the trading model to hide the RPC identifiers from the user (section 5.7.2).

- `GET_HCH(void)`
  Returns a hierarchical path which suggests where the service should be classified in the HCH hierarchy (see section 5.7.3). Path components are separated by "/".

- `GET_PROTOCOL(void)`
  Each service in the HIPPO+ network domain implements a different hypertext operation which can be invoked by the user client. Each operation may require different arguments, and is invoked in a slightly different way. This `GET_PROTOCOL` operation returns a *template string* which describes how the hypertext service must be called. This information can then be used when the `EXEC` operation is called. This simple form of dynamic invocation is discussed in section 5.6.5.

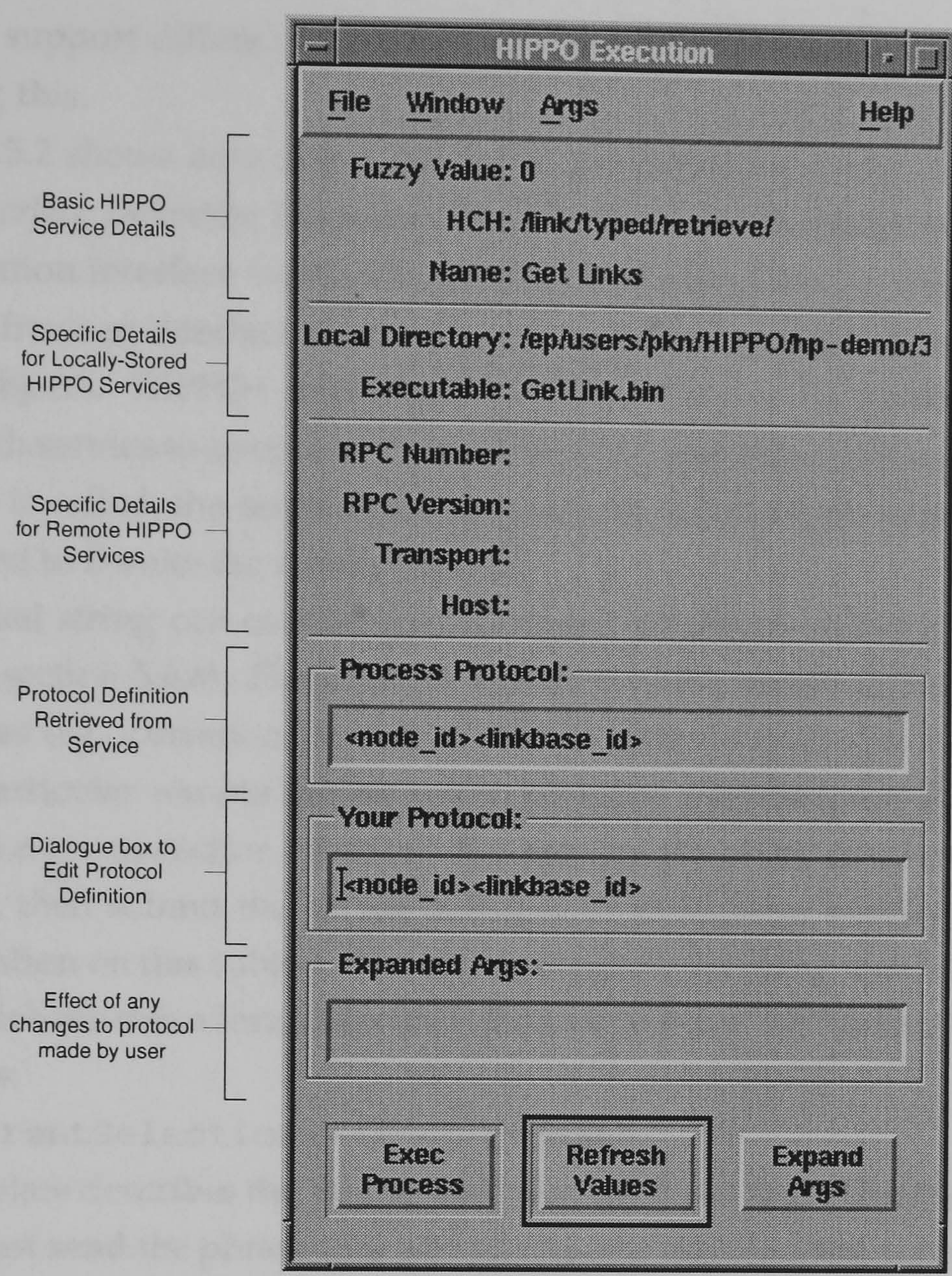


Figure 5.17: The Execution Manager

5.6.5 Execution Manager

Figure 5.17 shows the *Execution Manager* which is used to control how services are invoked. HIPPO+ allows services to be located remotely, or they can be stored on the user's local machine. These local services can be important for some operations which only have meaningful semantics when invoked locally (eg. *store node in local file, browse file system, check available disk space* etc). Local services are identified by their location in the user's filesystem and the name of the executable program. Remote services require a unique *RPC identifier, transport protocol* etc (see Section 5.6.3 for details of RPC in HIPPO+).

Each local/remote resource in the HIPPO+ environment implements a different service, and supports a different method of invocation. For example, an object which is responsible for viewing node contents may need to be invoked with the location of the particular node. Another service which creates a new link may require more arguments, and will be invoked in a different way. It is clear that different

services will support different interfaces, and the HIPPO+ model needs some way of expressing this.

Section 5.5.2 shows how object interfaces are supported in the CORBA model, using the *Interface Definition Language*. The CORBA model also incorporates a dynamic invocation interface (section 5.5.2), which allows clients to retrieve interface descriptions from an interface repository, and use these to build dynamic requests for remote objects. HIPPO+ implements a simple form of dynamic interfaces by requiring each service to support a `GET_PROTOCOL` operation (section 5.6.4). When this function is called, the service returns a textual string which tells the client the format needed to invoke the remote service.

This textual string can contain special tokens which are stored in the HIPPO+ *Registry* (see section 5.6.6). These tokens can then be expanded to particular client-specific values (eg. current node name, current selection etc). For example, if we consider a particular remote service which searches a linkbase for link definitions which match a given anchor selection. The user might select an interesting phrase from a node, then submit this to the link service to find out which nodes contain more information on this subject. The client begins by invoking the `GET_PROTOCOL` function, which returns a format for executing the service. In this case, the template is returned as:

```
-s <currentSelection> -l <linkbase>
```

This template describes the format the client must use to invoke the link service. The client must send the phrase that was selected by the user, and some identifier to describe the set of link definitions which will be matched against the selection. The client will automatically replace these tokens with the corresponding values, using the HIPPO+ *Registry* (see Section 5.6.6).

It is important to note that these interface descriptions for each HIPPO+ service are retrieved and evaluated dynamically on demand. This simple method of dynamic invocation allows the client to request remote services without having to know the syntactic requirements of each service beforehand. New services can be incorporated into the HIPPO+ model by simply providing a `GET_PROTOCOL` template string. Furthermore, the precise interface definition can change between requests so that the implementations can be updated dynamically without affecting other clients in the system. This provides a simple form of the *dynamic invocation interface* in the CORBA model.

5.6.6 HIPPO+ Registry

The previous section showed how the `GET_PROTOCOL` interface is used to inform clients about the syntactic requirements of each service invocation. The description tells the user the format of service requests, and which arguments must be passed to the remote object. However, it is often unreasonable to expect the user to replace

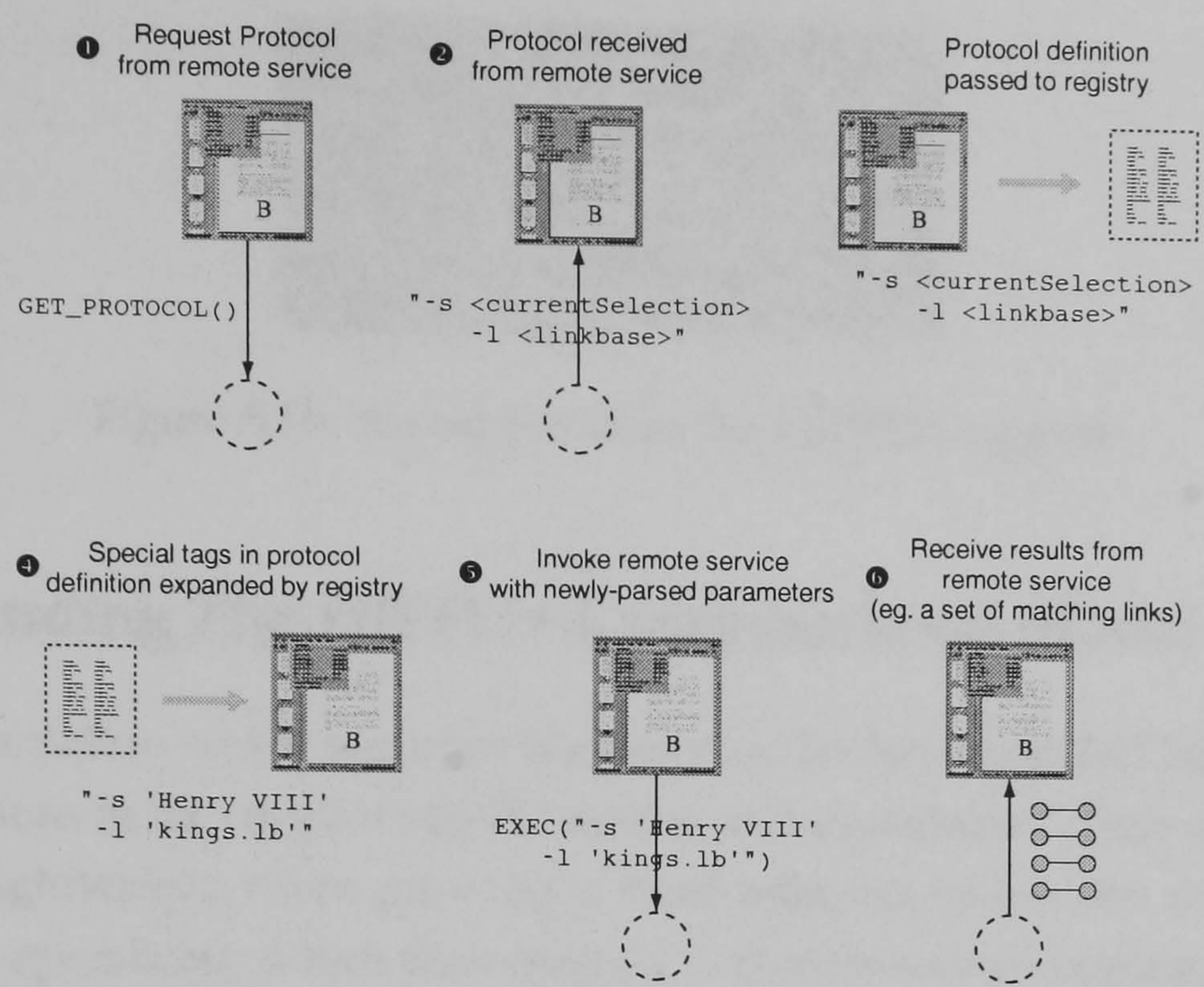


Figure 5.18: Some arguments automatically filled by HIPPO+ client

each of the tokens manually before calling the service. This would significantly add to the overhead of calling remote services, and would make the HIPPO+ system unwieldy to use.

However, many of these tokens which appear in the interface descriptions are known elsewhere in the system and can be automatically replaced by the HIPPO+ client. The previous example discussed a remote service which required the current anchor selection to be passed as a parameter. This anchor could then be used to match link definitions in a remote linkbase, and would return interesting links to the user. The HIPPO+ system uses the *Registry* to support special tokens, which can be automatically recognised by the client. The registry examines each of the tokens in the template string, and expands any that it recognises. In this example, the registry would replace the `<currentSelection>` token with the actual phrase that was selected by the user. Figure 5.18 shows the registry in use.

The *Execution Manager* (section 5.6.5) includes operations to expand the tokens in an interface, before the remote service is invoked (figure 5.17). The registry supports a diverse range of special tokens, which can be automatically expanded. Figure 5.19 includes an extract from the current registry which is supported in the HIPPO+ system. This registry reduces the overhead of invoking remote services by transparently expanding special purpose tokens. The syntax of most requests can be automatically constructed by the registry, which avoids the need for users to build service requests by hand. It also provides a simple means by which remote services can access client-specific information. Although this simple dynamic invocation support can be useful, Chapter 7 suggests ways in which it could be improved.

xdisplay	name of X display connection
appName	name of this application
aryBin	contents of argument bin
linkbase	current set of links
curSel	current selection
node	current node contents
curBuffer	current selected buffer contents

Figure 5.19: An extract from the HIPPO+ registry

5.7 Extending The HIPPO+ Computational Model

The implementation which has been discussed so far has identified key operations and abstractions in the original HIPPO model, and implements these as remote services. The lightweight client provides a fixed selection of buttons corresponding to hypertext operations, which then map on to remote service implementations. In this way, the functionality of the hypertext system is widely distributed throughout the network. The buttons which are presented to the user attempt to cover all of the commonly used hypertext operations (eg. *view node, follow link* etc). However, these *"commonly used operations"* are fixed, and decided in advance by the system developer. The mappings between buttons and remote services is also fixed and decided *a priori* by the developer.

While the HIPPO+ client may offer most hypertext operations that will be of use to the user, there are also many other operations and services that were not anticipated by the developer. Many remote services can provide useful functionality which may not necessarily be considered a typical *"hypertext"* operation, and so were not assigned a button in the client interface. For example, a particular remote service might extract the contents of a hypertext node, then email this to a colleague using some secure encryption. This is not the kind of service that would normally be considered a *"hypertext"* operation. Indeed, this kind of service might only be useful in a handful of cases, so the system developer quite rightly omitted this service from the HIPPO+ client. The developer did not provide a button in the interface, and the service cannot be invoked by the user. However, this is a useful service, and it seems unreasonable to limit users in this way.

There are many services and operations which could be useful in particular situations, which might not be considered hypertext operations. Nürnberg [NLS97] suggests that hypertext is just a specialisation of more general knowledge structuring, and so should incorporate a whole spectrum of knowledge management operations. One cannot expect the author to anticipate all of the services which will be useful to the user community. Indeed, these problems are not restricted to the kinds of unusual email services described previously. One could imagine a whole host of hypertext services and abstractions which might be developed in the future,

yet were not anticipated by the original HIPPO+ developer. Furthermore, the user may wish to use one of the author's predefined buttons, but may wish to map this on to some other implementation of the service. This alternative remote service may provide a more efficient implementation, or might support additional functionality etc. These services can be made available throughout the network, yet the current implementation of HIPPO+ does not provide any way for the user to invoke them. The user is limited to the fixed set of services that have been predetermined by the developer.

It is impractical to include a button or option for each possible operation which the user may wish to use. Hypertext models are widely applicable across *all* problem domains, and it is simply not possible to anticipate all of the computations and services which the user *may* find useful. Therefore, the HIPPO+ model of remote services has been extended to allow the user to execute other operations. As before, each of the buttons in the node browser (section 5.6.1) represents some key operation or abstraction in the HIPPO model, and these map on to remote services implementations. These are considered vital to the HIPPO hypertext model, and often have some implicit semantics (eg. the *Retrieve Node Contents* operation will place the results of the operation in the *Node Contents* buffer). In addition, HIPPO+ also allows the user to select *any* service in the network domain. The user can locate any desired service which they feel is useful to the hypertext, and invoke it when required. The user is no longer limited to the set of operations which were deemed suitable by the original developer of the system, but can select additional operations to augment the HIPPO+ system.

This extension to the conventional computational model offers a new level of extensibility to the HIPPO+ model. The functionality of the system is no longer fixed by the developer, but can be moulded and shaped to meet the precise demands of the user and the problem domain (figure 5.20). This encourages the HIPPO+ hypertext system to grow and expand – not only by providing new nodes and links – but also by providing new services which can be incorporated seamlessly into the hypertext system. Also, this approach helps to transfer control, away from the authors and system developers, to the users of the system. It is the users themselves who should decide which operations they require, and how they should be used. Therefore, it seems sensible to involve the user in deciding which operations and services can be used in the HIPPO+ system. Furthermore, this passes more control the user. The user selects the components which he finds appropriate, and is not forced to use simply those operations which the developer deems suitable.

This section explores some of the ideas which have been added to the HIPPO+ system to support a flexible approach to computation. In particular, the system provides additional services to support the management of remote components to help the user locate useful services. The *query interface* provides a simple method of

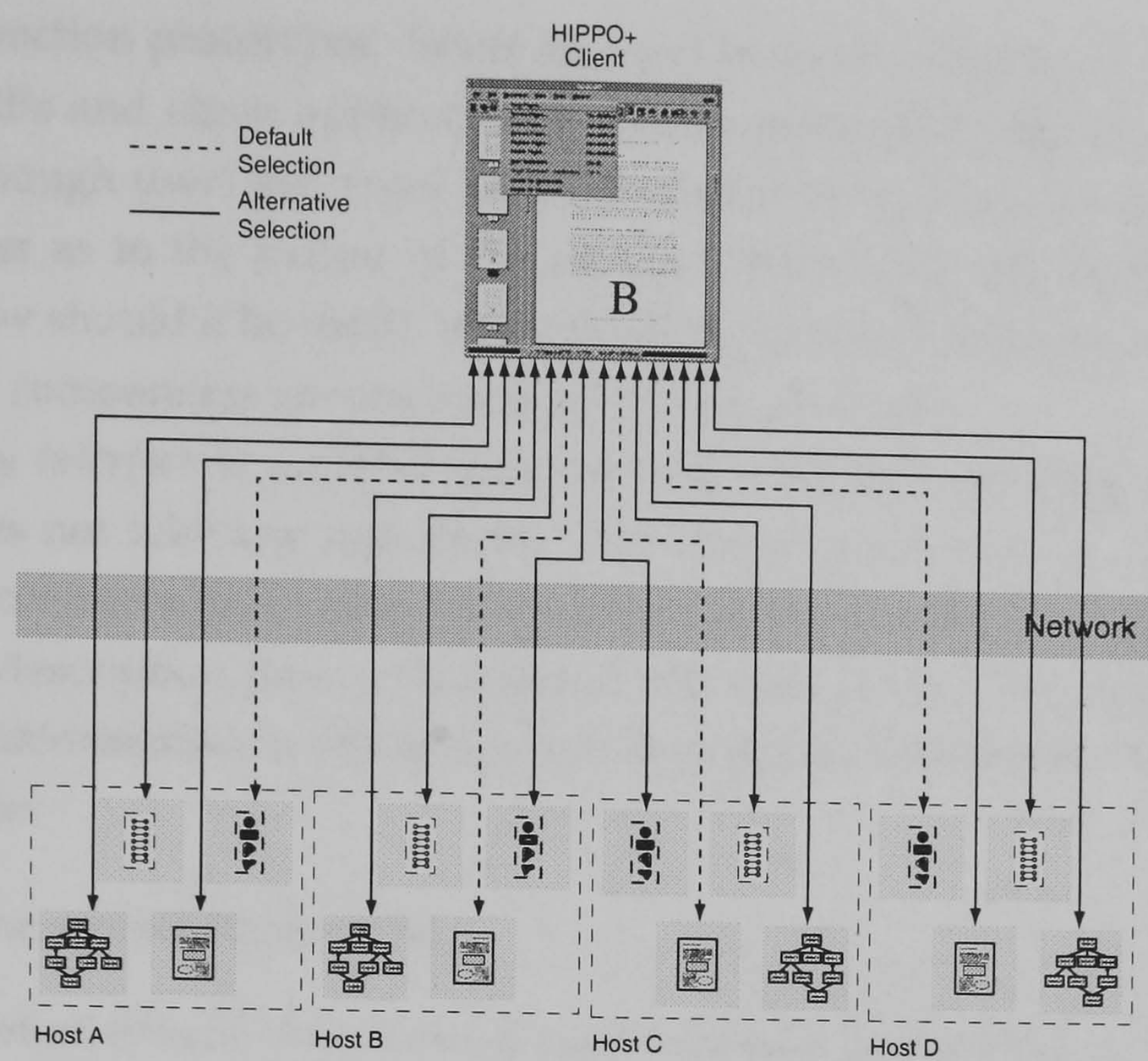


Figure 5.20: Extending the computational model to incorporate additional services

querying remote services before they are invoked. This provides users with information about each service, so they can decide whether the operation will be of use. The *trading service* supports a simple directory service for locating remote services using aliases. Finally, the *Hypertext Component Hierarchy* attempts to provide some form of classification system for organising collections of components into more manageable directed graphs.

5.7.1 Query Interface

One of the main difficulties of maintaining a distributed environment is providing an effective means of managing the remote services and components. A distributed system can contain many hundreds, perhaps thousands, of components – each offering different services and supporting different hypertext abstractions. The user needs some way of ascertaining the semantics of each service, and of finding out the precise details of each service. The CORBA model uses the IDL language to provide a portable way of defining object interfaces. This can also be used to support a *self-describing* system – every object and operation can be expressed unambiguously and precisely in a platform-independent way. The HIPPO+ registry described in section 5.6.6 offers a simple of defining the format of service invocations, and supporting dynamic invocation.

However, the IDL and HIPPO+ registry approaches provide a very low-level way of defining services. They express each operation in terms of standard data

types and function prototypes. While this can be useful (indeed, it is essential for building ORBs and client applications), it does not explain the *true* semantics of service. Although users are aware of the format that each request must take, they are no clearer as to the nature of the service. What does the service attempt to achieve? How should it be used? When should it be used? What are its limitations? Which other components should complement the service?

The *query interface* is an RPC interface which *all* services must support. The function does not take any arguments, and returns a text stream. This stream is intended to contain a description of the service and provides the user with a more meaningful description than conventional IDL definitions. The author is free to include any information in this text which they deem to be useful. A typical entry might include:

- what the service implements
- explanation of how the service is used (optional parameters etc)
- other services to be used in conjunction with this service
- relevant information (eg. implementation details)

This is a very simple tool for managing HIPPO+ services, but it can provide a useful method for HIPPO+ users to find out more information about a service. The query interface is a simple way for authors to include additional information for the user, and to suggest other services which might of interest. The query interface *must* be supported by all services in the HIPPO+ model, so every client can rely on this minimal functionality. The HIPPO+ application includes a browser for querying services and viewing the object descriptions (figure 5.21).

5.7.2 HIPPO+ Trading Service

Section 5.5.3 discussed the *Object Services* layer in the CORBA model, which provides additional services for a distributed environment. Of particular interest are the *Naming* and *Trading* services which allow clients to locate remote objects. The CORBA standard describes a powerful trading model which allows objects to advertise the services that they offer. Clients can then request an object by describing the type of service they require, and the trader will attempt to locate a suitable object. The trader incorporates directory services, query services and allows multiple trading spaces to be combined together into federated traders. CORBA traders can also support multiple implementations, and use different policies for matching services to client requests.

The extended HIPPO+ system allows users to invoke arbitrary operations to augment their hypertext environment. The author envisages a large-scale HIPPO+

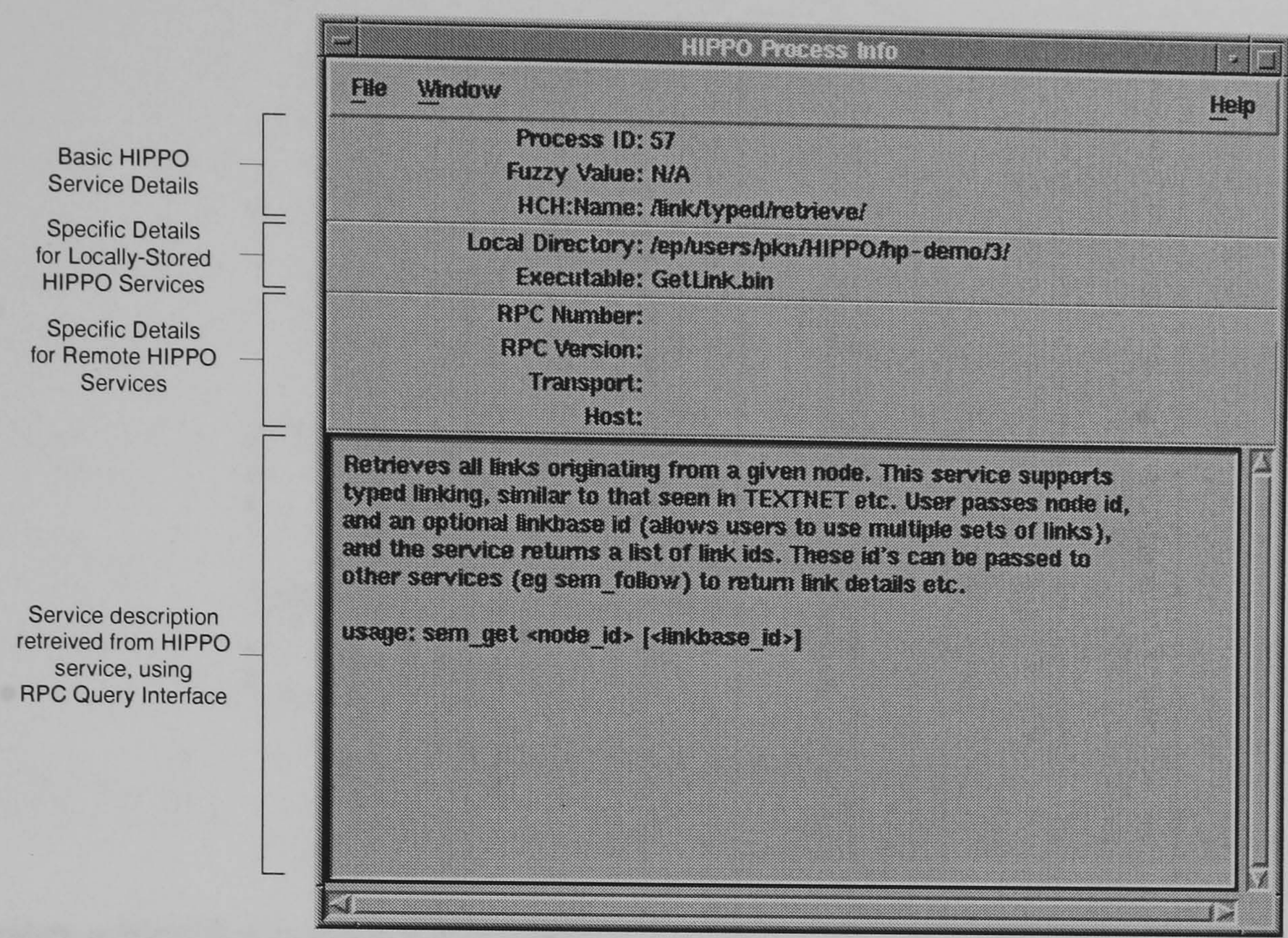


Figure 5.21: The *Query Interface* browser

domain, containing many hundreds of components – each offering different services and operations. The user can select objects based on their specific needs – different implementations; different hardware platforms; diverse hypertext tools which support different linking semantics etc. The user needs some means of managing this complexity, so that they can locate the desired services quickly and easily.

The current prototype implements a limited trading service which provides a name resolution mechanism for locating objects. Remote services can be assigned alias names, which can be used to reference objects in the network domain. This provides a more natural model which hides the complexity of manipulating RPC program numbers. This trading model based on alias references does not offer the flexibility of the CORBA model which allows more complex queries to be resolved. The CORBA trader service allows objects to advertise the type of services they implement and for multiple trading spaces to be combined together. The simple HIPPO+ alias-service model has more in common with the use of *monikers* in the DCOM model [Act]. Monikers provide a level of indirection for accessing DCOM objects, by providing an alias name for DCOM objects. They are usually maintained by the system registry, and resolve to a specific DCOM object. Chapter 7 discusses this idea of directory services in HIPPO+ in more detail, and suggests how this could be used together with existing standards such as X.500 [Uni93] etc.

The HIPPO+ trader uses a very simple implementation using the *rpcbind* services discussed in section 5.6.3. The *rpcbind* daemon maintains a list of all RPC

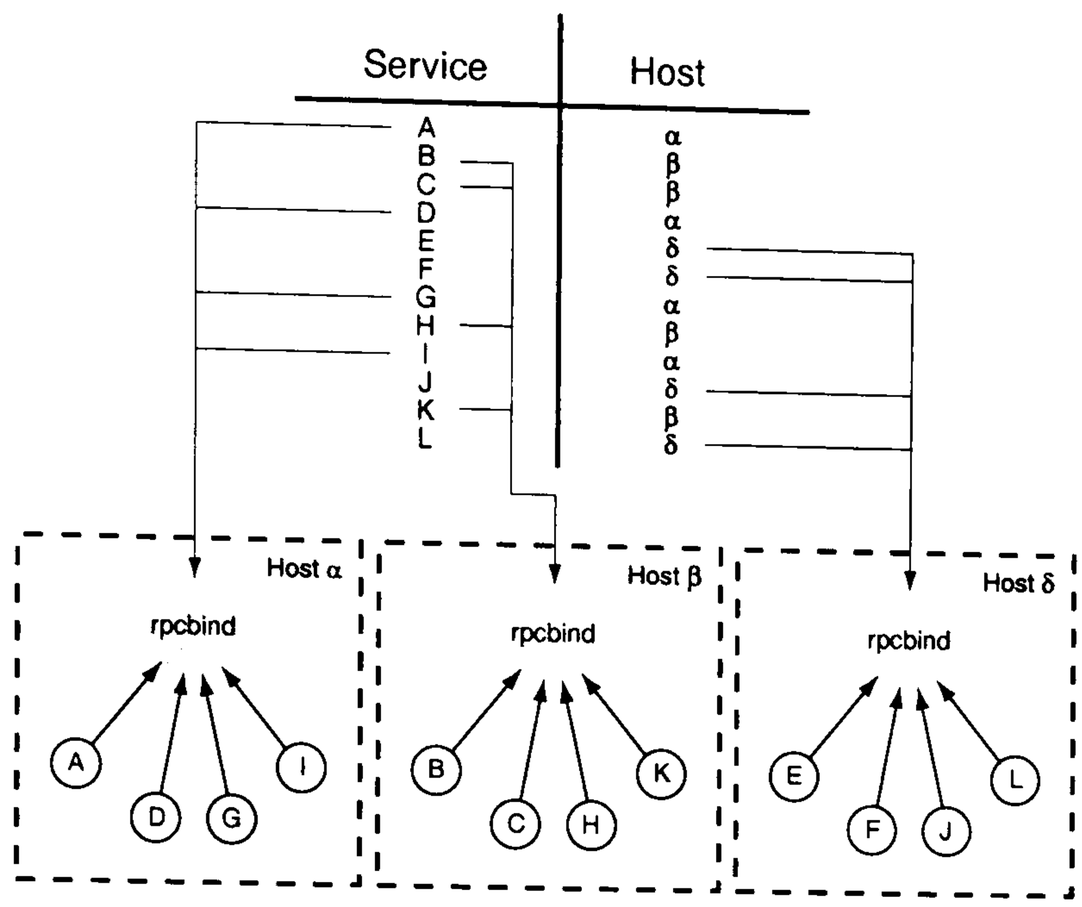


Figure 5.22: The HIPPO trader model

services which are currently running on the particular machine. The HIPPO+ client requests these details from the daemon, then asks each remote service to supply an alias. As in section 5.7.1, each service *must* support an operation to return an alias name. The client can then query each *rpcbind* daemon on each machine in the domain, to maintain a list of alias-object mapping (figure 5.22). However, this is not a complete trading service, and does not allow the flexibility of the CORBA trader service. For example, the user cannot ask the trader *"where is an object called <x>?"* or *"give me an object which does this task"* etc. Chapter 7 identifies some of limitations of the current trader implementation and suggests some future directions for development.

The HIPPO+ client maintains a list of every machine in the HIPPO domain, and allows the user to decide which nodes are included in the trader space. Users can add new machines to be polled, or remove unwanted machines from the trader space. This implements a simple form of federation for combining multiple trader domains together. The application includes a tool for manipulating these trader spaces and for requesting alias names from remote machines. An example of the trader tool is included in figure 5.23.

5.7.3 Hypertext Component Hierarchy (HCH)

The previous sections have shown some of the approaches used in HIPPO+ to help the user manage collections of remote services. The query interface allows the user to understand more accurately the true nature of each service, before deciding whether to use it. The trader model provides an *alias-object* mapping to access remote objects using textual names. It is hoped that these tools will help the user maintain a distributed domain. However the object space remains an essentially flat

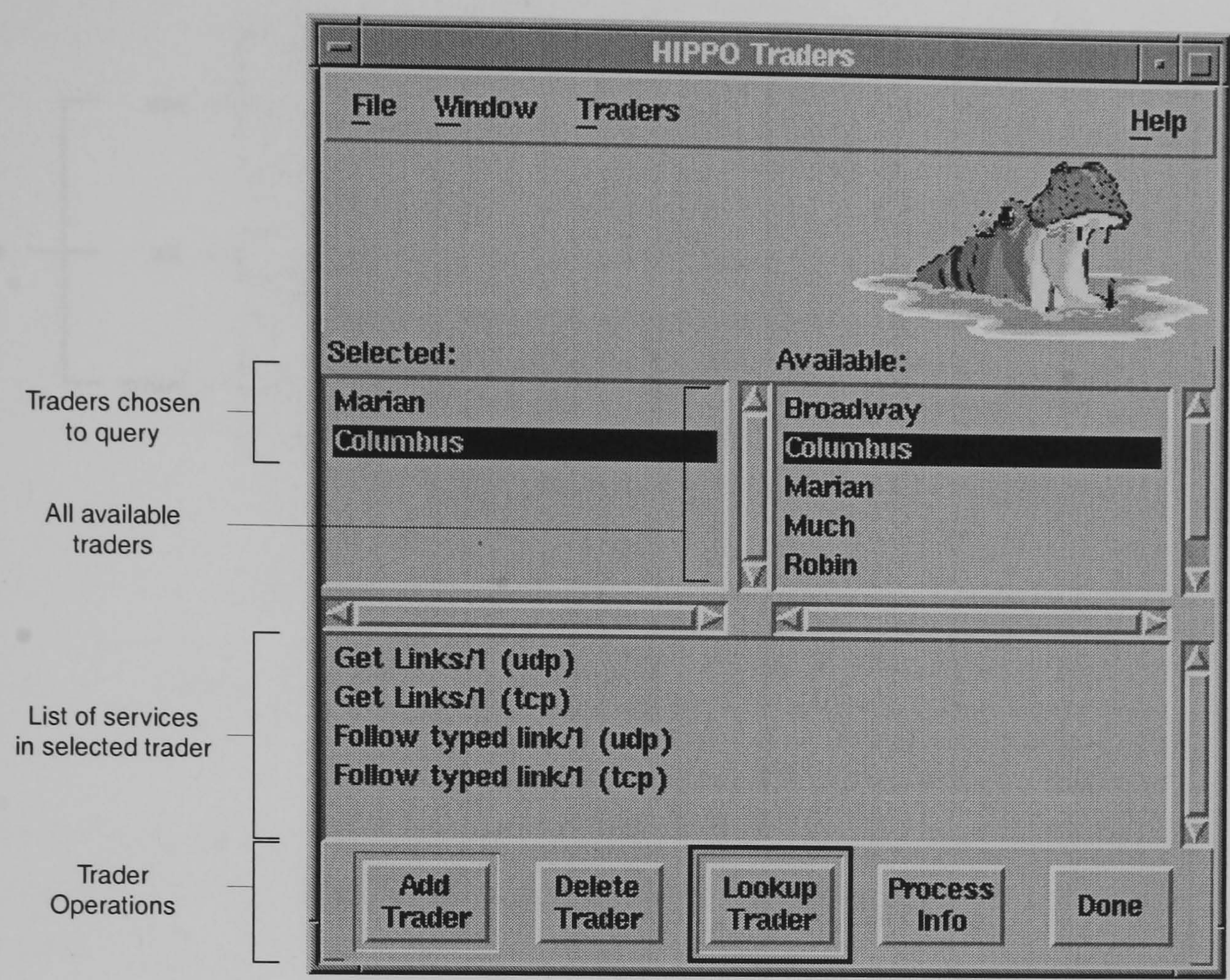


Figure 5.23: The HIPPO+ trader tool

structure, and the *alias-object* mappings only provide a limited level of abstraction. While users can query the services at a particular machine, they have no way of organising these components into meaningful structures.

The *Hypertext Component Hierarchy* (HCH) provides a hierarchical classification system for organising remote services. Services can be arranged into a hierarchy, and classified according to their semantics etc. This resembles some of the approaches used in the CORBA *Naming* and *Trading* services (section 5.5.3), and directory services such as X.500 [Uni93], DNS [SP82] etc. The HCH taxonomy allows the hierarchy to be constructed using any criteria for the graph – semantics, location, security etc. However, early experiences have suggested that the most useful HCH hierarchy arrangement can be achieved by using a layout which is tailored towards hypertext systems in particular. We suggest a HCH hierarchy organisation based around the three fundamental abstractions in a hypertext system – the *node*, *link* and *anchor*. Services can then be arranged according to the operations that they perform on each of these abstractions. Figure 5.24 suggests a typical example, showing a selection of arbitrary hypertext operations classified into a hierarchy.

The current implementation requires the original author of each remote service to decide on an appropriate HCH entry, which dictates where the service will appear in the hierarchy. This assumes that the author will understand the true se-

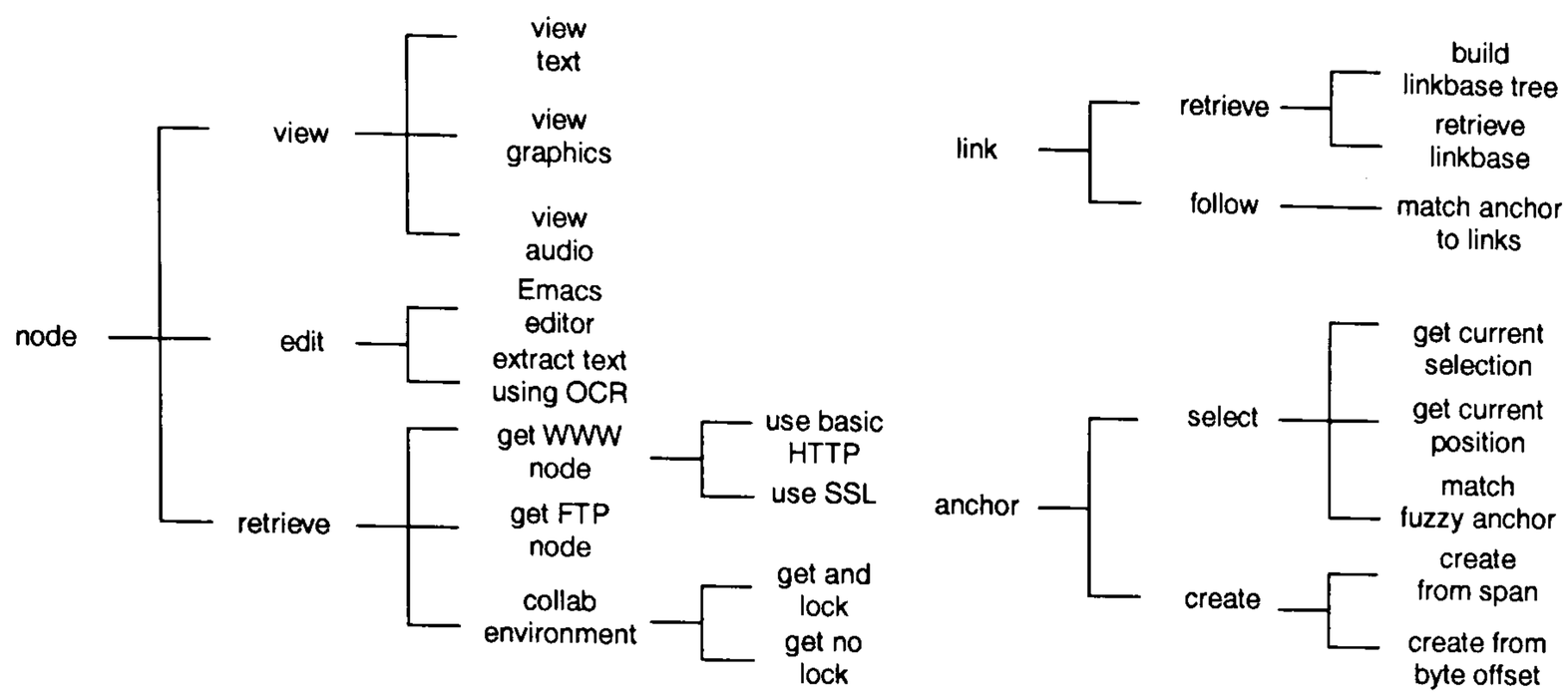


Figure 5.24: An example HCH hierarchy

mantics of the service, so will be better placed to decide on an accurate HCH entry. Each service must support a GET_HCH RPC operation which returns the hierarchical HCH path, which is used to position the service in the client HCH hierarchy. The HIPPO+ client also provides a simple browser for exploring a HCH hierarchy (see figure 5.25). The hierarchy is displayed as a simple tree widget, and users can open/close branches as they traverse the hierarchy. When the the user locates a service which is of interest, he can invoke this in the usual way (section 5.6.5), or query the service for more information (section 5.7.1).

This attempt to classify remote services is perhaps too simple for large scale systems, but suggests a useful route for future development. Chapter 7 explores some of these directions which could incorporate other directory services and classification systems.

5.8 Summary

This chapter has introduced the HIPPO+ system which attempts to re-implement the existing HIPPO prototype, using a distributed architecture. Section 5.1 summarised the HIPPO application which has been used to evaluate many of the ideas presented in this thesis. This develops many of the key abstractions in open hypertext systems such as fuzzy anchoring models, linkbase inheritance trees, adaptive models etc. However, the discussion also identifies some of the problems with the current implementation, which prevent it from being used as a true open hypertext

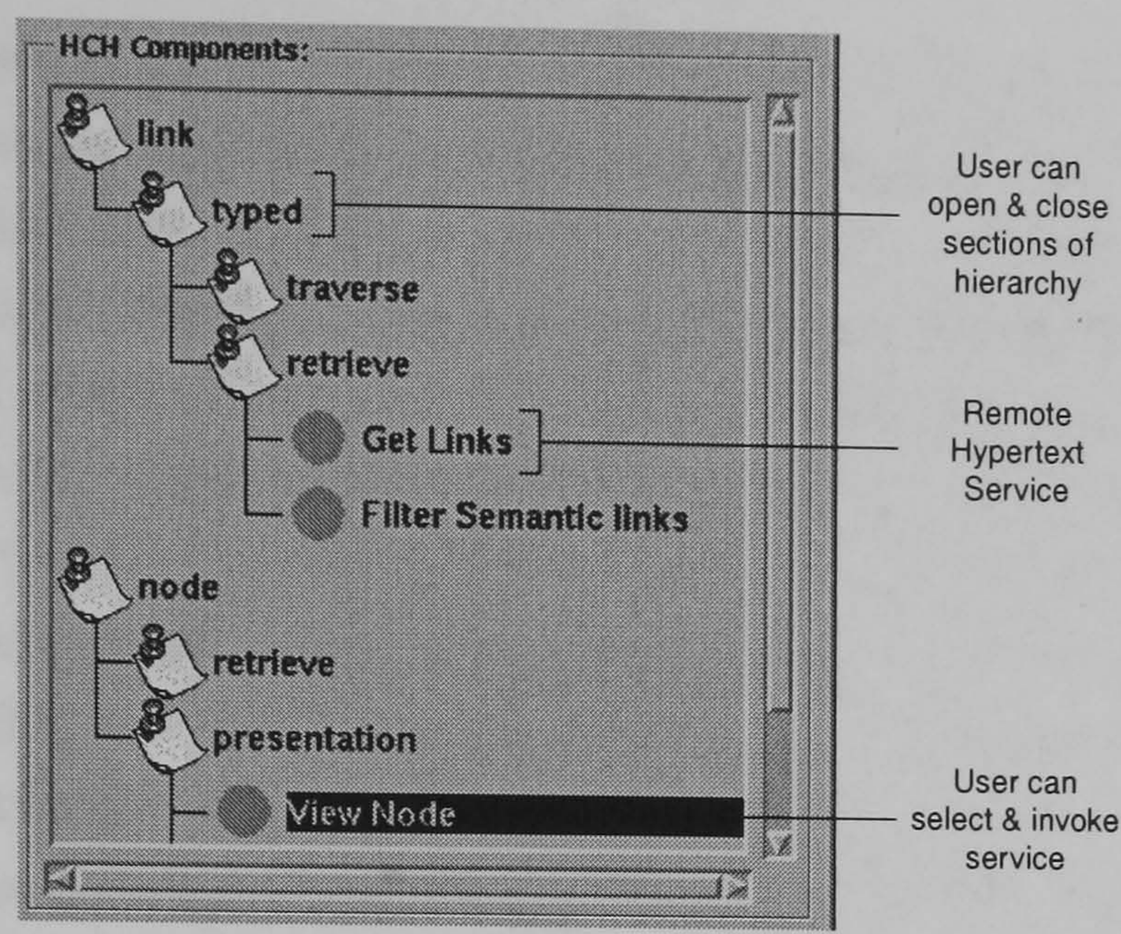


Figure 5.25: The Hypertext Component Hierarchy browser

system. In particular, HIPPO has been largely implemented as a single, monolithic application. The details of the implementation and the semantics of each hypertext abstraction are embedded deep inside the application, and are not accessible to external applications. This prevents the existing tools from incorporating the ideas presented in this thesis, and suffers from many of the same problems as early hypertext implementations.

Section 5.2 shows how the HIPPO implementation has been revised to include a loosely-coupled view of a hypertext system. HIPPO+ identifies each of the key operations and functions in the HIPPO system, and implements these as a set of widely distributed services. The main HIPPO+ application is simply a lightweight client which can be used to invoke remote services, and collect the results. In this way, the HIPPO+ client no longer implements any specific hypertext functionality, but distributes the functionality throughout the network. Section 5.3 goes on to discuss the advantages that this new model has over the previous HIPPO model. HIPPO+ develops a strong notion of computation and encourages a more optimal division of operations.

Distributed systems offer a number of benefits to the user, compared with local, centralised applications. As the interest in distributed architectures has grown, a number of mechanisms for supporting distributed communications have been developed. Section 5.4 describes some of the common inter-process communications models such as remote procedure calls which are used in the HIPPO+ system. A number of distributed frameworks such as DCE, DCOM, CORBA are introduced which provide environments for developing large-scale distributed applications. The section closes with a discussion of compound documents which provide a richer storage model for developing component-based systems. Chapter 7 also

discusses the choice of an RPC model, and suggests ways that HIPPO+ could be better implemented using a model based on compound documents.

The CORBA standard provides an open reference model for developing distributed applications. This provides a communications infrastructure and a collection of additional object services for managing large numbers of distributed components. CORBA has been influential in the design of HIPPO+ and section 5.5 explores the key features of the CORBA model in more detail. The naming and trading services have been particularly important, and some aspects of these have been implemented in the HIPPO+ system (section 5.7). Chapter 7 also suggests that a better implementation of the HIPPO+ system could be achieved using CORBA, and discusses reasons why an implementation based on RPC was used.

The current version of HIPPO+ has been implemented using C++/X11/Motif, and ONC-RPC libraries for invoking remote services. Section 5.6 describes the implementation of this prototype, and the tools which have been made available to the user. The RPC interfaces which are supported by remote HIPPO+ services are described in sections 5.6.3 and 5.6.4. The HIPPO+ prototype also supports a form of dynamic invocation which retrieves interface definitions on demand, and matches these to a system registry (section 5.6.6).

The revised HIPPO+ model introduces a very strong computational element into the hypertext environment, and views a hypertext system as a collection of services. HIPPO+ identifies each key operation in a hypertext system, and implements these as remote object services. This endorses the view of a number of hypertext researchers [Hal87, Kac90, HGC94, TBR93] which encourage a computational view of hypertext applications. Hypertext abstractions are no longer viewed as static, passive objects, but can include a stronger behavioural element. Section 5.7 explains how the HIPPO+ computational model has been extended to allow users to locate and execute *arbitrary* processes and hypertext services. This is an important development which allows users to move beyond the limited set of predefined hypertext operations. The user can incorporate services which were not even considered by the original system developer, and can choose alternative implementations of common operations.

This view of a hypertext system as a series of remote services is central to the HIPPO+ implementation, and provides a useful platform for future research. The intelligence and functionality of the system is widely distributed throughout the network domain, to provide a more scalable and robust hypertext environment. HIPPO+ allows the user to play an increased role in the development of the hypertext system, by allowing them to select appropriate services to use in the environment. This extends the approach taken by systems such as Intermedia [YHMD88] and Microcosm [FHHD90] etc which do not make any artificial distinction between the author and user. Fountain suggests that knowledge structuring tasks and link

creation are important learning tasks which do not belong solely to the domain of the author. Similarly, HIPPO+ suggests that the user should also provide input into the design and development of the system. Chapter 6 develops this idea further and suggests an adaptive model which could be incorporated into the HIPPO+ system. This uses feedback from the users to select appropriate services and implementations of remote operations, based on the current node being visited. Chapter 7 also suggests ways in which the HIPPO+ model could be developed further in future research.

Chapter 6

A Proposed Adaptive Model For HIPPO+

The previous chapter identified some of the limitations of the HIPPO prototype, which was implemented almost entirely as a single, monolithic application. In that chapter we proposed a new model based on widely distributed services. These remote objects are used to implement the functionality and operations which would previously have been embedded deep in the HIPPO application. By opening up the workings of the application, it was possible to provide a more open environment in which services can be shared between users. The new HIPPO+ model also inherits many of the advantages that distributed systems have over localised applications (avoid amount of redundancy, robustness, shared resources etc).

The HIPPO+ system identifies a collection of operations which are considered essential to a hypertext system, and binds these to buttons in the user-interface. Each of these buttons corresponds to a remote service which is implemented at some other location in the network domain. This distributed model attempts to re-implement the same functionality as the original HIPPO client, using a widely-distributed topology. However, it also can be useful to allow the hypertext system to extend its functionality and incorporate new services. Section 5.7 developed the computational model by allowing users to select and invoke *arbitrary* services which can be used to replace or augment the existing services. Existing services can be replaced by other components, providing alternative implementations or even slightly different semantics.

The user is no longer restricted to using the small set of designated services which were considered useful by the system developer. The user is now free to choose from hundreds, perhaps thousands of services which are available throughout the network. This new approach opens up limitless opportunities – users can choose their own linking tools; users can invoke arbitrary computations when traversing links; users can view nodes using different viewers etc. It is the *user* that

controls the functionality of the environment, not some software developer or hypertext author. The user can decide what constitutes a link traversal, what it means to view a node or which storage operation to use to store hypertext objects.

However, this new freedom to select arbitrary services requires a framework for managing this new complexity. The HIPPO+ application must provide some means of managing these large collections of processes and services. Section 5.7 described some initial steps which have been made in this direction – the *Query Interface, Trading Service* and *Hypertext Component Hierarchy* – which help the user manage remote objects. These have been compared to the object services available in the CORBA model, although these are only very early attempts at managing objects, and chapter 7 suggests ways in which the HIPPO+ tools could be developed further. This chapter proposes an adaptive model which could be used in the HIPPO+ system. This has not been implemented in the initial version of the current prototype, and is presented as a theoretical model for future work.

6.1 Advantages Of An Adaptive HIPPO+ Model

The current HIPPO+ model offers many of the advantages of distributed software systems, and allows the user decide on the operations and functionality in the hypertext environment. However, in some ways, the HIPPO+ system provides the user with *too* much flexibility. Users must constantly make decisions as to which services they should use to perform common hypertext operations. Are the default services suitable for this node? Could the hypertext operation be implemented better using another service? Should the user search for another service which may be able to offer additional facilities? Perhaps a service in the HIPPO+ domain could offer some operations which would be useful in this situation – operations which the user would not normally even consider? These questions all add to the overhead of using the HIPPO+ model. The role of the user is no longer to "simply" browse the hypertext; the user must now also decide which services they should use to perform these tasks.

An adaptive model can help to address many of these problems, by suggesting which services and operations to use, based on feedback from previous users. Many hypertext systems allow the environment to be extended by incorporating additional functionality into the hypertext model; indeed, many systems offer better ways of supporting this integration than HIPPO+. However, these systems do not help the user decide *which* services should be incorporated, or when additional services should be used. It seems natural to take the experiences of each user, and use these to guide other users that follow them. Novice users can share the expertise of more experienced users, and do not need to repeat the same mistakes. Services that particular users found useful in some circumstances will often be use-

ful to other users in the future, and modern hypertext systems should help capture this knowledge. This chapter proposes an adaptive model which has not initially been implemented, but could be incorporated into the current HIPPO+ prototype in any future work. This model uses feedback from the users to suggest services that should be used with particular nodes, or may be particularly useful in certain situations.

6.2 Example Adaptive Services

This section begins with some example scenarios which show how an adaptive model could be used in the HIPPO+ hypertext environment. Each example identifies some common situations which would benefit from adaptive modelling, by allowing the choice of operations and remote services to change in response to user feedback.

6.2.1 Example 1

Consider a user reading a node in a medical hypertext, which discusses some medical subject – perhaps the human nervous system – and includes many medical terms and concepts. The user is unfamiliar with many of the terms, and finds the document content difficult to comprehend. The user uses some default linking service to support basic hypertext links. The service is passed a single anchor word and matches this against a specified linkbase; in this case, the user might choose a general purpose medical linkbase. The link service then returns a list of matching links, which users can select if they feel they are appropriate (figure 6.1). This simple link matching service provides a basic linking model based on the ideas of linkbases, and would probably prove useful to many users of the HIPPO+ system.

While this linking service satisfies most of the users needs, the user still finds reading difficult. Finally, the user decides to try to find another implementation of a link service, which may offer additional facilities. After much searching, the user chances upon another remote service which offers a complex link matching service. With this service, the user supplies, not simply a single anchor word, but can include a larger selection of text – perhaps an entire paragraph. The service then extracts any terms, ideas, phrases etc from this text which could be problematic for a user. These terms are used to cross-reference with an exhaustive collection of medical resources which are stored remotely. The service then returns a set of links to appropriate definitions for each of these terms. However, the user can also provide some measure of their expertise (*novice, student doctor, consultant* etc), which represents the knowledge level of the user. When the service parses the anchor text and searches the linkbase, it can then use this expertise indicator to modify its

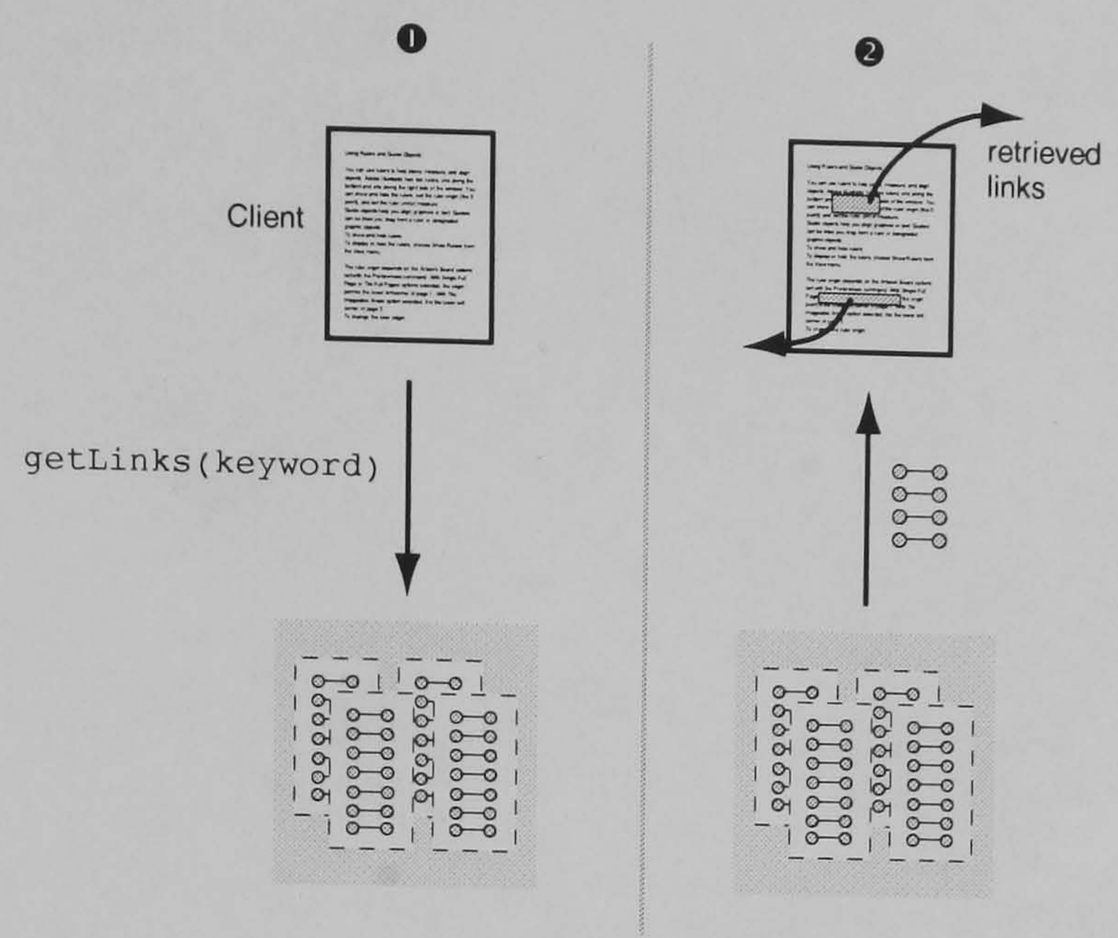


Figure 6.1: A simple default link service

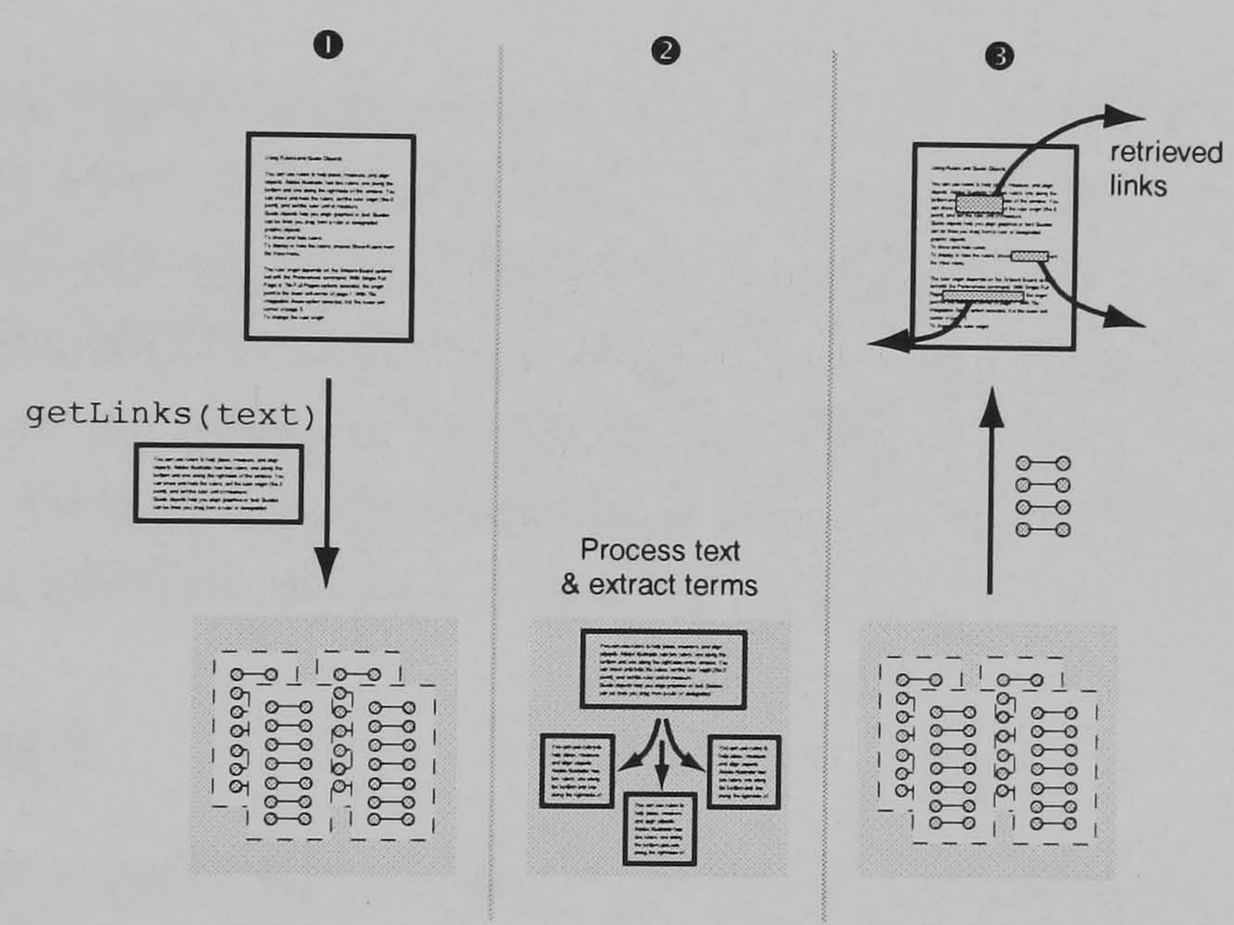


Figure 6.2: A more complex medical link service

behaviour (figure 6.2). In this case, this new link service provides a vastly superior method of retrieving links. The user decides that this link service will be more helpful for exploring this difficult medical hypertext, and decides to use this service for all future link requests, instead of the original default service.

The previous example could be typical of many users – they encounter problems in a hypertext, then locate services which are better suited to their tasks. Section 5.7 showed how HIPPO+ has been extended using query interfaces, a Hypertext Component Hierarchy and a simple trading model, to allow users to execute *arbitrary* services, in addition to the default services. However, the current HIPPO+ model forces each user to locate new services independently from all the other users. Each user must work in isolation, making the same mistakes – would another imple-

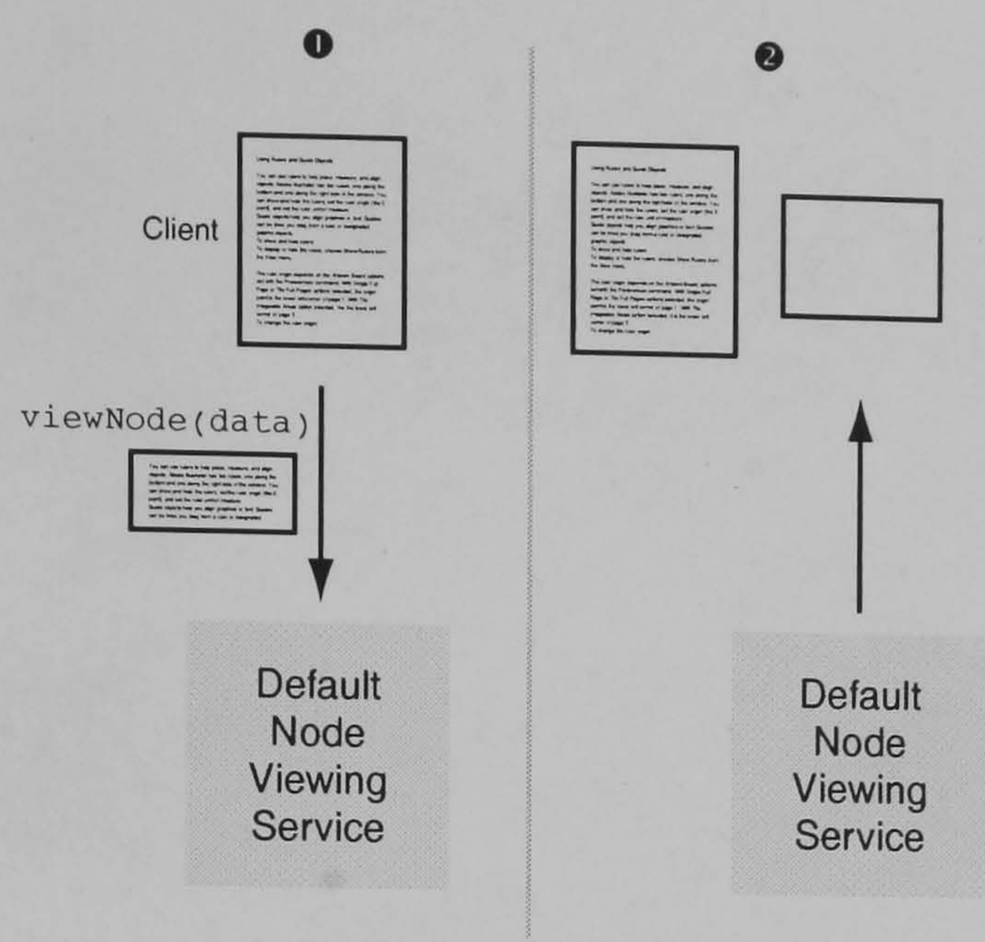


Figure 6.3: Default viewing service

mentation of this hypertext operation be more appropriate? What should this new implementation offer? Where can I find it? Can I guarantee that a better implementation exists? An adaptive model builds on the experiences of users and helps to share expertise between users. An adaptive HIPPO+ could recognise the value of a new medical link service, and suggest this the next time other users encounter the same node. Future groups of users can benefit from the experiences of previous readers, and the HIPPO+ system can be adapted to include this new service.

6.2.2 Example 2

One could imagine many other situations where the default implementations of hypertext operations may not be suitable for all tasks. The remote objects which are used to implement the hypertext functionality should change and adapt to match the needs of the precise situation. For example, consider the simple act of viewing the contents of a node. The current HIPPO+ system uses a pre-defined service to implement this operation (eg. *display the node contents using the GNU Emacs editor*). This service can be implemented remotely, and does not require the user to have the viewing editor installed on their local machine (figure 6.3). The viewing service can be administered remotely, and can be updated centrally. This default viewing service has its advantages, and users may find it to be satisfactory for most situations.

However, the Emacs editor only supports textual content, so what happens when the user encounters a node containing graphics? A short term solution might replace the default Emacs editor with a more functional browser which supports multiple media types. Perhaps the Emacs application itself could be extended to display images and support a variety of graphics formats? This might solve the problem for most nodes and documents, but what happens when new graphics for-

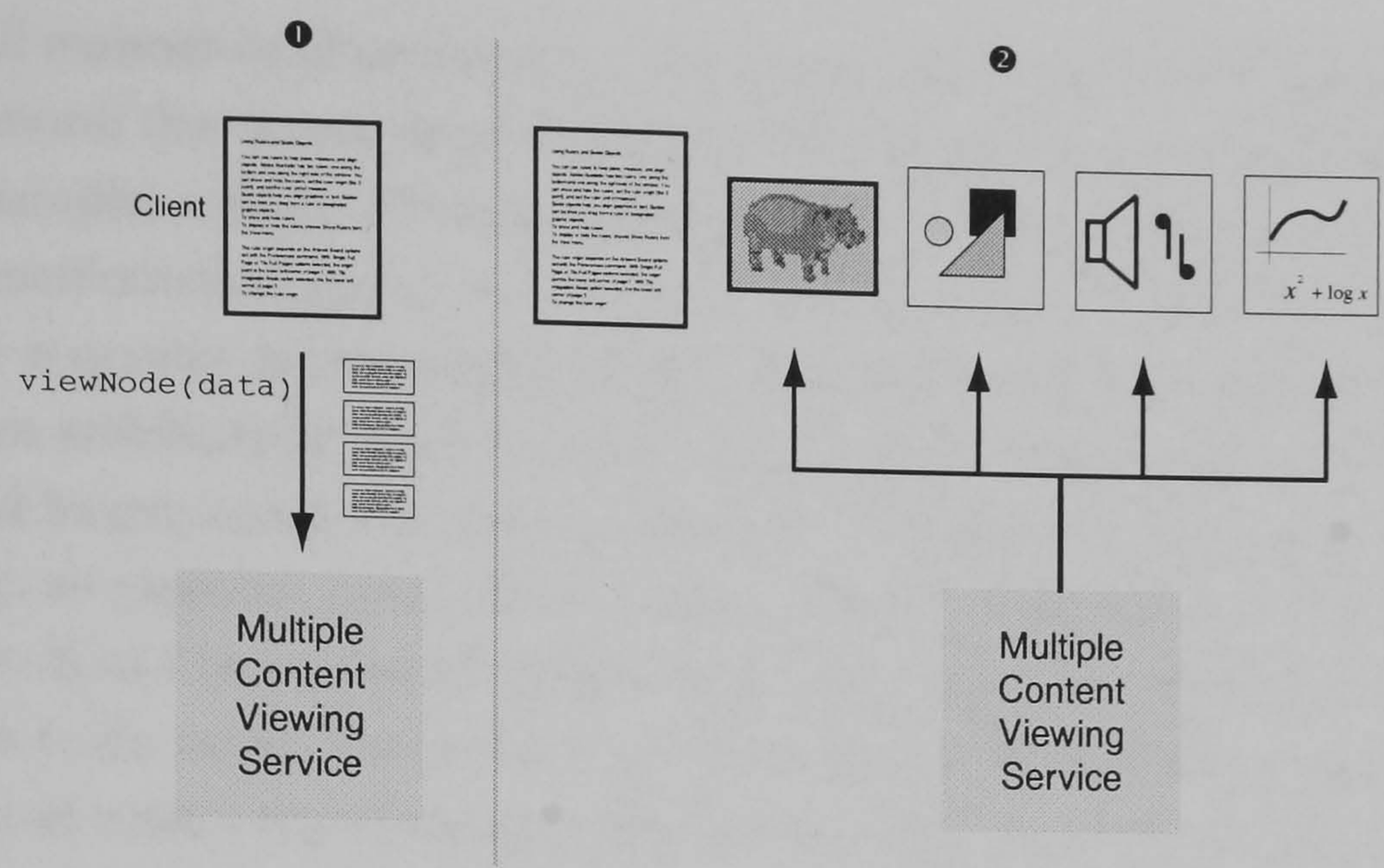


Figure 6.4: Multiple viewing services tailored to content formats

mats are introduced? What happens if the node contains more complex image data – perhaps a PostScript program – does the viewing application need to be extended to provide a native PostScript interpreter?

It is simply not sustainable to provide viewing support in this fashion. The default viewing service cannot be extended each time the user wishes to view a node which contains new content formats. The viewing service cannot be aware of *every* format of *every* node which *may* exist in the future. A more scalable approach should select viewing services based on the underlying node contents themselves. For example, if an author wishes to incorporate a new content format – perhaps the PDF document format described in previous chapters – then the user should use a *PDF Viewing* service. Similarly, complex graphical images should be viewed using native applications which provide dedicated support for graphical presentations (figure 6.4). This idea is often used in open hypertext systems which use native applications which are tailored to the particular formats of the node contents (although these are rarely implemented as remote services). This is usually achieved by examining the file type or suffix of node files to determine the appropriate viewer application (eg. files ending in `.pdf` use a PDF viewer, `.gif` suffices use graphics packages etc).

However, even this flexibility for viewing nodes is not really sufficient for expressing more complex viewing semantics. While it is true that the choice of viewer should be based on the node, it seems too simplistic to base the decision solely on the format used to store the node contents. Of course, the particular data format which is used inside the node must influence this decision – for example, it seems natural to view image data using some graphics package. However, the node format is by no means the only consideration, and the viewing service must take into

account all manner of other factors.. Some node data may benefit from particular interpretations; the viewer often depends on the type of user and the task at hand.

For example, a node which contains a complex typographic layout – perhaps a highly mathematical paper with many equations – may benefit from being processed by a quality typesetting package, then presented as a typeset document. Perhaps an architectural model should not simply be viewed as a plan drawing, but should be processed and used to generate a 3-dimensional model? The model could then be explored interactively using a virtual-reality engine. Perhaps a simple image should be processed using some image analysis tools before presenting the results to the user? Similarly, a technical computing document may include a discussion of some programming language etc. The conventional approach would view this using a simple text editor, yet a more useful viewing model may extract the relevant code sections, and compile these into executable code. The programs could then be executed and evaluated while simultaneously viewing the explanatory text.

Many open hypertext systems provide methods for adding new viewer applications, and may allow the user to select how each node is presented. However, the fundamental advantage of an adaptive model, is that the system itself actually *learns* which viewers to use. If a set of users find different viewing models to be useful with particular nodes, then the system can suggest these viewing services to other users. The process of *viewing a node* is a complex operation which aims to convey the semantics of the node to the user. Many situations may demand a much richer definition of *viewing* which does more than simply display the node on the screen. As such, the user should not be limited to using the viewing service which has been deemed suitable by the system developer. The user should explore other services which may bring new benefits to a particular node, so that the precise implementations and semantics of hypertext operations are tailored to the particular nodes and tasks. An adaptive model should observe which services are most useful with particular nodes, and suggest these to future users.

6.2.3 Example 3

The two previous examples have shown how an adaptive model can be used to complement many typical hypertext operations such as viewing nodes and traversing links. Many OHSs already provide some level of extensibility for tailoring these operations (eg. adding script executions to link traversals or specifying the viewer for a node). An adaptive model brings a new intelligence to this extensibility, so that users can benefit from the experiences of other users. Services which are found to be useful can be reused and incorporated in subsequent sessions. However, it is important to note that adaptation in HIPPO+ should not be limited solely to viewers or the choice of link services. These viewing and linking hypertext operations

are indeed central to any hypertext environment, but the HIPPO+ model views the hypertext system as a far richer collection of arbitrary services. An adaptive model should apply adaptive modelling to *any* hypertext operation, not just the viewing of nodes or link traversal.

For example, a common operation in an open hypertext system is to provide some means of making selections in an application. A user may wish to select a piece of text in an editor, then submit this to some linkbase. The link server then returns a list of matching link definitions, which can then be traversed by the user. The default *get selection* operation in HIPPO+ uses the clipboard of the windowing system for communicating between the client and other applications. This *cut-paste* model is widely used in many open systems [Net, FHHD90, ACDC96], and provides a simple method of making selections from external applications. However, example 2 showed that the user should be able to use *arbitrary* services for viewing nodes, and that the system will adapt to suggest those that are considered most useful. As such, the default method of making selections may no longer be so useful. Perhaps the viewing application doesn't support the system clipboard? The clipboard is widely used to communicate between programs in open hypertext systems, but it does not scale well in large-scale, distributed programs. The clipboard mechanisms rely on each application operating within the same domain – either the same operating system or sharing the same windowing environment (eg. X11). This becomes increasingly problematic as services are distributed throughout different domains and running on different platforms.

More importantly, the clipboard paradigm may simply be unsuitable in some situations, and an alternative implementation may be more appropriate. For some applications, a more efficient means of receiving the current selection would be to open some network connection, and pass the data across the network. A different *get selection* service may copy the selection to some shared storage engine which can be accessed at a later date by all users. Some users may require a faster implementation to retrieve the selection, or perhaps a service which transfers the selection using a secure transfer method. An adaptive model will identify those selection services which are found to be useful in particular situations, and associate these more closely so that they are reused by future users.

However, the most interesting opportunities arise when we consider the actual *semantics* of the *selection* operation. This is an important operation which is vital in any hypertext environment, and HIPPO+ identifies a default implementation of this service. The semantics of this operation seem clear – to return the contents of the current selection so that it can be used for matching links and submitting queries etc. A simple example might involve a user browsing a text document – users could select a phrase which they find interesting, and then submit this phrase to some remote link server. This interpretation of the *get selection* operation is quite natural,

and perfectly adequate for most situations. Yet there are many situations when an alternative understanding of this operation may be of more use.

Consider a user who encounters a document which has been typeset using some document authoring system, and stored using a rich page-description language, perhaps the PostScript language. Previous visitors to this node have located a useful service which can be used to view PostScript nodes, and the adaptive model has acknowledged the value of this service. When subsequent users visit the node, the HIPPO+ system now suggests this new PostScript viewing service in place of the default viewing component. The user decides that she wishes to find out more about a particular phrase that appears in the document, and tries to select this in the viewer. Previous discussions have suggested that alternative implementations of the *get selection* service may be useful. These retrieve the selection using different transport methods etc, but essentially share the same understanding of the operation, and return the selected word or phrase. However, in this case it is more important to consider exactly what the *semantics* of *"get current selection"* should be. Should the client adopt the usual semantics of this operation, and simply retrieve the words which appear in the phrase? This may be perfectly sufficient for most users, but is completely inappropriate for other tasks and user groups.

A typographer may be more interested in the fonts which are used to typeset the selected phrase. Alternatively, linguists may want the phrase translated into their native language. The format used to represent the node may have no notion of characters and words, and may store the text as a graphical image, in which case the selection operation may have to invoke some form of optical character recognition. If users select an area of a graphical image, what do they want returned to them? Do they want the graphical data stream, or do they simply want to retrieve some characteristics of the object (eg. colour, dimensions, orientation etc). Similarly, other sections of the user community way have completely different interpretations of the *"current selection"*. Graphic designers may be more interested in the abstract definition of graphical objects – if they select an area of a drawing which resembles a circle, then they want it to be recognised as such, and a mathematical definition of a circle object to be returned to them, instead of the graphical data. On the other hand, if an architect or engineer selects a drawing of some building, they may want a virtual reality representation to be generated which can be explored and interacted with (figure 6.5).

Even the simplest of operations such as *get selection* can have different meanings and semantics in different situations. An adaptive model should attempt to identify the most useful interpretations and implementations for each task, and suggest these to future users. The precise functionality of the system (ie. the services which are bound to interface buttons) should not remain fixed, but should change and adapt as the user explores different areas of the hypertext. Different tasks and

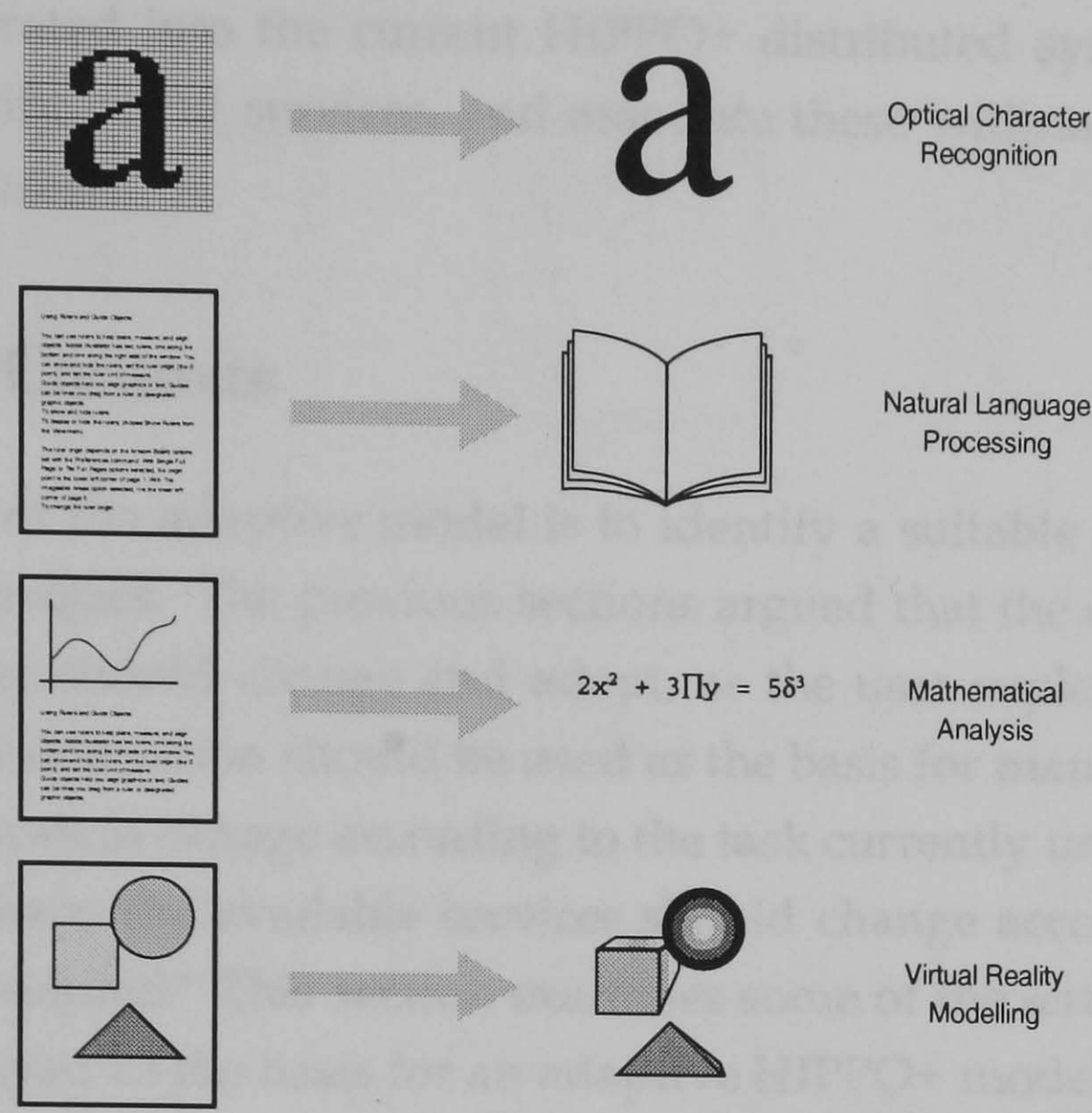


Figure 6.5: Alternative semantics of *get selection*

nodes have different requirements, and an adaptive model should reflect this. It is important that the experiences of users are shared amongst the user community, so that useful services are emphasised and reused.

The previous examples have attempted to show the benefits of an adaptive model in a distributed open hypertext system. The HIPPO+ system identifies common operations which are fundamental to a hypertext system, and implements these as remote objects. The system developer decides which hypertext operations are considered *"fundamental"*, and makes these available to the user as buttons in the client.

The discussion has shown the limitations of this approach, and suggests that implementation of these default operations should not remain fixed. The mapping between buttons and implementations should change to reflect the task at hand and the type of user. This idea has much in common with the work on activity spaces [HHL 92] and multi-modal hypertexts [GMP97]. The previous examples have showed how common operations can have widely differing interpretations and semantics, and that the system should use the appropriate remote service to implement the operation. An adaptive model can observe which services are used by users, and can use this information to identify the most useful implementations. These can then be incorporated into the HIPPO+ system for the benefit of future

users. The remainder of this section describes a proposed adaptive model which could be incorporated into the current HIPPO+ distributed system. This model attempts to identify useful services, and associate these with tasks and hypertext nodes for future users.

6.3 Service Contexts

The first priority of the adaptive model is to identify a suitable context for applying adaptive techniques. The previous sections argued that the services which are offered to the user should change and adapt, as the user explores the hypertext. However, which information should be used as the basis for managing this adaptation. Should the system change according to the task currently under consideration by the user? Perhaps the available services should change according to the node currently being explored? This section examines some of the *service contexts* which have been considered as the basis for an adaptive HIPPO+ model.

6.3.1 User Stereotypes

Perhaps the simplest method of supporting an adaptive model is to adopt a *user stereotype* model. This approach was introduced in section 2.3.1, which identifies general categories to describe some features of the user. These could be arranged into broad categories (*novice, intermediate, expert)*, or more specific roles (*architect, engineer, graphic designer, mathematician* etc). This would be simple to implement in the HIPPO+ system. The user could specify a suitable category/categories which in some way represents the user, and the system could load a corresponding *service profile*. This profile would define appropriate mappings between abstract operations (eg. *view node, traverse link* etc), and the corresponding remote service implementation (figure 6.6). Different user stereotypes would include different service mappings, and would use remote services that were considered more appropriate to the particular type of user. Similarly, if a user found a particular service to be useful, then this could be added to the *service profile* for the corresponding user category. In this way, the set of service mappings for each user group can change to adapt based on the experiences of the users.

However, this stereotype model seems to offer only limited adaptability. Service mappings can only be specified at a very general, coarse level of granularity, and do not allow the services to be tailored to the underlying information. It seems unreasonable to suggest that a single collection of services (albeit, tailored to the particular type of user) will be suitable for an entire hypertext. Users will want to use different services as they encounter different nodes and new sets of information. Furthermore, it is unlikely that the needs of any collection of users are the same, and

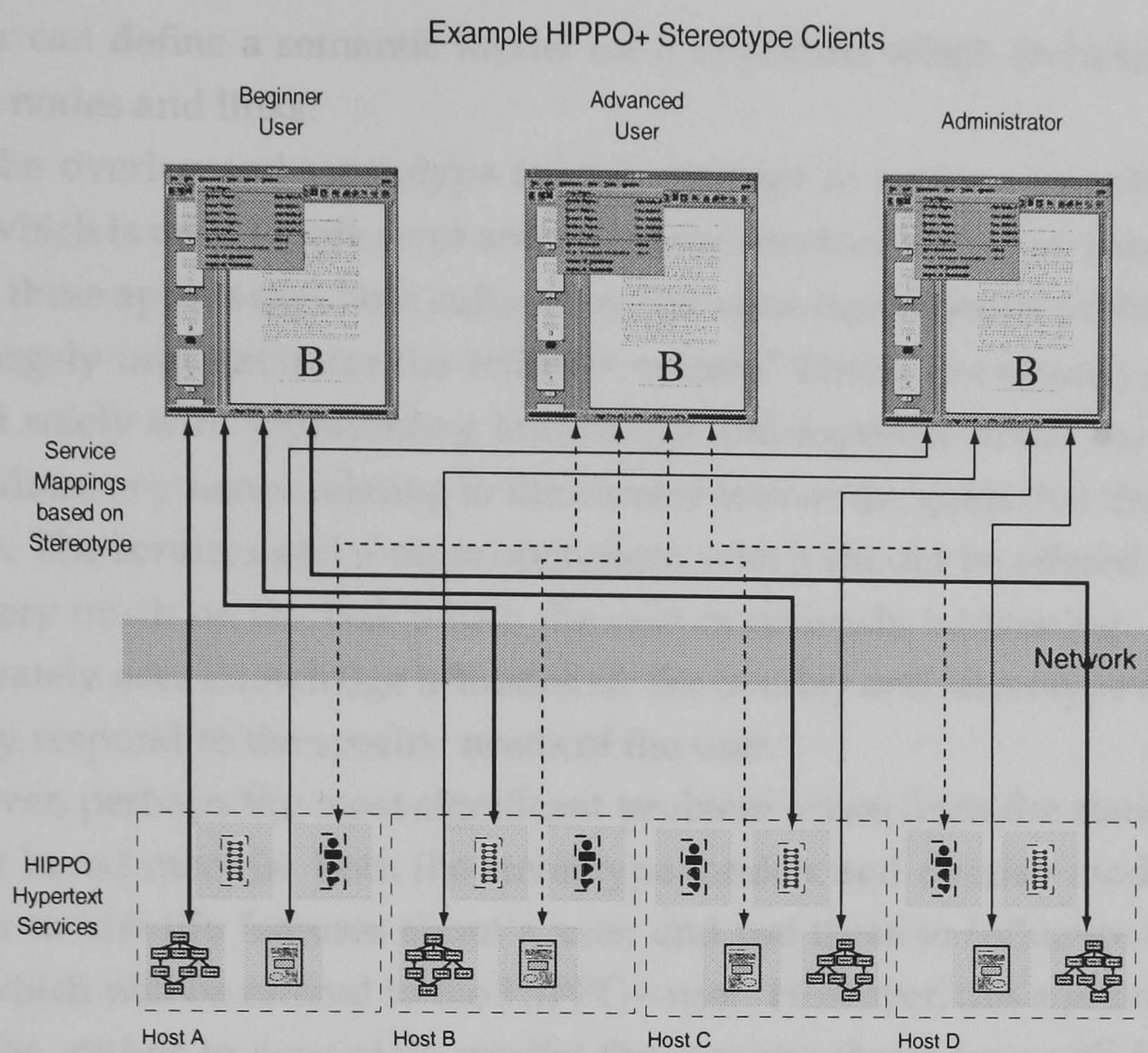


Figure 6.6: Stereotype model matches user categories to sets of service implementations

can be generalised across any one stereotype. The requirements of each individual change constantly between sessions, as the user gains new experiences and encounters new tasks. Section 2.3 also explores the limitations of adaptive systems based on a *user stereotype* model.

Section 2.3 introduced another approach to representing knowledge about the user, using an *overlay model*. This attempts to structure a knowledge domain as a series of related concepts arranged into a semantic network. This approach would offer a finer level of control over services, and allow the HIPPO+ system to adapt as the user moves through the hypertext. The author could construct a semantic network to represent the concepts and ideas which are expressed in the hypertext. The system must provide some means of measuring the user's knowledge of each of these concepts, which could then form the basis of the adaptive model. The author could associate particular implementations of services with certain concepts in the semantic network.

An adaptive model based on an overlay model seems to offer a more intelligent way of modelling knowledge about the user, and provides a finer level of control over adaptation. However, an overlay model adds significant overhead to the authoring process. An author must construct a detailed semantic model for the entire network, which is a complex and non-trivial task. Furthermore, it is unclear how

the author can define a semantic model for a hypertext which includes dynamic, non-static nodes and links.

Both the overlay and stereotype models attempt to capture knowledge about the user, which is used to influence and adapt the services which are used HIPPO+. However, these approaches both suffer from the same fundamental problems which make it largely unsuitable for the HIPPO+ system. Firstly, user-based models are concerned solely with representing knowledge and expertise about the user. This fails to address any issues relating to the current task or the goals that the user aims to achieve. The services and remote operations which should be offered to the user depend very much on the task which the user is currently addressing. No matter how accurately user knowledge is modelled, the overlay and stereotype models can never fully respond to the specific needs of the user.

However, perhaps the most significant problem arises from the static nature of these user based models. Both the stereotype models and overlay models require the author to identify features about a user, and use these to influence the remote services which will be offered to the HIPPO+ user. However, this *a priori* definition requires the author to accurately predict the services that a user will find useful. Even if the author can accurately model knowledge about a user, the author must still anticipate the particular remote services that should be used. Yet this is precisely the reason why an adaptive model is proposed in HIPPO+ – to remove these responsibilities from the author. The true value of remote services is unclear, and only emerge as the users interact with the hypertext. The precise implementation of a service which would be of most benefit to the user can only become apparent after the user has explored the hypertext and used different services. Indeed, the value and importance of services can change over time as the user becomes more experienced and attempts new tasks. While the author clearly has some role in suggesting useful services for the users of a hypertext, he should be considered as just one of *many* voices. An adaptive model cannot rely too heavily on predefined, *a prori* models which may prove inaccurate. For these reasons, while a stereotype/overlay model can be useful and may be combined with other approaches, it is not considered sufficient for an adaptive HIPPO+ system.

6.3.2 Goal-Based Model

The user stereotype model can be useful as a general guideline to the type of services which will be required by the user. The background of the user will influence the type of services that he will require, and provides some sort of focus for an adaptive model. However, the stereotype model does not allow sufficient control of the services which are offered to the user. An overlay model offers a more flexible model which provides a complex knowledge representation. However, both models completely ignore the goals of the user and the nature of the current task.

Each user uses the hypertext to solve a particular problem; each user has different requirements and demands which change according to the situation. This suggests that a better adaptive model could use the current task as the basis for adaptation.

Each user could outline the nature of the problem which they are exploring, and attempt to describe the task that they are attempting to solve through the hypertext. The author could then associate particular service implemenations with particular goals, and use this to implement an adaptive model. The HIPPO+ system would then incorporate the appropriate set of service mappings and *service profile* to reflect these tasks. This approach offers a fine level of adaptive granularity, by allowing users to define rich task descriptions. The mappings between abstract hypertext operations and actual service implementations can be tailored to the current task at hand, to suit the demands of the user. The system could offer task descriptions at an arbitrary level of detail – engineering, engineering mathematics, mathematical modelling, mathematical modelling of heat flow etc. Furthermore, users can alter their current task, and change their task profile at any time. This allows the set of service mappings to dynamically change as the user moves through the hypertext.

This adaptive model based around some *task profile* seems to offer a more intelligent context for changing HIPPO+ services. The services can be adapted to suit the goals of the user, and the user can change their task descriptions at any time. However, while this approach offers a finer degree of control over the adaptation, it can be difficult to provide suitable descriptions for each task. For example, consider a graphic design artist who is exploring a hypertext of documents, and is attempting to see how typographic standards change over time. The needs of this user, and the hypertext operations they require are very different from a casual reader who is exploring a hypertext from a literary point of view. However, the HIPPO+ system must provide suitable ways of describing these two points of view. Hypertext models can be applied to potentially limitless situations and problem domains, and it is impossible to provide task definitions for every possible goal which might be anticipated.

A goal-based model seems to be too exhaustive to implement in the current HIPPO+ system. Furthermore, a goal-based model still requires some *a priori* model which is defined by the author. The author must define a complex set of goals for each hypertext, then decide on an appropriate set of services to be used in each of these cases. As discussed in the previous section, the aim of an adaptive model is to move away from these predefined, static *a priori* definitions towards a more emergent adaptive model. This model should not predetermine which services will be used in particular cases, but should allow useful services to emerge over time. This has led to the development of an adaptive model based around the nodes themselves – *document objects*.

6.3.3 Document Objects

The previous sections have suggested *overlay/user stereotyping* and *goal-based* models as a platform for managing and adapting HIPPO+ services. The *user-stereotype* model is simple to implement, but seems too restrictive for a general hypertext model which is to be used in diverse situations. The *overlay model* offers a richer knowledge representation, but seems to add an unacceptable overhead to the authoring process. The construction of a semantic model is a complex process, and raises the problem of accurately measuring the user's knowledge of each concept. A *goal-based* approach offers a similarly rich representation which has the added advantage that the services which are suggested to the user can be tailored to each particular task. However, the range of tasks which could be expected is too overwhelming for a direct implementation of this model. Furthermore, both approaches suffer from the same limitation that the author must provide *a priori* definitions. The author must decide in advance which services are most appropriate to particular tasks or types of users. While the author clearly has some level of expertise, she cannot be expected to accurately identify optimal services in all cases. An adaptive model should involve the user in establishing an adaptive framework, so that the value of particular services can emerge over time.

This final approach introduces the idea of *document objects* and attempts to use the hypertext nodes themselves as the basis for an adaptive model. Section 6.2 included some example scenarios which showed how the needs of different users can differ dramatically for the same hypertext. Some users can find particular operations more useful than others, and different users can have widely differing interpretations of even the most fundamental hypertext operations (*view node, store link, get anchor* etc). The profile of the user has a big effect on determining the value of a particular hypertext service – the background of the user, level of expertise, goals and tasks etc. These have been widely used in a number of adaptive hypertext systems (section 2.3.1), and have been discussed in the previous section. However, these approaches fail to recognise the importance of the underlying data which is being examined. The examples in section 6.2 showed that the choice of services, and the mapping of hypertext operations to remote objects often depends heavily on the current node which is being visited. The service which is used to view a node is (partly) dependent on the data format of the node contents. The service used to retrieve the current selection is often dependent on the viewing service etc. Furthermore, it seems natural for the collection of hypertext services to change and to adapt as the user moves from node to node.

This adaptive model suggests that the document node itself is a useful platform for an adaptive model, and could form the basis for deciding which services are presented to the user. The current prototype defines the HIPPO+ system as a fixed

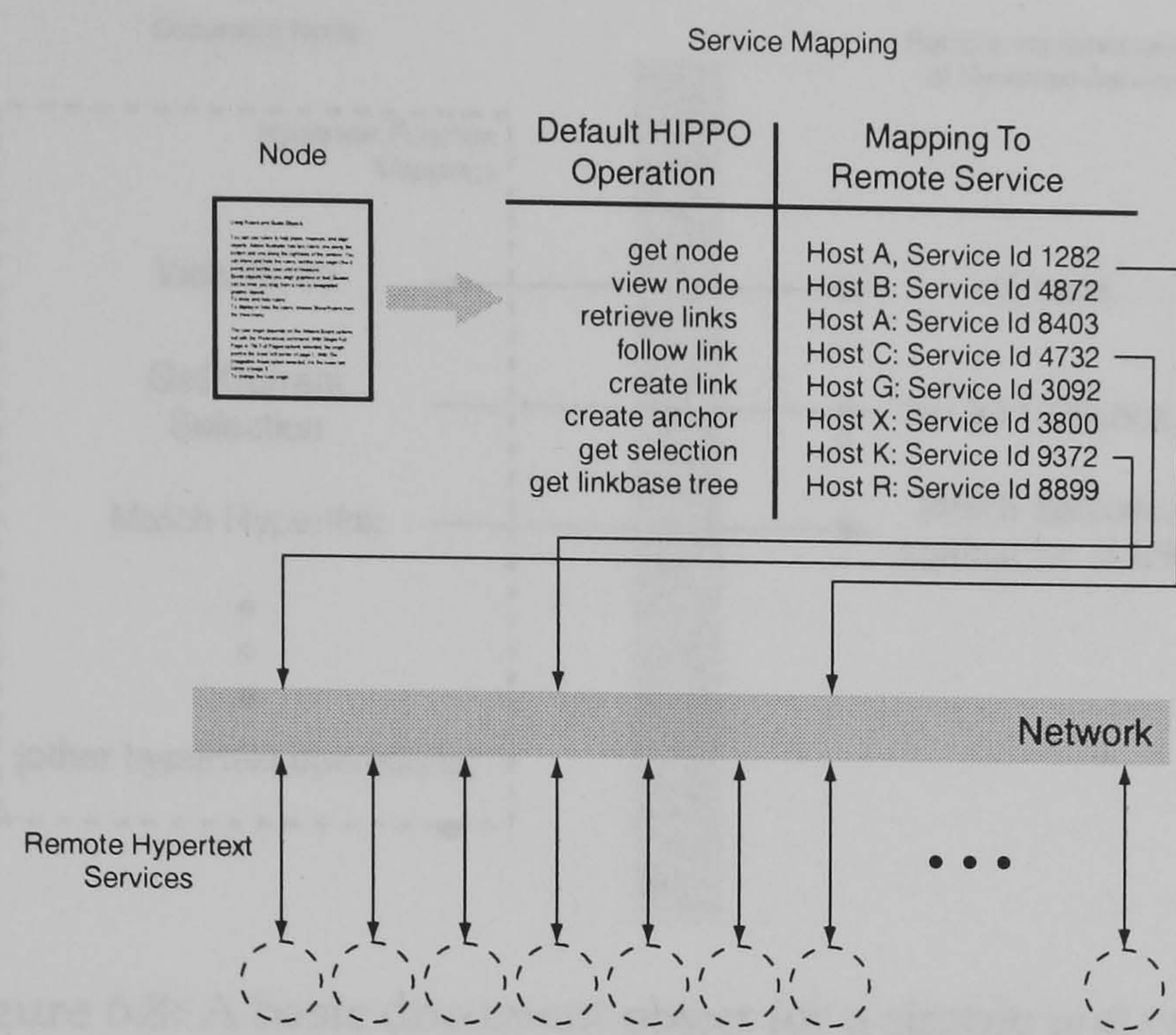


Figure 6.7: Each node has an associated service profile

set of remote services, which are mapped to predefined buttons in the HIPPO+ client. Users are free to select additional services which they may find useful at any time. However, this model suggests that nodes should also have a *service profile* associated with each node. Each node contains, not only the data contents, but also a set of definitions which map abstract hypertext operations on to particular instances of remote services. These mappings are tailored to the particular node, and allow the system functionality to change, depending on the current node (figure 6.7).

6.4 An Example Using Document Objects

Consider a hypertext which is used to provide on-line lecture notes for a Computer Science programming course. The hypertext allows students to access notes, programming examples, additional tutorials etc. A simple node example might have a textual description of some introductory programming concepts. Hypertext operations are easy to support in this case, and the HIPPO+ system could use simple remote services to implement the hypertext. A basic text-only editor could be used to display the node contents, and a clipboard service might be used to retrieve the current selection. The user could then submit these selections to a simple computer glossary service, which returns links to further information and definitions. The user might select a computer acronym which they are unfamiliar with, and pass this to the remote link service. The service would then return a list of nodes provide additional information about the selected term. All these mappings of abstract hypertext operations on to actual remote services are no longer fixed, but are asso-

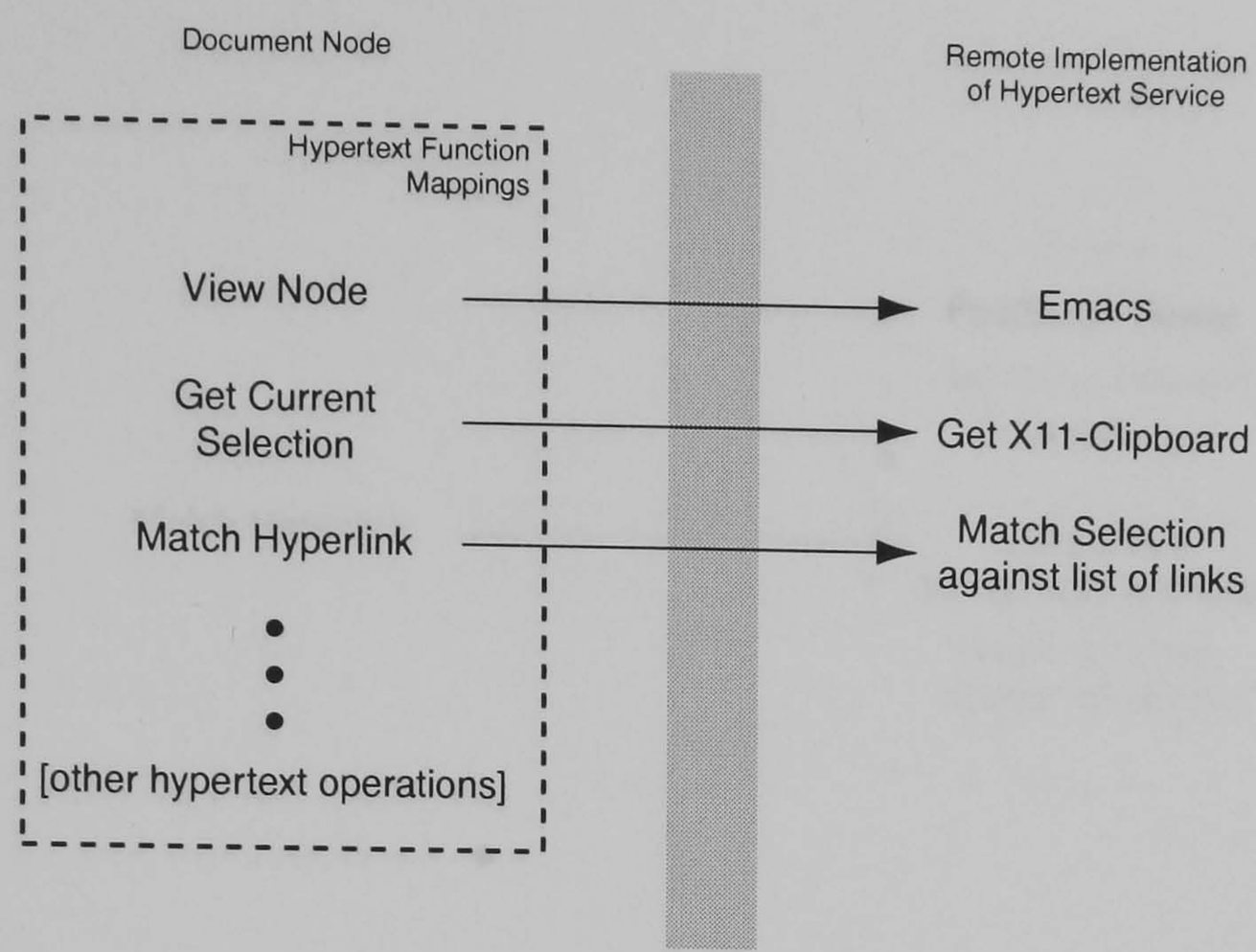


Figure 6.8: A basic document object for a simple text node

ciated with this particular node. This *document object* might resemble that shown in figure 6.8.

The functionality offered by HIPPO+ using these basic remote services is perfectly sufficient for viewing this text node. Items can be selected, and hypertext links are provided for locating further definitions and other supplementary information. However, the user may then move to another node in the hypertext which contains more complex node contents. Perhaps the node contains some archive material which has been scanned in from an academic journal? The article may describe the history of some programming languages, and was considered useful by the lecturer, as a way of placing the course in some kind of historical context. This node is stored in a graphical format instead of the usual text representation – perhaps using PostScript or some graphic image format. Clearly, the services which were chosen to implement hypertext operations in the previous example are no longer so useful for this node.

The most obvious requirement is to provide a suitable viewing service – the text editor which was used in the previous example can no longer provide a meaningful presentation of the graphical node contents. Instead, a graphics package or a PostScript viewer might be considered more useful. However, the services which are required to provide a meaningful hypertext extend much further than simply providing a suitable viewer. For example, the existing method of selecting regions of the page which copies the text phrase to the clipboard is no longer appropriate. Perhaps a more complex *get selection* service is more useful which selects a region of a window, then performs some *Optical Character Recognition* on this graphical data? The resulting text results could then be copied to the the clipboard in the usual way, or using some alternative transport mechanism. Similarly, a different linking model

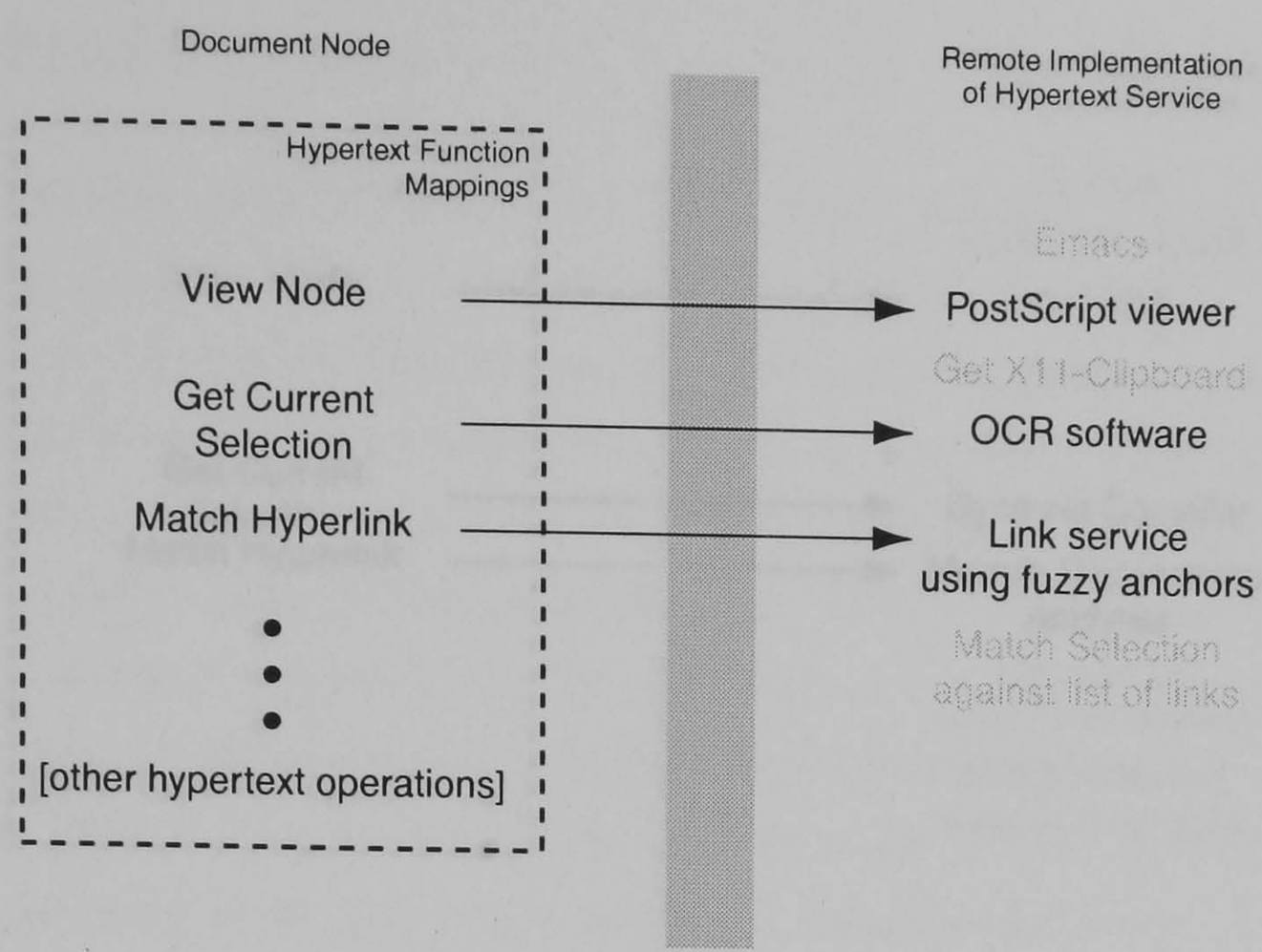


Figure 6.9: New document object for graphical archive node

may be more appropriate than the current link service which simply matches text phrases against link definitions. A link model based on fuzzy anchors offers some advantages in graphical environments so might be worth considering (see Chapter 3). Perhaps a fuzzy linking service could be used in place of the traditional basic linkbase service?

An adaptive model should identify which services are considered most useful with each node. The system can then alter the mappings between abstract hypertext operations (buttons in the HIPPO+ client) and actual remote implementations. A document object is associated with each node, which defines these mappings (figure 6.9).

We have seen how the services which are invoked by the user to implement particular hypertext services can change as the user moves to a new node. The requirements of the initial introductory textual node are very different from the scanned image of some archive material. Similarly, we could identify other nodes in the hypertext which also have very different needs, and demand alternative hypertext services.

After the lecturer has discussed some programming aspects, he may decide to include an example program to demonstrate the techniques that have been learned so far. The node can be viewed quite satisfactorily using a text browser, and can be treated in the same way as other textual documentation. The user can scroll through the program code, and view it in the usual way. However, this approach completely ignores the true nature of these node contents – a programming example which represents a truly executable object. It seems far more sensible to emphasise these semantics, and to provide suitable hypertext services to support this. A viewing service could use an integrated program development environment to view the

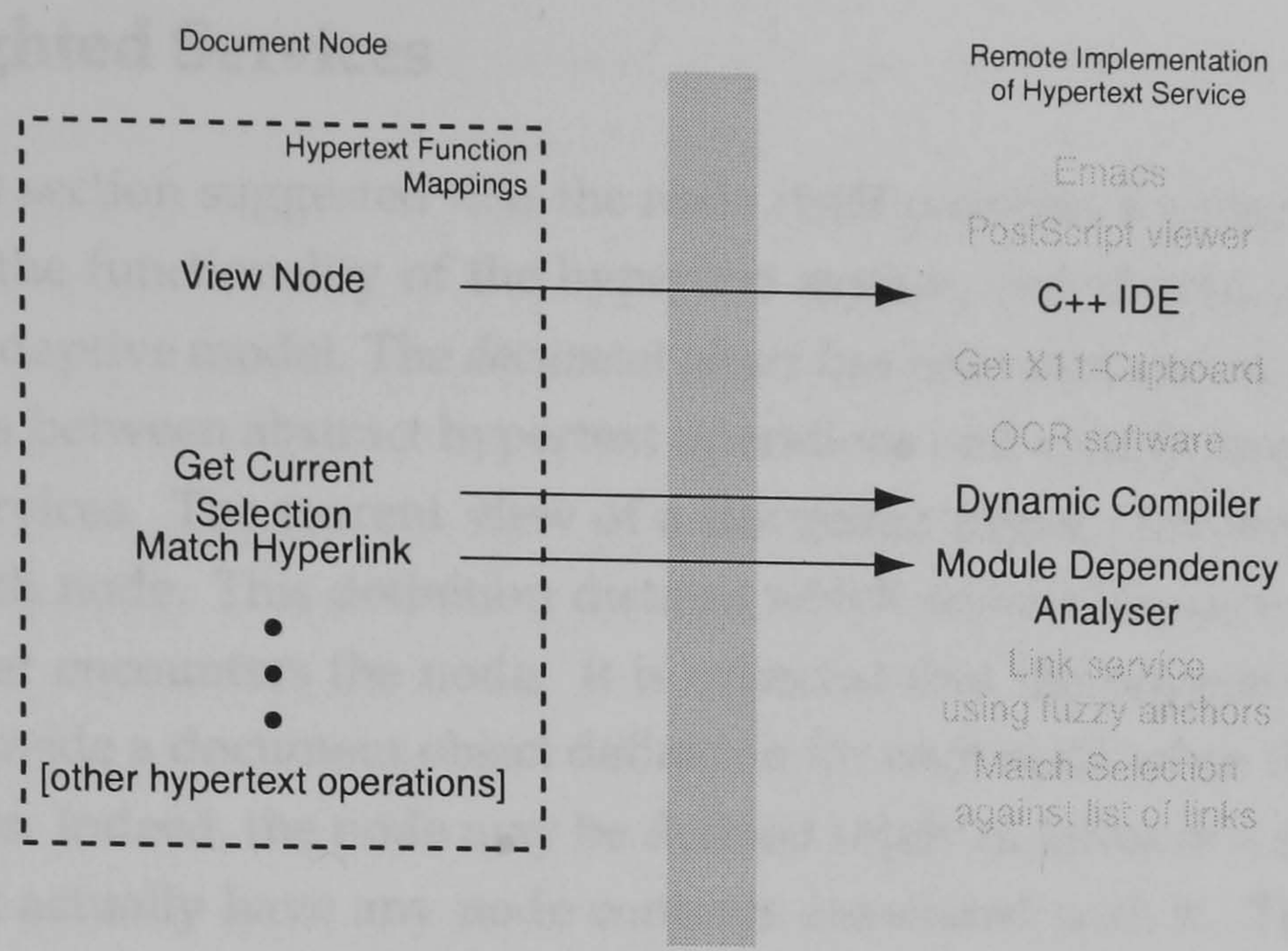


Figure 6.10: Document object for programming example

program example, instead of a simple text editor. The user can then see the program in a more natural environment, where they can use the features of the development environment to explore the program further.

Similarly, when users select a section of the program, they may not want the contents to be copied to the clipboard. The user could benefit from a much richer definition of selection, which extracts the selected text and compiles it into an executable object. This can then be executed separately from the main program, to see the effects of this small section of code. When a user follows a link, the conventional linkbase service does not seem so appropriate. Perhaps a link service should dynamically evaluate the dependencies between the program example and other code modules in the system. The link service could then return a list of related examples and code extracts. Figure 6.10 shows a document object containing hypertext services which are more suited to a programming example.

This section has included an example hypertext which is used to explain computer programming techniques. The idea of *document objects* has been used to tailor the hypertext services to the node itself. The precise implementations and semantics of abstract hypertext operations such as *view node, get selection, follow link* can have widely differing interpretations between different nodes. The node itself has an important role to play in defining the functionality of a hypertext system. Each node presents different information to the user – not simply the format used to represent the data, but also in a deeper sense. A hypertext system should provide different services and implementations of hypertext operations which are tailored to each node. This section has attempted to show the value of an adaptive model which uses the *node* as the basis for adaptation. Section 6.5 shows how these *document objects* are developed to support an adaptive model using *weighted services*.

6.5 Weighted Services

The previous section suggested that the node itself provides a suitable abstraction for defining the functionality of the hypertext system, and should be used as the basis for an adaptive model. The *document object* has been introduced which defines the mappings between abstract hypertext operations and specific implementations of remote services. The current view of a document object provides a fixed definition for each node. This definition dictates which service instances will be used when the user encounters the node. It is expected that the original author of the node will provide a document object definition for each node when they create the node contents. Indeed, the node may be defined solely in terms of a set of services, and may not actually have any node contents associated with it. This allows the contents to be generated dynamically, or retrieved from remote sources or as the result of service invocations. The author has a good knowledge of the particular problem domain, and will have a clear idea of the kinds of services which will be most useful to the user. When the user reaches a node, the document object object will be used to map these remote instances on to the abstract hypertext buttons in the client.

However, this approach places great demands on the hypertext author. Not only must authors meet the usual demands of hypertext authoring – creating node contents, identifying suitable hypertext links etc – but they must now also provide a set of useful service mappings. The author clearly has a role to play in defining the services which will offer the greatest benefits to the user, yet the authors opinion must be considered as simply one contribution. The value of services and particular functionalities in a hypertext system often only becomes clear by using and interacting with the hypertext. Some navigational tools may prove themselves to be particularly useful when viewing certain types of node data, while they may be less important for general use. Similarly, certain viewing tools, linking models, selection mechanisms etc may of little interest initially, but prove to be invaluable for certain nodes and documents. The previous chapters have shown the benefits of applying adaptive modelling techniques in these kinds of situations. The importance of certain abstractions can emerge gradually over time, in response to the experiences of users. The author can still offer initial suggestions, yet these can be modified in light of new information and feedback from the user.

The adaptive model which is described here uses the initial definitions of document objects as a *starting* point in the hypertext system. Each set of service mappings is not viewed as fixed, absolute definitions, but can be adapted and modified to suit the needs of the users. The model achieves this by introducing a means of expressing the *importance* of remote hypertext services for each node. The current discussion of document objects requires each abstract hypertext operation to

be mapped on to a *single* remote service. Each remote service in the HIPPO+ domain is either referenced in a document object definition, or remains unused. The authors of document objects are forced to make difficult decisions as to which remote services should be included in each node document definition. The author must choose a single service to be used for a node, and to discard any other service implementations.

This is a very unrealistic view of the authoring process which views remote services as discrete objects. While it is true that some operations may map easily on to particular service instances, there are many cases then these decisions are more problematic. It is not always obvious which service implementation should be used with a particular node. Often, several services may be considered appropriate and the author is forced to choose between them. Consequently, the author may well select a particular service which later turns out to be less useful than originally anticipated. Some of the other services which were considered may have turned out to be a better choice. Indeed, this is the main problem with the current definition of document objects. The importance of a particular remote service can only be truly evaluated over time, after users have interacted with the hypertext. Furthermore, the importance of services may well change as the expertise of the user increases. As each user becomes more familiar with the hypertext, or changes their particular goals and tasks – so too, the services and functionalities provided by the hypertext system must change.

The adaptive model described here addresses this problem by allowing document objects to contain, not just one, but *several* mappings for each hypertext operation. Each abstract hypertext operation (eg. *view node*) can contain mappings to multiple service implementations. Each of these mappings then has a weighted value which reflects the relative importance of the relationship. For example, figure 6.11 shows a document object definition for a particular node. The author may decide that a simple linking model will be sufficient for most users, which provides a basic link service. Users pass a textual selection to the service, and the remote object returns a list of matching link definitions. However, the author may also decide that an alternative linking service may be useful in some cases – perhaps a typed link model which requires the user to request links of a particular type? Perhaps a linking model which uses the identity of the user to influence the list of links which are returned? This weighted document object model allows the author to express each of these suggestions, by attaching weighting values to each mapping. The author is not forced to choose any one single service, but can suggest that all three services may be of use. A weighted model can capture the thoughts of the author more closely, and express the uncertainty which the author experiences when choosing remote service implementations.

This weighting system provides a way of defining mappings between remote

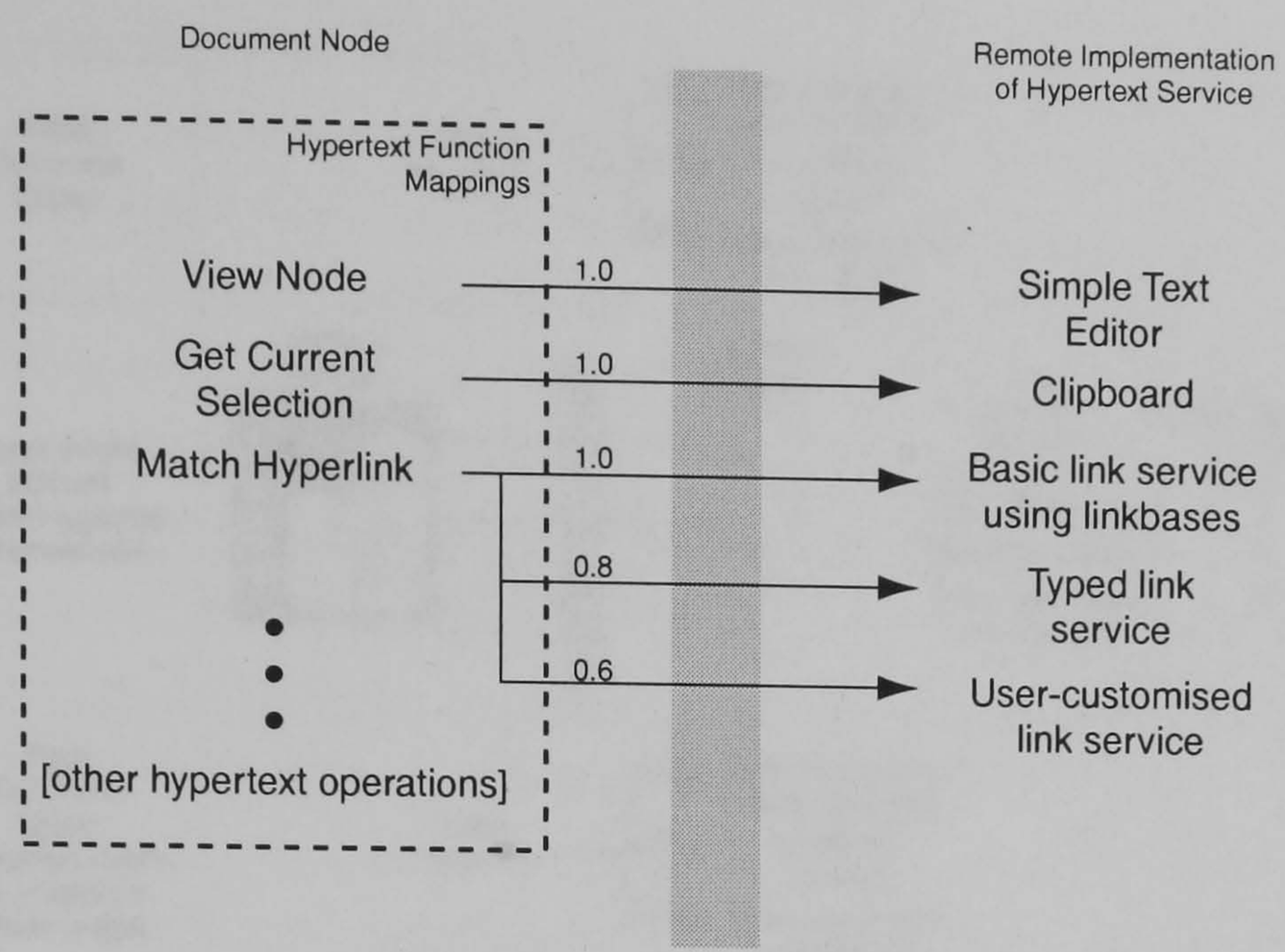


Figure 6.11: Weighted document objects

services, and also a means to express the *importance* of these relationships. Each weighted value represents a kind of attraction between a node and the surrounding hypertext services – services with high weightings are considered more important than those with correspondingly lower values. This could be viewed using an analogy with chemical modelling, where these weighted relationships correspond to the strength of bonds between atoms and molecules. The node could be seen as a nucleus, surrounded by other services with varying degrees of *attraction*.

The author can suggest several services which may be of value for each node, and the user can select the service with the highest confidence value. More importantly, these weighted values can be adapted and changed as the value of each service becomes apparent. Users may find that a particular service for viewing mathematical models becomes particularly useful for a certain node in the hypertext. The adaptive model could then increase the confidence value which is associated with this service. Figure 6.12 shows an example of this adaptation which increases the weighting associated with a particular service, as its importance becomes clearer. The confidence values which are associated with less useful services can be correspondingly reduced.

6.6 Automatic Identification Of Useful Services

These weighted relationships reflect the importance of each service for a particular node. Those services which show themselves to be particularly useful are given increased confidence values, while less important services are reduced. This raises the problem of how to identify useful services. Figure 6.11 showed a document object definition for a node, which suggested three link services that might be useful. The

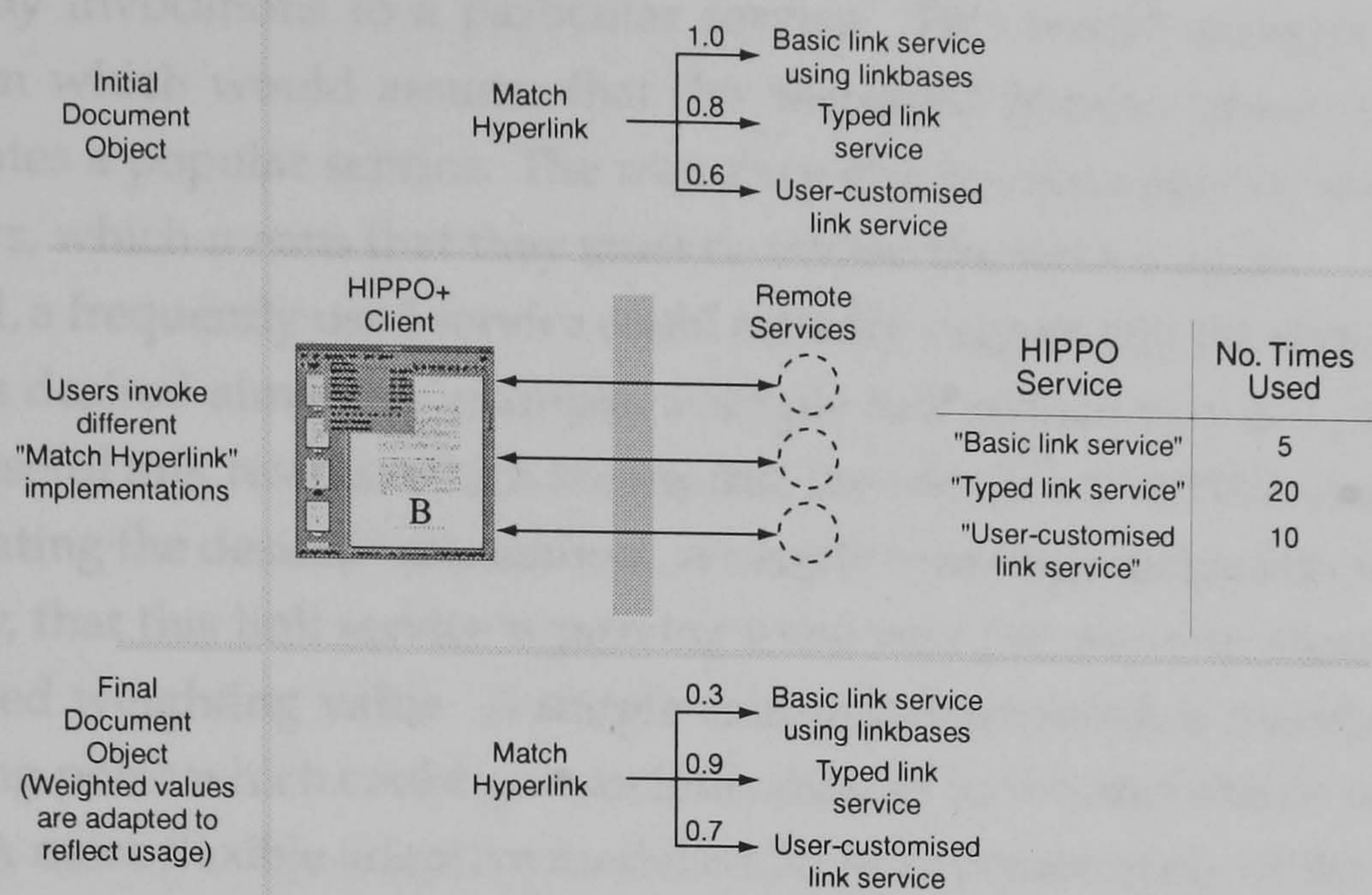


Figure 6.12: Adapting weighted service values

author felt that each of the services could offer some benefits to the user, but was uncertain which the user would find most useful. The author finally decided that the basic linking service would perhaps be most widely used, and assigned this the highest confidence value. The adaptive model needs to test this hypothesis, and find some way to ascertain which service really is the most useful.

A simple approach could simply observe how much each service is invoked. The popularity of each service would suggest how useful the user found the service. An important component will be widely used, while less useful services are neglected. Each remote service would be required to maintain an internal counter which is incremented each time the service is invoked. Alternatively, a separate *audit server* could be implemented which runs centrally, and maintains usage statistics for each service in the network domain. In this case, each time a client invokes a service, they also notify the *audit server* which increments the corresponding service counters etc. Each node would also require some unique identifier, so that the audit server knows which service was used with which node. This raises additional issues relating to unique naming services, which are not discussed here. Readers are referred to other work on naming schemes [WWWb] and other hypertext systems which use node identifiers (eg. EHTS [Wii91a], Dexter [HS90] etc).

This is a simple approach which suggests a useful starting point for an adaptive implementation. However, the frequency of execution is not always an accurate guide to the real value of a service. This ignores any failed executions and also ignores many of the subtleties of using a hypertext system. The user may invoke a service, before deciding that this does not actually achieve the desired goal. Some operations which would be logically considered as one operation may actually in-

volve many invocations to a particular service. This would mislead a counting mechanism which would assume that this increased number of executions actually indicates a popular service. The user may also invoke a service with incorrect parameters, which means that they must re-invoke the service again.

Indeed, a frequently used service could actually suggest that the service is failing to meet its desired aims. For example, a simple link service may not produce particularly useful link results, which means that the user has to submit many requests before locating the desired information. A simple counting mechanism would infer, incorrectly, that this link service is proving to be very popular and should be given an increased weighting value. A simple counting mechanism is merely suggested as a starting point which could give an indication as to the importance of particular services. A more flexible adaptive model could incorporate many of the techniques which have been used existing adaptive models (Section 2.3). Chapter 7 also discusses some directions for future development of an adaptive HIPPO+ model.

6.7 Combining With Other Adaptive Models

It seems sensible to combine this *node-based* approach with some of the previous models which have been discussed. This allows the adaptive model to use additional information about the user to influence the choice of services – background, knowledge, experience, goals etc. The user stereotyping model in particular would be simple to incorporate into the document object model. The current model associates a single document object with each node, which contains suggested mappings between abstract hypertext operations and actual service implementations. This model could be extended such that each node could have multiple documents objects associated with each node. Each of these alternative document objects represents a set of service mappings for a particular type of user – eg. *beginner, intermediate, expert* or *scientist, artist, end-user* etc. The user could then select an appropriate document object based on their user category (figure 6.13). The implementation of the adaptive mechanisms would also have to be extended to adjust confidence values based on the type of user executing the service.

6.8 Summary

The HIPPO+ model provides a distributed implementation of the original HIPPO prototype. The system identifies each key hypertext operation and function, then implements each of these as a remote lightweight service. This approach provides an open and extensible environment, which shares many of the advantages of distributed systems. Chapter 5 discussed the HIPPO+ implementation, and described the tools which have been developed to help the user manage these remote ser-

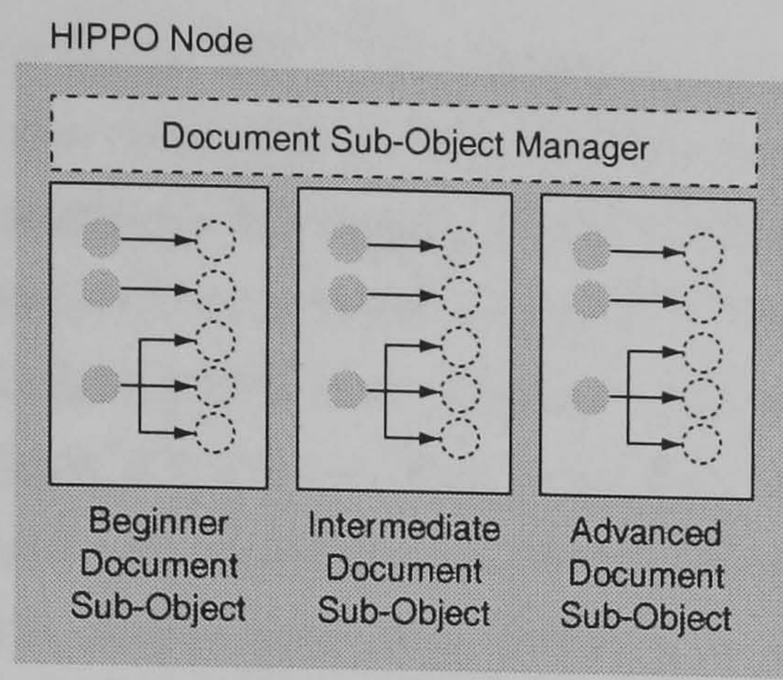


Figure 6.13: User stereotyping using multiple document objects

vices. The computational model has also been extended to allow users to select arbitrary remote services to replace the default, predefined remote services. This allows users to choose the most appropriate implementations of remote services to match the task at hand.

This additional flexibility can prove overwhelming for many users, as they search through potentially hundreds and thousands of objects to find suitable implementations of hypertext operations. This chapter has suggested an adaptive model which could be incorporated into the HIPPO+ system to help manage this complexity. These ideas have not been implemented in the initial version of the HIPPO+ prototype due to lack of time, although some implementation suggestions are included where appropriate. An adaptive model can be used to identify useful services, so that these can be suggested to future users. Section 6.1 discusses some of the advantages of an adaptive model when incorporated into the HIPPO+ environment. Section 6.2 includes some example scenarios which demonstrate how an adaptive model might be used.

Section 6.3 begins by exploring some of the different approaches to developing a suitable adaptive model. A user-based model is considered which attempts to capture knowledge about the user to influence which services whill be offered to them (section 6.3.1). This offers only a limited level of control over services, and proves too general for a widely applicable adaptive model. An overlay model extends this approach by modelling user knowledge using a semantic net. This adds considerable overhead to the authoring process, and relies on the author being able to accurately identify useful services. A goal-based model allows the choice of HIPPO+ services to be tailored to the current user task (section 6.3.2). This allows the mappings between abstract hypertext operations to change between user sessions, as the user explores different problems. However, this also places significant demands on the author, and fails to acknowledge the value of the underlying node contents in a hypertext.

The value and importance of remote services is often determined by the type

and content of nodes in a hypertext. The choice of viewer is often a function of the content format, and the selection of link service is heavily influenced by the semantics of each node. Section 6.3.3 suggests that the hypertext nodes themselves should be used as the basis of an adaptive model. The idea of *document objects* is introduced which associate particular services with a given node. Each node has a document object which determines the mappings between abstract hypertext operations in the client, and actual remote implementations of services. Section 6.4 includes an example which shows how these document objects could be used.

Document objects allow the choice of HIPPO+ services to be tailored to the underlying node contents. It is expected that authors will provide these document object service mappings, when they create the actual node contents. The author has a good understanding of the node semantics and is well placed to select services which the user will find useful. However, this places significant demands on the author, and the author is not always best placed to select services. The true value of services often emerges over time, and should involve the user in some way. Section 6.5 introduces the idea of *weighted services* which associates confidence values to reflect the importance of services. Section 6.6 then goes on to discuss how this could be used to identify useful services, and to form the basis of an adaptive model.

The HIPPO+ model identifies key operations in a hypertext environment, and implements these as remote services. The lightweight HIPPO+ client provides interface buttons which map on to particular remote implementations. The user can also select from other services, and use these to augment the existing default services. However, the author envisages a HIPPO+ environment which contains hundreds, perhaps thousands of services, and the user can experience problems deciding which services to use. This model shows how the mappings between abstract operations and remote service implementations can be modified to reflect the value of particular services. The system does not place sole responsibility for choosing appropriate services with the author. The true value of a service is not always immediately apparent, and can only be evaluated in light of user experiences.

This adaptive model attempts to capture the uncertainty and ambiguity which is part of developing hypertext applications. The choice of which operations to offer, and the implementations to use is not always clear. This adaptive model attempts to provide a limited way of moving some of the responsibility away from the author. The author provides an initial definition for a system which dictates which services will be used with a particular node. These can be used as a starting point, and can be adapted to towards a more optimum set of services. This is a significant departure from other adaptive hypertext systems which focus on the adaptation of the user interface and link definitions etc. The HIPPO+ adaptive model attempts to provide a *deeper* level of adaptation which modifies the actual semantics of hypertext operations. The adaptive model described in this chapter has not been implemented

in the current HIPPO+ prototype, although could be incorporated in future implementations. This adaptive model has not been implemented in the current HIPPO+ prototype, and Chapter 7 suggests a way in which this could be implemented using compound documents (see Chapter 5 for details of compound document models).

Chapter 7

Discussion

The previous chapters have discussed the research which I have undertaken as part of the HIPPO project. The fundamental ideas behind the hypertext paradigm and the advantages of hypertext modelling have been explored. Some of the early hypertext implementations have been introduced, along with some of the more recent developments in the hypertext research field. Of particular interest to the HIPPO project are the ideas of open hypertext, distributed hypertext systems and adaptive modelling techniques. These have all been developed further, and incorporated into the HIPPO model which has been presented in this thesis. The main contributions of the HIPPO research have been described in more detail, in particular: *adaptive fuzzy anchors, adaptive linkbase inheritance trees* and a *distributed HIPPO model.* The thesis also includes a proposed adaptive model which has been suggested for the distributed implementation of the HIPPO system. This has not been implemented in the current prototype due to lack of time, and is left for future implementation. In this chapter I will discuss these ideas, and show how they were developed over the course of the research. I will also identify some areas for future research and explore ways in which the HIPPO model could be developed further.

7.1 Overview

The initial ideas in this thesis grew out of an interest in open hypertext which was beginning to gain increasing importance in the hypertext community. Open hypertext attempts to identify operations and abstractions which are fundamental to the hypertext paradigm, and separates these out into a link services subsystem. These link services can then be used to provide hypertext functionality to existing applications, which are not "hypertext-aware". I was particularly influenced by the Microcosm [DHHH92], Sun Link Service [Pea89] and Dexter [HS90] models which demonstrated the advantages of separating linking information from the underlying node contents. Links and anchors were no longer implemented at the interface

level, or embedded deep inside an application, but were instead treated as first-class, primary objects in the hypertext model. This had many practical advantages such as being able to integrate hypertext functionality into existing environments. However, I also felt that open hypertext systems extended the hypertext model itself, by forcing the designer to think about key hypertext abstractions, and to identify operations which were fundamental to a hypertext system. Some of the defining ideas in open hypertext research are discussed in Chapter 2. Appendix B also includes a summary of the open hypertext systems which have been influential in this research.

At the same time, I also became interested in the idea of distributed systems and distributed architectures. Distributed computing has been used for many years in all manner of areas – operating systems, remote access terminals, database systems etc – and was gaining popularity with work on distributed architectures such as CORBA [Objc] and ActiveX [Act]. Distributed ideas had also been incorporated into some hypertext applications (eg. KMS [AMY88], EHTS [Wii91a]). Chapter 2 also discusses other important distributed hypertext systems. More recently, the success of the Internet and, more importantly, the World Wide Web [WWWa] showed how distributed techniques could be used in the hypertext community. Distributed models offer many advantages over traditional, centralised systems, by providing more robust and scalable architectures. The WWW is an extreme example, which provides a global hypertext, on a scale not seen anywhere before. The WWW and Internet offered many exciting possibilities, and this was an area that I wanted to develop further.

I began my research on HIPPO by attempting to combine some of the open hypertext work with existing distributed techniques. Some systems such as PROXHY [Kac90], D^2 [HGC94] and SPx [LS94] had already offered some interesting insights into distributed, open systems. However, I wanted to extend this further, by providing a *completely* distributed environment, in which every operation, every resource and each component of the hypertext system, was accessed remotely. In particular, I wanted to see how this fine-grained approach could provide a more flexible model, by allowing users to choose components, and to define their own hypertext environments. This early work was to become the HIPPO+ prototype described in Chapter 5.

However, during the development of the distributed HIPPO system, I came across the concept of adaptive hypertext [Bru96]. This was a relatively new discipline, which moved away from a static view of hypertext. Instead, adaptive hypertext is viewed as a dynamic, flowing network which changes in reponse to the user. Adaptive hypertext systems recognise that a knowledge structure should be tailored to the individual user, and indeed, *should* change as the user gains more expertise. Adaptive systems use information about the user, and feedback from the

user to modify and adjust the hypertext environment. This was an idea which appealed to me, and I began to explore ways in which I could combine all three areas of research – open hypertext, distributed hypertext and adaptive hypertext – into one common system. This vision went on to shape the HIPPO research, and proved to be the defining direction for the ideas presented in this thesis.

A number of interesting ideas emerged from this work, but the most significant contributions can be summarised as:

- Adaptive Fuzzy Anchors
- Adaptive Linkbase Trees
- Adaptive Distributed Systems

These were used as the main structure for the thesis, and have each been allocated a chapter of discussion. The remainder of this section will briefly discuss each of these, and outline how these ideas emerged. I will also explain why I think these are original, important ideas, and are worthy of future work.

7.1.1 Adaptive Fuzzy Anchors

I began by looking at the most basic of hypertext objects – the hypertext anchor. This is often seen as a simple abstraction in the hypertext world, and receives correspondingly little attention. I soon realised that existing hypertext systems provided widely differing anchor implementations – each with slightly different interpretations [AMY88, HMT87, YHMD88]. However, while many other aspects of the hypertext model have been developed, the anchor remains largely unchanged from many of the early implementations. The anchor is seen solely as a way of identifying a link endpoint – a single point on a page, or a simple span of text. I feel that the anchor plays a much richer role in a hypertext – it is more than just a marker, it *encapsulates* the very concepts which form the basis of hypertext links. For this reason, I began to look at ways in which I could develop the anchor abstraction, by applying some of the open hypertext and adaptive techniques seen in other systems.

One of the defining characteristics of many open hypertext systems is the strong notion of a link, and the way this is maintained separately from other node contents. This is often seen in the form of a *linkbase* [DHHH92, YHMD88]. However, I noticed that the anchor did not enjoy the same treatment, and was often implemented as an afterthought – perhaps as an attribute of a link definition. Some open hypertext models such as PROXHY [Kac90] and D^2 [HGC94] did elevate the anchor to that of a first-class, primary object, and I used this idea in the HIPPO research. The anchor was an individual hypertext object which could be manipulated in the same way as any other hypertext abstractions such as nodes and links. I then looked at the

advantages that this offered (eg. sharing objects, reusing anchors etc), and readers are referred to Chapter 3 for more information.

Despite this first-class status given to anchors in my HIPPO system, I was still unhappy with the addressing model which I was using. Single anchor points [AMY88] were useful as user interface hints, and spans of text [YHMD88] were interesting anchor developments. However, I did not feel that this approach to addressing node contents was sufficient for complex knowledge structures. An anchor should truly encapsulate a concept or idea in a node, and it is rarely possible to do this using existing anchor models. A concept does not simply appear in a piece of text, but rather emerges and grows as the textual description unfolds.

Furthermore, it seemed natural to apply some adaptive modelling to an anchor definition, so that an anchor could change and move in response to user actions. Existing approaches to adaptive hypertext had been concerned mainly with the presentation of links – providing sets of links which were tailored to the user, or altering the user interface for a user. Adaptive hypertext had not been applied to the hypertext anchor. I wanted to develop a anchor model which could better encapsulate a concept in a node, as well as providing a natural platform for adaptation. These ideas led to the development of the *fuzzy anchor*.

Fuzzy anchors build on the work of Zadeh [Zad65] and others, by applying fuzzy sets to the hypertext anchor. An anchor is defined as a set of partial truth values – each representing the strength of an anchor at that point. In this way, a anchor does not simply have a start and an end, but has a more gradual definition – an anchor emerges, increasing in strength before reaching the true centre of the anchor. Chapter 3 describes this idea in more detail, and introduces the analogy of a *contour map* on a geographical map. Authors are no longer forced to select a start and end point for each anchor, but can use a fuzzy anchor to show how an idea gradually unfolds in a node,

These "fuzzy values" which are used to define fuzzy anchors, also provided a useful basis for an adaptive model. These values could be modified in response to user selections, and could be used to adapt an anchor over time. This seemed an attractive idea which would allow the user to play a role in defining anchors. Indeed, this was an important idea which went on to shape many of the other ideas in the thesis. I do feel strongly that the end users of any system should play an active role in its initial design. The same applies to a hypertext system, so that feedback from users should be incorporated into the definition of the hypertext. This feedback should influence *all* aspects of a hypertext system, not just the link definitions. Fuzzy anchors seemed to offer one way of achieving this.

Chapter 3 described these fuzzy anchors in more detail, and discussed the HIPPO prototype which was used to implement these ideas. A number of different presentational approaches were considered in the implementation, and this also raised a

number of other areas where the fuzzy anchor could be improved. I discuss some of the more interesting areas for future research later in this chapter (Section 7.2.1). In particular, I would like try a fuzzy-anchor system in a real test environment, to observe how users interact with fuzzy anchors. This was not something I had time to do in the current research, and I would like to conduct some evaluation studies in the future. Fuzzy anchors build on some of the work in open systems [Kac90, HGC94] and adaptive hypertext [Bru96], but use the ideas of fuzzy sets [Zad65] to present a novel approach to hypertext anchoring. This work on adaptive fuzzy anchors was presented in a paper at the HHPTM conference [New97b].

7.1.2 Adaptive Linkbase Trees

The *fuzzy anchor* combined the notion of first-class objects from open hypertext, with ideas from adaptive hypertext. I then began to explore how I could develop the hypertext link in a similar way. The hypertext link is fundamental to the hypertext model, and has been developed in many ways over the years – ranging from simple links and multi-way links, through to typed linking models and dynamic links [YHMD88, DS86, Eng84a, HMT87, Tri86]. Much of the research to emerge from the open hypertext community also identifies the link as a key abstraction, and often separates linking operations into a separate link services layer [Pea89, YHMD88]. This was an area that I wanted to develop in the HIPPO system, which led to the idea of *adaptive linkbase trees*.

One of the interesting ideas to emerge from the work on fuzzy anchors was the idea of reuse – sharing anchor objects between multiple link definitions. I had also become very interested in the work on Object-oriented design in software systems [Boo94]. Object-oriented research proposed software design methodologies based on reuse and the sharing of existing resources. Units of work are reused in other situations, extended and specialised for particular tasks, to reduce the maintenance efforts and the overhead of building software systems. I wanted to find ways of modelling this reuse in hypertext systems, so that links and relations could be shared between authors.

The current approach to hypertext authoring would see authors working in isolation, creating each link from scratch. Links are not shared between users or authors, and authors are largely unaware of any hypertext work which exists outside of their domain. I did not feel that this was an efficient way to develop hypertexts, and began to look at more reusable, *object-oriented* linking models.

Many systems such as Microcosm [DHHH92] separate linking information into separate units, known as *linkbases* or *contexts* etc. These are used largely as a practical measure to separate link definitions from the node contents themselves. This allows hypertext services to be integrated more easily with legacy applications, which would normally be "hypertext-unaware". However, I became more interested in the

linkbase, not simply as a collection of links, but as a primary object in a hypertext model. The linkbase could be seen as first-class object, which could be used to form the basis of a reusable, *object-oriented* model. This led to the development of the *linkbase tree*.

Linkbase trees combine linkbases into inheritance hierarchies, in the same way that software objects are used in the object-oriented world. Linkbases become a basic unit of work, and allow authors to build on existing link definitions. Linkbases can be augmented with additional link definitions further down the tree, or links can be overridden by new definitions. This encourages a constructive approach to hypertext authoring, in which new hypertexts are defined in terms of older, existing networks. Chapter 4 explains this idea in more detail, and discusses the advantages of linkbase inheritance hierarchies.

The linkbase tree hierarchy promotes the reuse of existing link definitions. Authors are encouraged to build on other authors' work – adding their own links to existing definitions. Furthermore, the author is forced to think about the relationships *between* linkbases in more detail. However, while this does offer a more reusable approach to hypertext design, it is still a largely static model. Linkbase hierarchies do not change over time, and this was an area where I felt adaptive hypertext could be of use. As before, I wanted to combine open hypertext ideas with an adaptive model, so that linkbase trees could respond to user actions. This was the next stage of development.

I felt that the idea of a fuzzy anchor had been quite successful, which had achieved some degree of adaptability by using confidence values. I began to look at ways in which I could used a similar idea in the linkbase hierarchy, which could then form the basis of an adaptive model. I was aware of some existing work which used weighted hypertext links [Fur86, PD90, PT90, Fri87]. Many of the existing adaptive implementations also used some form of rating or confidence value for tailoring links to a particular user. Weighted links seemed to be useful in the HIPPO model, by allowing the author to model the importance or confidence in a hypertext link. I was also looking to use these weighted links in a final adaptive model, so these weighted links were added to the HIPPO prototype as a starting point for adaptive linking model.

However, I was more interested in pursuing the idea of linkbase inheritance hierarchies, and looking at ways that these could be used in an adaptive model. I then considered associating weighted values, not only with each link, but also with each inheritance relationship. Each branch in an inheritance hierarchy is given some confidence value, which represents the importance of that particular inheritance relationship. This would allow authors to model, not only the importance of each link, but also the "strength of inheritance". The author can then decide on the relative importance of each linkbase in the tree, and how much they will contribute to

the overall hypertext.

More importantly, these weighted inheritance hierarchies could be used as the basis for an adaptive model. In this case, I chose a relatively simple approach which increased a weighted value, each time a link was traversed. Clearly, this is too simple for a usable system, and a more *intelligent* adaptive model should be considered in future work. However, it is the idea of using linkbases as reusable blocks, and structuring these with weighted inheritance hierarchies which is the important point. Chapter 4 describes the adaptive model in more detail, and suggests ways in which this could be improved.

Adaptive linkbase trees combine ideas from open hypertext systems, object-oriented design and adaptive modelling. The linkbase is a common abstraction in many open hypertext systems [DHHH92, GB80], and has been combined with the inheritance model seen in object-oriented research [Boo94]. This linkbase inheritance model uses some of the weighted value approaches used in adaptive systems and other research on weighted-links [Fur86, PD90, PT90, Fri87]. I would like to develop the adaptive model further in any future work, and some other future directions are discussed in section 7.2.2. In particular, I would like to explore the impact of linkbase inheritance trees in real-world situations, to see how this idea can promote reuse of links between authors. This was not something I had time to do in this research, but even if linkbase inheritance hierarchies are not the best way to achieve this, I believe that the reuse of hypertext objects between authors is an important one, and should be considered in future research.

7.1.3 Adaptive Distributed Systems

The final chapters in the thesis explore the idea of a distributed HIPPO model. As mentioned previously, one of the initial motivations behind the HIPPO research was to combine existing open hypertext work with distributed techniques. The early work on a distributed hypertext model changed direction, as I became more interested in applying adaptive modelling to open hypertext ideas. I had implemented the HIPPO prototype which supported fuzzy anchors and linkbase trees. However, I decided to return to the original idea of a distributed hypertext system, to see how I could incorporate distributed ideas into my current prototype.

The HIPPO application had used a number of different technologies – C++, X11/Motif, Acrobat etc. The adaptive servers which were used to support the adaptive modelling of anchors and linkbase trees also used some Remote Procedure Calls to distribute the servers. However, the prototype was largely implemented as a monolithic application which operated on a single platform. Despite offering some interesting ideas, it did not sit easily with my vision of an open, distributed system. Wiil et al [WL96] also identified the importance of a distributed architecture in open hypertext systems. The remainder of my research looked at ways in which I could

re-implement the HIPPO application using a more distributed model.

My initial idea had been to view a hypertext system as a collection of lightweight components which offered specific hypertext services. I was very interested in exploiting the advantages of distributed architectures, to provide a more robust and scalable system. I had been influenced by the early work of Nelson [Nel95], Engelbart [Eng84a] and others, who developed large-scale, collaborative hypertext systems. Hypertext was seen as more of an *environment* or a way of working, rather than a particular application. The PROXHY [Kac90] and D^2 [HGC94] systes also used distributed ideas to good effect. This was something that I wanted to explore in my HIPPO research, and I felt that distributed ideas offered one way to do this.

I decided to re-implement some of the ideas in my initial HIPPO prototype, using a distributed approach. The CORBA model [Objc] has been a very influential distributed architecture, which provides a well defined layered model for developing distributed systems. At the time I started my initial distributed implementation, CORBA research was still in its infancy, and there were very few available implementations. For these reasons, I decided to use a relatively low-level and simple distributed mechanism – the Remote Procedure Call [Sun95a] – to distribute hypertext services. However, some aspects of the CORBA model such as interface trading, directory services and other CORBA services [Objb] did influence later work on HIPPO+.

The original HIPPO prototype was re-implemented using distributed components, and became the HIPPO+ system. I wanted to build on the work of PROXHY [Kac90] and D^2 [HGC94] and other open hypertext systems, to show how existing distributed techniques could be used in hypertext systems. Furthermore, I wanted to explore the effect of widely distributed hypertext systems on the user – remote access to particular hypertext operations, tailorable hypertext systems, blurring the boundary between user and developer. These were all ideas that shaped the development of the distributed HIPPO+ system.

My initial approach to HIPPO+ was to identify key operations which I considered fundamental to the hypertext paradigm – view node, retrieve links, follow links etc. I also included some of the newer ideas which I had implemented in the original HIPPO prototype (fuzzy anchors, linkbase trees etc). This gave me a usable distributed hypertext system. The main area where my system differs from other existing hypertext systems was the *level* of distribution – distribution is applied to *all* aspects of the model. Not only is node data and link data distributed throughout a network, but also the actual components themselves. In this way, there is no such thing as "the application"; instead, the system becomes a collection of components. This fine-grained approach was an area which I wanted to develop further.

In particular, I was interested in the idea of involving the user in the definition and construction of the hypertext system. Current approaches to hypertext design

are very strict and "one-way". The developer defines the functionality of the hypertext application, and the user is seen largely as a passive participant. However, the fundamental idea of adaptive hypertext is that a hypertext should be tailored and customised to meet the needs of the user. This HIPPO research attempts to combine ideas from adaptive hypertext with existing work on open hypertext and distributed systems. As a result, I wanted to look at ways in which I could apply some adaptive modelling to the existing distributed HIPPO+ system. This was to form the remainder of my research into HIPPO.

The first step towards an "adaptive distributed HIPPO model" was to look at the mappings between abstract hypertext operations (*view node, follow link* etc) and the actual remote implementations. It occurred to me that these mappings could be use to provide some degree of customisation. A user could decide to use alternative implementations of a particular hypertext function, and was free to select the appropriate service from somewhere in the network. This was inspired by the use of interfaces and object references used in the CORBA and OMA models [Obj97]. This was added to the HIPPO+ prototype, so that users could choose which implementation was used for each hypertext operation. I also allowed users to add their own hypertext operations, in addition to the pre-defined set of functions. This would allow HIPPO to be used in situations which were not originally envisaged by myself.

While this approach did offer some additional flexibility, it was still a long way from the vision of a truly adaptive hypertext system. Most approaches to adaptive hypertext have focussed on modifying link definitions or some customisation of the graphical interfaces. However, adaptive hypertext systems do not apply this level of adaptivity to the actual *functionality* of the hypertext system. This was my ultimate goal, to provide an adaptive system which actually changed the *meaning* of hypertext operations. For example, it seems quite natural that the act of *viewing a node* should mean different things depending on the contents of the node under examination. Similarly, *follow link* can, and *should* have many different meanings depending on the context.

Chapter 6 details the final phase of my research which was to propose an adaptive model for the HIPPO+ system. The chapter introduces the idea of *document objects* which use the node as the unit of adaptivity, so that the actual implementations of hypertext operations changes according to the current node. Each of these hypertext operations is assigned a confidence value, which is modified each time a user invokes the particular service. In this way, the mappings between abstract hypertext operations and actual implementations will change – not simply from node to node, but will also adapt and be modified over time.

This work on an adaptive HIPPO+ attempts to combine adaptive techniques with the existing open, distributed hypertext system. However, this differs from

existing work by applying adaptive ideas to a *distributed* architecture. Also, the adaptivity applies, not just to the hypertext network, but to the actual hypertext system *itself*. The meaning of operations will change and adapt, to select the most appropriate service for a particular node. I consider this a real departure from existing adaptive work, and I would like to see how this approach could be applied to a more general audience, in the adaptivity of actual software systems. Some of this work was presented at the HTF Workshop held during the Hypertext 1997 conference [New97a].

I did not have time to implement the adaptive model in the intial version of HIPPO+ which is described here, and it is included only as a theoretical proposal. However, I feel that this model could be implemented, and this is an area for future work. This is an area of my research which I would like to develop further, and some possible areas are discussed later in the chapter. The adaptive algorithms which are used to modify the weightings for each hypertext service, are an obvious candidate for more work. I would also like to see how my adaptive model would work in a real environment, with real users and some more rigorous analysis. In particular, I am interested to see how a user would react to a constantly changing hypertext system. I am concerned that this level of change could prove disorientating for users, and that a user would like some way of *controlling* the adaptivity. Users often need some features which remain constant and act as reference points. Similarly, the user of an adaptive system may require a way of "fixing" the system, to prevent any changes taking place. Indeed, this could be a problem for the entire adaptive hypertext community, and is an area I would explore in the future.

7.2 Future Research

The previous discussion has provided an overview of the HIPPO research, and has shown how each of the ideas were developed. The key contributions and original research which has emerged from the work on HIPPO have also been outlined. This final section explores some of the areas which have caused some problems and are considered useful directions for future research.

7.2.1 Developing Fuzzy Anchor Model

Alternative Metaphors For Fuzzy Anchors

The prototype discussed in this chapter adopted a very visual representation for fuzzy anchors, using colour to indicate fuzzy membership values. Each anchor is maintained internally as a matrix grid which represents the physical display, such that each cell has a direct mapping to an area of the display window. This approach has proved very successful, and the visual, *point-and-click* metaphor has been very

intuitive. However, the use of colour to indicate fuzzy values does have some limitations. In particular, the limitation of colours which were available caused some problems, and the prototype application encountered problems when displaying larger ranges of values. Some improvements were made by mapping large ranges of values on to a smaller range of colours, or by using *private colourmaps* in the X11 windowing system. While the use of colour has proved very successful, this does raise the possibility of using alternative metaphors to represent fuzzy anchors.

One alternative might be to use the interaction between the mouse and cursor to indicate the presence of fuzzy anchors. This could emphasise the analogy with *contour maps* and landscapes, so that the cursor may move more slowly as the user *"climbs"* an anchor, and accelerates as they descend the other side. This physical view of fuzzy anchors as a form of terrain or landscape seems to hold much promise and could be developed further. One avenue of research might be to introduce some virtual reality and 3-dimensional modelling techniques. The series of fuzzy anchors could be represented as a series of hills, and the document could be overlayed on top of these anchors. The document of a surface becomes more analagous to a rough landscape, and the user could explore *valleys* and *peaks* using some of the existing techniques used in virtual reality modelling.

Managing Complexity

A document space of any non-trivial size has many complex relationships and dependencies, and a single node could (and *should*) have many links originating and converging on the node. Each phrase can act as an anchor for many links, and many anchors may overlap with each other. A particular concept described by a node could mean many things to many users, and could provide the basis for many competing regions. This shares many of the ideas put forward by Nelson in the Xanadu system [Nel93], in which many hundreds and thousands of documents are interconnected. It is the role of a hypertext application to embrace this richness, and provide tools for managing the complexity, rather than restricting the user to simple hypertext structures.

The prototype discussed here has adopted a *resource-based* approach to hypertext anchoring, similar to the linking mechanisms used in the Microcosm system [FHHD90]. Users identify a concept in the text, then query the application to see which fuzzy anchors have some presence at that position. The idea of fuzzy anchors encourages a larger, more expressive anchor which includes more content than a simple span of text. When a user selects a region, they are more likely to locate some anchors; this is just as it should be, because every phrase and piece of content in the node has some part to play and should contribute to other links and anchors.

However, these larger, more expressive fuzzy anchors can raise some complexity

issues. Each time a user selects a region of a node, the application may return a long list of possible anchors which appear at that location. The prototype provides some simple thresholding operations so that the application only acknowledges anchors which have fuzzy values above a certain limit. This could be seen as viewing only the *highest peaks*, and ignoring the smaller, less important hills. Also, the chapter emphasises the idea of *anchorbases* in which related anchors are grouped together into a single file. This allows the author to provide several interpretations and representations of each anchor to suit different users, and the user can select the appropriate anchorbase which is most suitable.

However, if the prototype is developed further, it would be beneficial to consider alternative methods for managing the number and complexity of fuzzy anchors. This could borrow from much of the existing work in the hypertext community for managing hypertext links, which help address the infamous *disorientation* problem [Nie90]. Perhaps an application which supports fuzzy anchoring could provide more advanced filtering operations which reflect the more complex nature of fuzzy anchors. A user could select anchors based on the *gradient* of the anchor, or perhaps the distribution of values etc. These are all areas which should be addressed in future implementations of the prototype.

Logical Fuzzy Anchors

The discussion of the prototype implementation explained the presentational, graphical approach to anchors. Each fuzzy anchor was viewed in terms of graphical marks which were *painted* over the document image. This was a useful approach which greatly simplified the implementation of the prototype and associated tools. However, the idea of a more logical, descriptive approach to anchors was suggested for defining fuzzy anchors. Similarly, the idea of using generic, logical documents as the underlying node content was considered, which would define the logical elements of each document (title, heading etc). This contrasts with the presentational, physical view of documents which is supported by the PDF document standard [BC93].

This logical representation of documents raises many interesting possibilities for fuzzy anchoring. Anchors could be defined using a series of tags to delimit elements with the same membership value. Perhaps an anchor could be defined in terms of the elements it encapsulates, along with a *distribution* function which defines the *gradient* of the anchor. For example, the author could define a fuzzy anchor which includes an entire paragraph, followed by an image, and the opening subsection of another paragraph. The author could describe how the fuzzy anchor will change over the region, and explain the gradient and type of contours. Some situations require very specific anchors which identify the central concept, then immediately fade away. In contrast, other scenarios might benefit from a more ambiguous an-

chor which fades away gently from the central concept, to include many of the surrounding elements.

Indeed, a more descriptive approach to fuzzy anchors could be developed in a number of useful ways. An application could support a number of different patterns and shapes which indicate different anchoring semantics. Perhaps an author could refer to fuzzy anchors by the intended meaning rather than the graphical pattern eg. *general concept, definition, emphasis* etc. In this way, the anchor becomes a truly logical abstraction which is defined by the semantics of the anchor, and is not tied to any particular appearance or display. Furthermore, an application could use these anchor semantics to infer anchors and types of anchors from logical documents. For example, an application would automatically know the shape and type of fuzzy anchor which should be used for a quote, a heading, a mathematical formula etc. This would provide more accurate methods for defining anchors, and could reduce the time and effort involved in authoring hypertext applications.

Logical descriptions also simplify the problem of maintaining a hypertext. Many researchers have identified problems which arise when authors edit the underlying node content, and how a hypertext application must ensure that anchors and links are updated and remain consistent. This can be complicated further in more open systems, when hypertext information is maintained separately from the node contents. Researchers have suggested alternative solutions to this consistency problem – heuristics in HyperTED, generic links in Microcosm [FHHD90] etc. Also, HyTime [GNKN97, CBDW94, Buf96] and Augment [Eng84a] both show the advantages of anchoring methods which are based around descriptive methods (eg. paragraph 3, chapter 2). A descriptive approach to fuzzy anchors which defines the elements to be included would provide a more maintainable anchoring system, which can better respond to any edits and changes which are made to the node contents.

The development of a more logical approach to fuzzy anchors shows much promise, and is an area that should be continued in future work. Logical anchor descriptions open up new possibilities for tools and utilities, and provide a more maintainable environment. An approach to fuzzy anchors which views the anchor as a more logical abstraction helps to express the true semantics of the anchor. An anchor is defined, no longer as a graphical object, but in terms of the concept being anchored. It is hoped that the semantics of logical anchors can be developed to the point where the author can define anchors simply by the intended meaning – a definition, an introductory concept, a repetition – and the application generates the appropriate anchor definition. This is clearly a long way from the current implementation of fuzzy anchors, but offers a fruitful area for future research.

Efficient Storage

The current implementation of fuzzy anchors uses a hidden matrix to divide the visible page into cells, which are then used to define fuzzy anchors. Each cell in the anchor grid corresponds to a physical region of the page, and contains a numeric value to indicate the fuzzy value at that region. This approach is very simple, but has proved to be very effective in the current model, and has simplified the implementation of the prototype. However, one of the main problems which arises from this approach is the large amount of storage space required for each anchor definition. Each anchor demands an entire grid to represent the page, and although the application allows the author to specify the resolution of this grid (*xres,yres*), a typical anchor still incurs an unacceptable overhead. The previous subsection which discussed a logical description for anchors offers one possibility for reducing the space requirements, and an earlier discussion suggested using mathematical descriptions to define splines etc. Other alternatives might be to incorporate some form of compression to reduce the total storage size. The majority of the anchor grid remains empty, and this could almost certainly be compressed, and other approaches such as *run-length encoding* might be useful. Future development of the implementation must address these storage problems if fuzzy anchors are to become more widely used

Authoring Tools

The prototype application supports a number of tools to help the user define and manipulate fuzzy anchors. A number of *paintbrush*-style tools allow the author to create new fuzzy anchors, and these appear to be quite effective. A number of anchor *patterns* are provided which allow authors to reuse common shapes, along with some simple utilities for manipulating anchors (thresholding functions, editing parameters etc). However, there is considerable scope for more development in this area, and a more usable environment should provide additional utilities and tools. For example, the implementation is currently lacking any means of managing sets of anchors (*anchorbases*), and would also benefit from better tools to edit and change existing anchors. Future applications should provide tools for manipulating anchor values, perhaps *smoothing* a range of fuzzy values to remove high frequency variations which may distract from the anchor definition. Users may wish to sharpen and accentuate any variations, as these may suggest concepts which may be better expressed as separate anchors. Also, the author may wish to alter the resolution of the underlying matrix used to define the fuzzy anchors, to increase or decrease the level of detail, and will require tools to control how existing values are mapped to the new matrix.

Alternative Adaptive Models

One of the most important contributions to emerge from the work on fuzzy anchors has been the idea of more adaptive anchors which reflect the needs of the users. The prototype implementation adopts a very simple adaptive model which uses a centralised server to receive feedback from the user clients, which are then used to adapt the anchor definitions. The current algorithm monitors the region that is selected by a user, then increments the corresponding value in the matching anchor definition. Conversely, the remaining values in the anchor are also decreased. This simple approach is very effective and allows anchors to grow and evolve, in response to the users' browsing patterns. However, while this implementation is sufficient to demonstrate the idea of adaptive anchoring, it does raise the possibility of more complex, *intelligent* adaptive techniques.

For example, it could be argued that the current form of adaptive anchors is very volatile, and that anchors are too easily influenced by a single user. These effects should be reduced by the large numbers of users that are expected to use the system, so that any anomalies are avoided. However, a simple improvement would be to reduce the increments, and perhaps reduce the rate of decay which is used to decrease surrounding cells. Indeed, it would be useful to experiment with different rates of change, to see which growth rate produces the most accurate and stable fuzzy anchor. More complex techniques could be used to alter the distribution of the fuzzy values, perhaps altering the most important values, but leaving the lower membership values untouched, assuming that these have little impact on the anchor specification. Alternatively, the author may wish to specify some areas of a fuzzy anchor as immutable, so that other fuzzy values may be increased, while some areas cannot be changed. Chapter 2 outlined a number of different approaches which have been used in adaptive modelling (user knowledge, goal-driven models etc), and these could all be incorporated into future applications.

The possibility of including some form of neural network model has been considered, so that the application can *learn* from the user feedback, rather than blindly applying changes to anchor definitions. It would also be useful to gain some more reliable experimental data, using a larger sample space of users. It would be beneficial to observe users, to see how they interpret fuzzy anchors, and how different types of users adopt different browsing patterns. Some anchor patterns may prove more useful than others, and it may transpire that users do not require the flexibility of fuzzy anchoring in certain situations. Future implementations almost certainly require some form of group management, so that users can be separated according to therir experience, background, goals etc. It is too simplistic to allow the browsing patterns of every user to influence the anchor definitions for every other user. These are all areas which need to be considered if the adaptive modelling of fuzzy anchors

is to be improved.

7.2.2 Developing Linkbase Trees

While it is hoped that the idea of inheritance hierarchies for linkbase management can offer a more flexible and powerful view of hypertext linking, a number of issues need to be explored further. The use of OO-techniques such as inheritance promotes a form of link reuse and encourages authors to build on other existing hypertexts and link definitions. However, the true effect of inheritance has not been explored in proper use. It is important to understand how the inheritance hierarchies affect the way in which linkbases are used. Should hierarchies be complex, deep trees or should more shallow, simpler trees be used? Are inheritance trees too dependent on the initial parent linkbases? How do changes higher up the hierarchy affect the linkbases lower down the tree? What are the effects of poor quality and inaccurate linkbases on the derived linkbases further down the tree? These are all aspects which have a direct analogy in object-oriented design methodologies, and many of the heuristics and guidelines which have been developed in O-O design could be usefully employed here.

The thesis showed how these weighted linkbase trees could be combined with the work on fuzzy anchors, to produce an overall weighting for each user selection. In this case, the fuzzy value of the anchor was multiplied with the weighting of the appropriate link, which was finally multiplied with the inheritance weighting of the linkbase. The idea of combining each of these new hypertext abstractions into a single, coherent weighting seems to be useful. However, it is not clear that a simple multiplication is the most appropriate way to achieve this. Future work could look at the relative importance of each of the weighted objects – fuzzy anchors, weighted links and weighted trees – and suggest alternative ways of combining these.

There are also a number of implementation issues which must be addressed in a practical system. The environment must check for cycles in the inheritance hierarchy when multiple inheritance is used, to avoid any circular dependencies. The environment must also decide on policies for detecting and resolving conflicts – for example, what happens if the same link is defined in more than one linkbase? The idea of weighted inheritance relationships is a novel approach which holds much promise, and future development could explore many different avenues of adaptive strategies.

This research has showed the benefits of applying the techniques and methodologies developed in Object-Oriented research to the field of hypertext. Linkbases can be reused and shared, producing higher quality hypertexts which can be developed faster and maintained more easily. This has attempted to show the value of viewing links at a more abstract level, using collections of links as the unit of granularity in a hypertext. This suggests that linkbases could now be viewed in the

same light as single, isolated links once were, and perhaps much of the research into hypertext links could now be usefully applied to hypertext linkbases. Perhaps linkbases could benefit from a form of typing, just as the hypertext link was developed to include this aspect? Does it make sense to have directed linkbases? What is an *n*-ary linkbase?

Indeed, the similarities with Object Oriented design and implementation could be usefully exploited further. OO research has developed many techniques and formal theory which could be incorporated into the hypertext community. The recent work on *design patterns* [Ale77, GHJV94] which attempts to identify recurrent design strategies and reusable components seems an obvious area which could be of interest. Perhaps hypertext researchers could identify fundamental hypertext components, link types or patterns of use which can be generalised and widely reused in other hypertext contexts. Indeed, there is already some interesting work in the application of patterns in the hypertext community which looks set to play an important role in hypertext design [GMP96, RSG97]. This raises some other interesting questions – what constitutes a well-designed linkbase? How can an author create a reusable linkbase? What dictates reusability? When should an existing linkbase be reused, and when should a new linkbase be created from scratch?

Tree Brokers

While the reuse of linkbases to construct other link sets can be very flexible, the inheritance model can also raise a number of complexity issues. In particular, a system needs to address the problem of building the inheritance trees and how to manage the many interdependencies between linkbases. For example, in the engineering scenario which was discussed in Chapter 4, who should decide that a specialised linkbase covering *bridge construction* should be derived from linkbases from the *mathematics* and *materials disciplines*? An obvious approach is to delegate these tasks to the author of the new linkbase – when an author constructs a linkbase, they are also expected to construct the appropriate inheritance hierarchy for the new linkbase. However, while these inheritance relationships are an integral part of the linkbase authoring process, it is sometimes desirable to separate the construction of the inheritance hierarchy from the actual definition of new links. For example, one could imagine a number of different inheritance trees which could be overlaid on to the linkbases, which all offer different solutions to the same problem. Users could then select the optimal inheritance tree to use for their hypertext.

An interesting development would be to allow users to consult several expert sources or *brokers*, which each suggest the best inheritance relationships to use for a particular subject area. The particular hierarchy which is returned by the broker could be based on any number of criteria – availability of linkbase components, network topology, linkbase popularity, user profiles etc. Users could then select

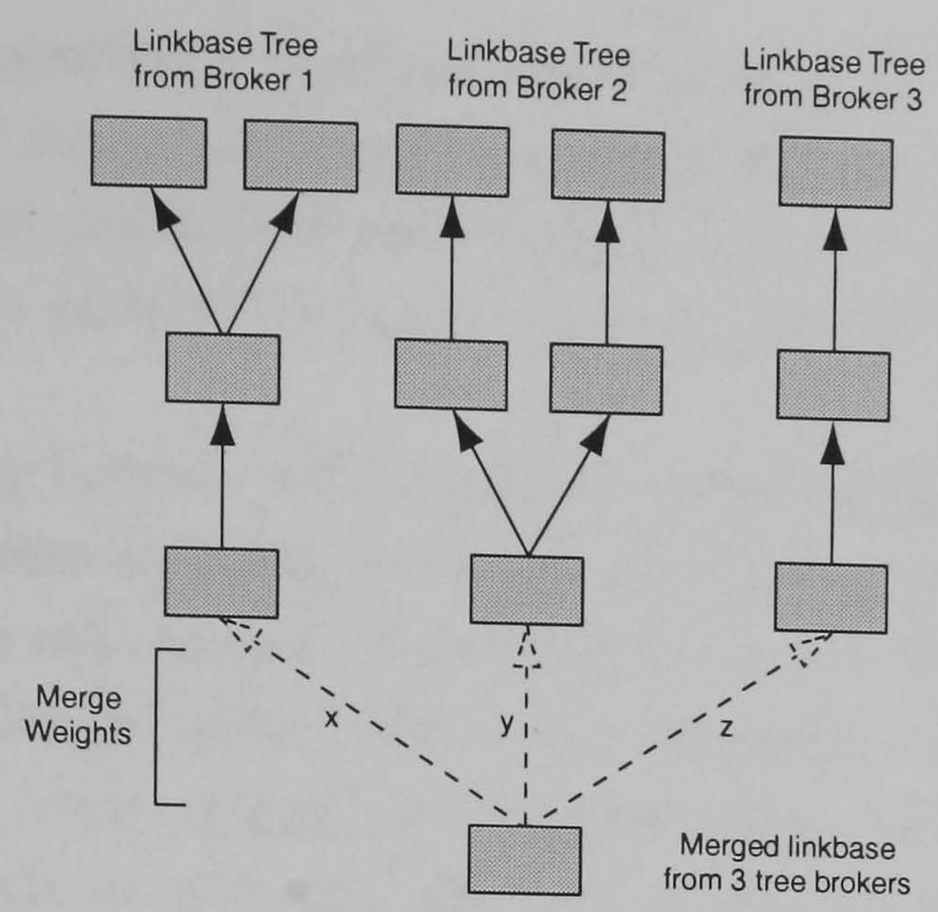


Figure 7.1: Merging multiple inheritance trees

one tree, or more interestingly, they could combine several of the trees together to form a more rounded, inclusive tree which incorporates several hiearchies. For example, the user might select three inheritance trees which each produce a linkbase – the user could then merge these linkbases together to form a new, more complete linkbase. Furthermore, they could apply some form of weighting factors to each tree, as discussed earlier, so that they can express a preference or attach greater importance to a particular tree (figure 7.2.2).

7.2.3 Developing HIPPO+

Improve Component Model

The work on the distributed HIPPO+ explored the idea of a widely distributed hypertext system, based on a collection of remote hypertext services. This developed the idea of a *"component-based approach"* to hypertext design, in which developers make components available to clients, who can then incorporate them into their hypertext environment. This component-based approach is beginning to emerge from some of the work which combines object-oriented ideas into the hypertext field [Gro94a, SR95, OHSa]. The Microcosm-TNG model uses a distributed model to provide an open hypertext framework [GDHR97], and the idea of *link resolving components* was used to good effect by Tompa *et al* [TBR93].

The current implementation of HIPPO+ uses a very simple communications model based on low-level Remote Procedure Calls. Hypertext services are implemented as RPC services, and data are passed around as arbitrary text-streams. This model is useful to demonstrate some of the advantages of distributed hypertext systems, and provided a basis for developing the adaptive model in chapter 6. The RPC model was chosen because these libraries were widely available, and offered

a simple means of supporting a distributed environment. Early implementations of the CORBA standard introduced significant overheads, and had limited availability. The CORBA model and related technologies were still very immature when the initial implementation of HIPPO+ began, so these component models were not a practical option.

The loose coupling between RPC services is an advantage of many distributed systems, but this is often achieved at the cost of a less integrated environment. In particular, the simple *call-response* communication in the RPC model is not satisfactory for more flexible interaction between components. All data in the HIPPO+ system is transferred using opaque text streams which, while suitable for simple hypertext services such as retrieving links and nodes, is unable to transfer richer data contents such as objects or structured messages. The current implementation of service interfaces is also too simple for a scalable system, and the dynamic invocation model using the HIPPO+ registry service, while useful as a general concept, is unsuitable for a large-scale software system.

Any further research on the HIPPO+ system would benefit from a more formal implementation of a component model. HIPPO+ should move away from the procedural, function-based environment which uses RPC services as the basic unit of distribution. Future HIPPO+ implementations should move towards an object-based implementation. This would lend itself to one of the emerging distributed messaging architectures which were discussed in chapter 5, such as CORBA, Java RMI, JavaBeans etc. Indeed, many of the current Object-Oriented languages and technologies such as Java would provide many benefits for future HIPPO+ implementations. Java could be used to support dynamic class loading, to allow the users client to integrate new hypertext services on demand. The platform-independence provided by Java objects would allow users to execute services locally, when remote invocations are less appropriate. Also, much of the work in the Java world could be incorporated into HIPPO+ – object directory services [SM98d], messaging interfaces [SM98a], object serialisation [SM96], transaction services [SM98b] etc.

Distributed approaches to software design are playing an increasingly important role in the open hypertext community (eg. SP3 [LS94], Microcosm-TNG [GDHR97]), and this could be used to build on the current HIPPO model. The idea of distributed hypertext toolkits has much in common with the HIPPO model of distributed services. Similarly, the idea of *link services badges* proposed by Arents *et al* could be a useful starting point for extending the Hypertext Component Hierarchy used in the HIPPO+ prototype. There is also some emerging work on the use of component models to implement open hypertext frameworks and support communication between hypertext applications [OHSa], which could tie in with this HIPPO research.

Using Compound Document Models in HIPPO+

Chapter 6 described a proposed adaptive model which could be used to provide a more dynamic and adaptive distributed hypertext system. The document or nodes were used as the basic unit of adaptation, and document objects were introduced to map abstract hypertext services on to specific remote implementations. Each of these mappings used weighted values to control the relevance and importance of each services, and these were then modified in response to the users actions.

The adaptive model described in the chapter has only been developed on a theoretical level, and has not been implemented in the HIPPO+ system described here. One of the priorities of any future development on HIPPO+ would be to implement this adaptive model, and evaluate how effective it is in supporting an adaptive distributed system. Chapter 5 introduced the idea of compound document models such as OpenDoc which have been developed to support component based systems. These compound documents could be used in HIPPO+ to manage collections of hypertext components. In particular, a compound document could be used to implement the adaptive model developed in Chapter 6. Each document or node in the hypertext could be modelled as a compound document which contains, not only the data associated with the node, but also the appropriate hypertext service components to implement a suitable hypertext model. Each document would use a different compound document, which would in turn contain the hypertext services and components which are most appropriate to the node. Services could then be built on top of the existing storage layers, to support the weighted fuzzy relationships between components, and for adapting these values in response to user actions. An adaptive HIPPO+ model based on compound documents may resemble that shown in figure 7.2.

Infrastructure Issues

Distributed systems offer a number of advantages over conventional localised software models – more robust architectures, exploitation of remote resources, shared resources etc. However, one of the most significant drawbacks of a distributed model relates to the performance. The HIPPO+ uses remote services to implement hypertext services, which can involve a significant amount of network traffic. In the current implementation of HIPPO+ even the simplest of hypertext services must invoke a remote implementation. The user's client requests the interface that should be used to invoke the service, and the service must return a text stream which describes this. The client then sends the required parameters over the network to the service, the service then processes these instructions, before the results are finally returned over the network.

When the HIPPO+ prototype is running on a small-scale, local-area network,

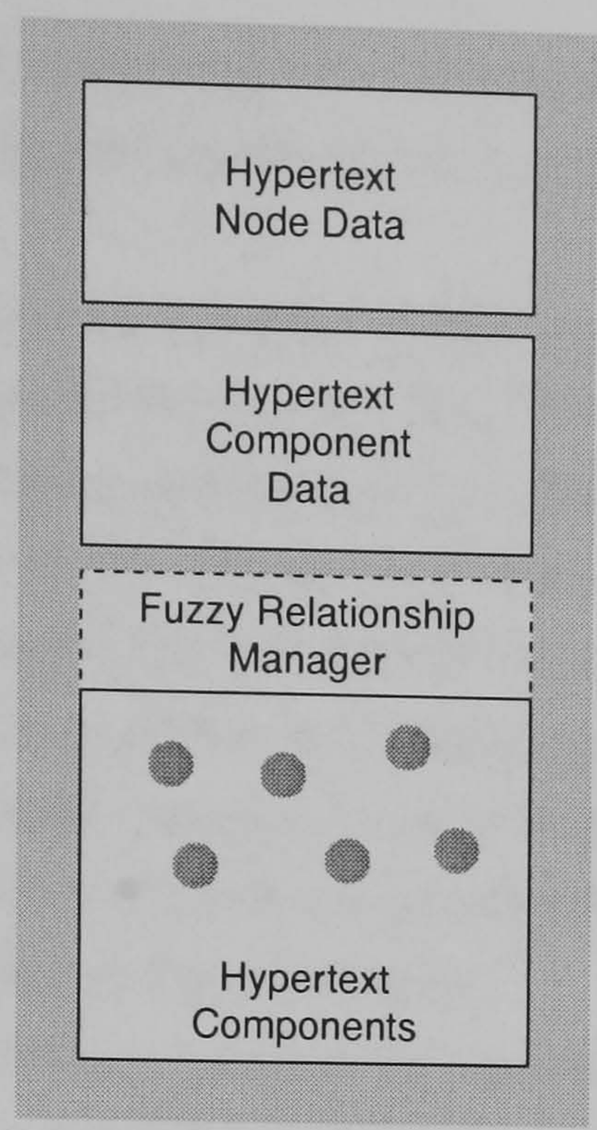


Figure 7.2: An adaptive HIPPO+ implementation based on compound documents

this overhead is not significant. However, as the number of the components grow and the load on the network increases, then this performance can cause problems. This is a problem which must be addressed by all distributed systems, and should be explored in any further research on HIPPO.

One approach might be to develop the transport mechanisms which are used by HIPPO to transfer the low-level data between components and client. For example, components could use some form of compression to reduce the size of any data packets. Future implementations may consider alternative inter-process communication and messaging systems such as those described in chapter 5, when larger quantities of data need to be transferred between components. This also raises the larger problem of messaging and data transport. The current approach based on RPC is useful for invoking remote services, but is less suited when large data streams – perhaps node contents or link definitions – need to be transferred. HIPPO+ ignores the issue of guaranteed-delivery – what happens if there is a network failure, or if the receiver becomes temporarily unavailable? Should the sender resend lost packets? Should the receiver provide acknowledgements?

HIPPO+ should also provide some form of fault tolerance, so that the hypertext system can continue to operate, should some components become unavailable. A distributed system already offers some degree of fault tolerance, by distributing components throughout the network, so that there is no single point of failure. If a component fails, then the user can continue to use other components, and can look elsewhere for the service. However, future HIPPO+ systems should provide some additional fault-tolerant support – starting new services when existing services fail,

mirroring components, load balancing etc. These are all areas which have been developed in other messaging and middleware systems, and could also be applied to the HIPPO+ domain.

The HIPPO+ client could also optimise performance in a number of other areas. For example, the client could cache results of remote requests, and return these local copies instead of sending requests over the network. Similarly, remote components could look at the ID of any clients, and optimise processing based on this. The previous section discussed some of the advantages of implementing HIPPO+ using a more mobile, platform-independent language such as Java. This would allow HIPPO+ clients to download commonly used remote components, and execute them locally. This could significantly reduce network traffic for common operations, at the cost of the initial transfer of the component.

One final area which has been ignored in the current design of HIPPO+ and should be addressed in future versions, is that of security. The current implementation does not implement any security control – any service can be invoked by any client and data streams are transferred unencrypted. Any large-scale implementation of HIPPO+ which is to be used by larger numbers of users needs to address these security issues. A robust implementation must provide authentication to prevent unauthorised service invocation. The ONC RPC libraries which are used in the current implementation provide some support for authentication and encryption using DES, Kerberos etc, and these may be a useful starting point. Security is a complex area which is beyond the scope of this thesis, and has not been implemented in the HIPPO research. However, security issues become more important in any distributed system, which introduces large numbers of users under different jurisdictions and controlling authorities. Furthermore, the computational aspects of the HIPPO+ model raise the possibilities of malicious or accidental damage, which makes suitable security policies even more important. This is an area which should be explored in any future development of HIPPO and the HIPPO+ system.

Management of Hypertext Services

The HIPPO+ model implements a hypertext system as a collection of remote components which provide common hypertext services. This approach to hypertext encourages the development of many lightweight components – hundreds and thousands of services throughout a network. Only a small subset of these will be used by the user at any one time, but the user is always free to use new components at any one time. This can raise the problem of managing large numbers of components – how to arrange hypertext services into some meaningful structure, and to prevent the user becoming overwhelmed by the sheer volume of services. Section 5.7 in chapter 5 discussed some of the approaches that have been used to address this in the HIPPO+ system – most notably the *Query Interface, Trading Model* and *Hypertext*

Component Hierarchy.

While these approaches to managing remote hypertext components have achieved some degree of success, they do not fully address the problem. The query interface is useful for inspecting particular components, but is not feasible for large numbers of services. The trading model which has been implemented is still only very simple, and does not manage large numbers of components well. The *Hypertext Component Hierarchy* is an interesting attempt to develop a classification system for hypertext services, but represents only a very simple taxonomy.

Future versions of HIPPO+ need to look at this problem of managing services and resource discovery in more detail. The trading model needs to be extended to allow more flexible registration and query support. The trader should support a flexible way of querying the trader contents, so that services which match some criteria can be retrieved. The HCH classification system also needs to be developed further, to see how it performs in real-world situations. The Hypertext Component Hierarchy can build on existing work on directory services such as JNDI [SM98d], JavaSpace [SM98c], NIS [NIS], LDAP [YHK93] etc.

Other approaches to resource discovery such as Harvest, WAIS etc may also prove useful for managing HIPPO hypertext services. One possible direction for future development is to use the hypertext paradigm *itself* for managing hypertext components. Remote hypertext services could be arranged into a hypertext network, and the hyperlinks used to express relationships between components eg. which components should be used together, with which documents etc. In this way, the entire body of hypertext research could be used to provide an effective means of managing hypertext services. Indeed, it may prove possible to use some elements of the actual HIPPO+ system to manage HIPPO components. This would allow HIPPO+ to manage itself – using fuzzy anchors, linkbase trees, adaptive modelling etc. This offers a nice form of closure, and could provide some interesting opportunities.

7.2.4 User Evaluation

The work in this thesis has presented a number of new ideas which combine some of the work in open and distributed systems with that on adaptive hypertext models. The author has shown how these concepts have emerged, and discussed some of the advantages that these offer to hypertext authors and users. The discussion of the HIPPO and HIPPO+ models has included a justification for each of these concepts, and for the methods that have been used for implementation. However, any discussion of these ideas should include a more formal evaluation of the hypertext model. The thesis has including some compelling reasons for introducing new concepts such as fuzzy anchors, linkbase trees or distributed hypertext services – however, the real advantages of these can only truly be explored after further user

evaluation.

The constraints of time and resources have meant that any evaluation of the HIPPO and HIPPO+ models has been necessarily limited and informal. The author has attempted to gain feedback from other colleagues and users and used this to develop the ideas in this thesis. While this has been useful, any future work should include a more exhaustive and formal evaluation using larger numbers of users in a controlled environment. An evaluation should get feedback from real users, either using evaluation surveys, observational methods or more automated evaluation methods. This raises all manner of issues concerning the evaluation of a hypertext system – how do you measure a *good* hypertext system? how do you extract useful feedback from the user without influencing their responses? do new hypertext concepts require new methods of evaluation?

Any evaluation of the HIPPO and HIPPO+ models should explore the real value of these new hypertext concepts to the user, and how these impact the way in which the user works. Fuzzy anchors, linkbase trees etc may all have a useful research contribution, but their value should also be measured in terms of how useful they are to the end user. The hypertext model has to be implemented in a way which is meaningful to the user, so that they can exploit the advantages that fuzzy anchors, link trees etc purport to offer.

A user evaluation would need to explore the impact of the user interface which has been used in the HIPPO and HIPPO+ systems. Can the *point-and-click* model used in HIPPO still be used to support fuzzy anchors? How effective is the use of colour to represent different intensity values of anchors? The user may find alternative graphical representations more useful, and some of these have been discussed elsewhere in this chapter. Similarly, a user evaluation may find that the user requires a more concrete representation of linkbase trees, perhaps with some graphical interface which can be used to display and manipulate linkbase hierarchies. The HIPPO+ interface in particular would benefit from some form of user testing and interface evaluation. The implementation described in this work is a prototype which aims to demonstrate the advantages of distributed hypertext services. However, the interface which allows users to interact with these hypertext services could be developed further, and some form of iterative prototyping phase would be useful.

The testing plan should also assess the ways in which the HIPPO model impacts the working patterns of the user. Do fuzzy anchors alter the way in which a user perceives a hypertext? Does the user gain a stronger sense of *anchoring* when using these fuzzy anchors? Do authors share and reuse existing link collections? Are users encouraged to think in more fuzzy, less discrete terms, by examining the strength of hypertext links and relationships? This might be done by casual observation of users interacting with the system, or via more structured methods – question and answer sessions, comparisons with other hypertext systems, the logging user

navigation etc. Finally, it would be interesting to examine the way in which HIPPO and HIPPO+ can support larger numbers of users. Many of the ideas presented in this thesis encourage a more collaborative approach to hypertext in which users share resources and build on existing work. The adaptive modelling ideas allow users to influence other users of the system – suggesting useful link collections or important hypertext services. This is also an area which should be explored further, perhaps using some form of user trials, involving larger numbers of users. The user testing of the HIPPO and HIPPO+ systems has been largely ignored in the current work, but this is vital to fully evaluate the ideas presented in this thesis, and should be incorporated into any future work.

7.3 In Conclusion ...

This chapter has attempted to summarise the key points of my research on the HIPPO system. I have outlined some of the main ideas, and showed how these were developed over the duration of my work. This thesis combines ideas from a number of different disciplines – open systems, distributed architectures, adaptive hypertext, object-oriented design. These are all very exciting areas of research which are undergoing constant change and development. The success of the Internet and World Wide Web have also made a significant impact on the hypertext community and continue to influence the development of open hypertext environments. The work which I have presented here offers just one way of combining these diverse areas, and this initial attempt may not prove to be the best way. However, I do feel that future research into open hypertext systems should continue to combine ideas from other disciplines, and I hope that this thesis has offered some interesting ways of achieving this.

Paul Newton
1998

Appendix A

Early Hypertext Applications

Researchers have widely interpreted the ideas of hypertext, and applied these to many different domains – macro literary systems, problem exploration tools, browsing systems, general hypertext applications [Con87] etc. Indeed, it is often difficult to identify what exactly constitutes a hypertext system, or even what the defining features of hypertext should be. One could argue that this is precisely as it *should* be; hypertext addresses the universal problem of knowledge structuring and information management, and it is only natural that this should appear in many environments. Engelbart [Eng95] argues that *all* documents are inherently hypertext documents – reports, programs, notes, manuals etc. For this reason, elements of hypertext have been applied universally, and it is difficult to identify any closure in the hypertext domain. This appendix focuses on some of the more important hypertext systems which are widely viewed as significant contributions to the field. It should be noted that these systems are all typical of the early generation of hypertext, which gave rise to more open, flexible hypertext environments. These later systems are not included here, but are discussed in more detail in Appendix B.

A.1 Augment/NLS

The Augment/NLS system[Eng84a] was one of the pioneering systems in the hypertext field, designed as a problem solving tool for *"planning, analysing and designing in complex domains"*. Augment/NLS was not developed specifically as a hypertext system, but employed many hypertext features for managing the many thousands of reports and papers at the Stanford Research Institute. NLS/Augment files consisted of hierarchically organised nodes, which could each contain a variety of data types (text, graphics etc). In addition to this hierarchical structure, links could be established between statements and files.

Augment supported a flexible addressing mechanism for nodes, allowing explicit selection, relative addressing, node identifiers etc—these are then used in Augment commands to manipulate the hypertext. Augment also developed the idea of views, using multiple windows and user- controlled view filters to control the display of information. Augment/NLS also gave special consideration to group collaboration which was considered an important part of information processing, providing facilities such as teleconferencing, which allowed several users to view and edit files, and integrated electronic mail.

A.2 Xanadu

Ted Nelson was another early researcher in the field of hypertext, and his ambitious vision was of a completely integrated hypertext containing all the world's literary works [Nel93]. This unifying system allows any data to be linked to any other object in a non-hierarchical manner, and never deletes objects, instead storing the original document along with any changes. Nelson's vision is based on a combination of back-end and local storage systems, and allows any object throughout the Xanadu universe to be uniquely addressed, and transparently accessed. Xanadu makes a strong separation between user interface and database storage, predicting many front-ends running over a single back-end. Xanadu addresses many of the wider issues of information dissemination such as copyright, security, auditing and payment etc, neglecting traditional copyright measures, but paying royalties based on the access to information. Some parts of the Xanadu system have been implemented, and *Project Xanadu* has continued through a number of commercial sources (Xanadu Operating Company, AutoDesk Inc.).

A.3 TEXTNET

TEXTNET [Tri83] was designed for structuring and manipulating scientific literature—addressing all aspects of critiquing, refereeing and paper writing in the

"on-line scientific community". A TEXTNET network defines two types of nodes—*chunks* and *toc* nodes: chunks an be thought of as the primitive content, and toc nodes correspond to organisational nodes (cf. an entry in a table of contents). TEXTNET implements a typing mechanism for links, connecting nodes with typed links which are meant to capture the relationship between nodes.

Chunk nodes consist of structured attribute/value pairs which contain information about the node (author, date, list of links etc), and a pointer to the actual node data; this allows arbitrary size node contents. These chunk nodes are then organised into directed, acyclic hierarchies, using *toc* nodes. Links connect chunk and toc nodes and capture the explicit relationship of two nodes, with a link type—such as *Summary, Argument-by-Analogy, Example, Continuation*. Users can augment the network in any way—by commenting on the text, linking to other nodes, or even critiquing other users comments. Readers can also formulate multiple structures and create paths through the document space, which allows subsequent readers to follow these threads. This is in many ways similar to Bush's idea of *trails* in his memex device, whereby readers can benefit from the work of others.

TEXTNET incorporates ideas from the previous systems, Xanadu and Augment, supporting tree structures, but implemented using a graph structure at its core. However, it is the focus on the typing of links to capture the essence of a relationship which separates it from many systems, a feature which is often lacking in many modern day systems.

A.4 ZOG/KMS

Work began on the ZOG Project[AM84b] in 1972, at Carnegie-Mellon University, to design a general purpose system combining features of database, word processing and operating systems. ZOG was later developed into a commercial version, known as KMS. The KMS data model consists of screen-sized nodes, called *frames*, which may contain combinations of text and graphics. These frame nodes are organised into hierarchies, and frames may also be linked together across hierarchies, using annotation links. Links are contained within the fixed-size frames, linking an area of text to a node (links cannot link to parts of nodes). KMS supports two types of links—tree items and annotation items; tree links are used for the hierarchical structures and annotation items are used for referencing peripheral material such as comments and other frames. Versioning of frames is supported using stacks of frames, representing the history of changes of a node.

KMS provides some support for collaboration, by distributing the hypertext network across multiple servers, so the physical location of a frame is transparent to the user. KMS assumes that interference between users editing the same frame is rare in a large hypertext, and so assumes an optimistic concurrency policy where users can

access the same frame yet only one author may save changes. Users are expected to leave messages on frames to indicate that they are editing the information, and can be used to support simple communication. The use of annotation links allows users to easily add items to frames, to add comments and discuss work with other users. In addition, KMS provides a general purpose programming language for manipulating KMS structures which allows the functionality of KMS to be extended by the users.

A.5 NoteCards

This system, developed at Xerox PARC is one of the most well-known hypertext systems, and its design and implementation are well documented. NoteCards[HMT87] operates under the Xerox InterLisp environment, and was designed to aid people with the formulation and structuring of ideas. The system is based around four main ideas—*notecards, links, browsers* and *fileboxes*. A notecard encapsulates the idea of a hypertext node which may contain arbitrary amounts of text, graphics etc, and are arranged into networks using links. These links are typed, directional connections between cards, and like KMS links, allow a specific sub-node anchor to link to a complete node. Users specify the type of link by including a label, which denotes the nature of the relationship. A browser is a specialised form of notecard, which contains a structural overview of the hypertext network showing the notecards and interconnecting links, and allows users to edit and alter the structure of the hypertext. Fileboxes are also forms of notecards, which can be used to organise and categorise large collections of notecards and encourages users to use additional hierarchical structures to arrange the notecard.

Users access the notecards either by navigating through the network, or using the overview browsers to select items of information. The system also provides a limited search facility, allowing the user to locate cards in the hypertext structure. The functionality of the system can be extended using the Xerox InterLisp programming environment, allowing the user to create new types of cards or integrate other applications, for example, the Instructional Design Environment (IDE) was built on top of the NoteCards system, and completely customised to produce a new system. The original version of NoteCards has been used within the Xerox organisation, and externally in industry and academic institutions, and has been a very influential system in the hypertext community. Halasz' important paper on the future of hypertext research discussed the lessons learnt from the design of the NoteCards system, and highlighted some of the issues which must be addressed by next generation hypertext systems (see Chapter 2).

A.6 Intermedia

Intermedia[YHMD88] was another very influential hypertext system – developed at Brown University – designed to support teaching and research in an academic environment. The system is heavily influenced by the copy/paste interface and allows users to connect regions using bidirectional links. Anchor and link information is not stored with the documents, but is stored in a database managements system, and superimposed on the documents. Links can be grouped together into *webs* — this allows users to maintain their own sets of links. This is a technique developed in *open hypertext systems*, which are discussed further in chapter 2. Central to the idea of Intermedia is the development of a linking protocol, allowing applications to be integrated into the Intermedia environment. In this way, users can incorporate hypertext linking into their applications, allowing their documents to be the source and destination of links. The system provides a number of integrated applications to manipulate text, graphics, animations etc, the user is also provided with a number of browser applications for viewing files and link information. Intermedia supports multiple users accessing the hypertext, which can be distributed over a network; concurrency control is implemented using a locking mechanism.

The Intermedia system aimed to provide a seamless information environment for educational use, and was successfully used to teach several university courses. The educational focus had some impact on the design of the system, and despite a promising future, government funding was discontinued in 1991.

A.7 Neptune

The Neptune system[DS86] was designed for use in the field of engineering and Computer Aided Design, and provided explicit support for collaborative work. Neptune is designed as a layered architecture, built on top of a transaction-based server, the Hypertext Abstract Machine (HAM). The HAM was designed as a general purpose hypertext engine which can be used as the base engine for other hypertext systems, and provides storage and access mechanisms for nodes and links. The HAM runs as a central server, which provides distributed, multi-user access and supports complete recovery from any aborted transactions.

Neptune maintains a complete version history of the hypertext network, allowing instant access to any version of the graph, and direct comparisons of different node versions. Neptune provides browsers for viewing the hypertext: a graph browser which provides a pictorial view of the hypertext, a document browser for viewing hierarchical structures, and a node browser for examining the content of the nodes. Other browsers are provided for viewing attributes, version histories etc, and future development focused on multiple threads and additional support

for collaborative authoring. Neptune was another of the early hypertext systems to pay particular attention to collaborative authoring and version control, and the HAM server was one of the first general-purpose hypertext models to have a significant influence on the development of hypertext engines. The idea of *hyperbases* is explored in more detail in chapter 2.

A.8 Guide/OWL

The GUIDE hypertext system[Bro89] began as a research project at the University of Kent, and was later developed by OWL as one of the first popular commercial hypertext systems. Guide was developed to aid the reading of electronic documents, and adopts the scrollable window model of hypertext. Hypertext links are implemented as embedded buttons — the system supports three forms of button: replacements, pop-ups and reference buttons. Replacement buttons are used to expand text in-line, these implement hierarchical structures within the text, and are useful for the traditional modelling of document chapters and sections etc. Pop-up buttons are used to generate out of line information, which is displayed in a separate window and are used for adding annotations, extra information etc. The final type of button models most closely the traditional view of a hypertext link — the reference buttons allow the user to jump to another location in the hypertext. Guide has proved to be a popular commercial system, and offers a different view of hypertext to many other systems.

A.9 HyperCard

HyperCard[App87] has played a major part in the history of hypertext systems, and was perhaps the most famous hypertext application during the late 1980's. HyperCard was not designed as a hypertext system, but as a graphic programming environment.. Much of its success was due to the aggressive marketing by Apple Computers, who distributed the system free with every Macintosh computer sold. HyperCard, as its name implies, is based on the card metaphor, much like the KMS system described previously. Cards are grouped together into stacks, and may be connected using buttons, which when activated, jump to another card in the hypertext.

One of the features of the HyperCard environment, and one of the reasons for its popularity, is the inclusion of a programming language, *HyperTalk*, which enables buttons to be programmed with arbitrary actions. This allows more complex actions to be invoked, such as computed links, and conditional actions, allowing the functionality of the system to be altered and extended. Although not strictly a hypertext system, HyperCard and its various incarnations on other platforms (Su-

perCard, Plus, MetaCard etc) have been an important influence on the hypertext community, and for many, have been their first experience of a hypertext system.

Appendix B

Open Hypertext Systems

Chapter 2 introduced the idea of *open hypertext*, where hypertext is used to *integrate* applications. An open hypertext system provides *link services* which can be used by any existing tools and applications, instead of implementing hypertext as a closed, monolithic application. This section describes some of the more significant systems which have been developed, and which influenced much of the work on HIPPO. This is not an exhaustive review of open hypertext research, and readers are referred to the literature for more information on some applications.

B.1 Sun's Link Service

Sun's Link Service[Pea89] is a product shipped by Sun Microsystems with their Network Software Environment (NSE), and provides conventional applications with a set of hypertext link services. Sun's LS loosely couples the management of links and data, by separating hyperlinks from the node contents. The Link Service stores only representations of the hypertext nodes, which allows nodes to contain arbitrary data formats, and so any application can be integrated into the link service environment. The Link Service provides persistent storage for linking information, and supports a protocol which applications can use to incorporate hypertext functionality (figure B.1). By separating the hypertext linking mechanisms from the storage of node data, the Link Service provides generic hypertext services, which allow users to make use of existing applications without adapting to a new environment.

The link service has no control over objects within the hypertext environment, and it is left to the managing application to define hypertext nodes; in this way, the hypertext makes use of abstractions which are suitable for the particular application. The Link Service is designed to have minimal impact on the user interface layer, and it is the responsibility of applications to provide suitable mechanisms for selecting objects, and provide visual indications for links. Pearl recommends the adherence to user interface standards such as the OpenLook[Hoe89] specification,

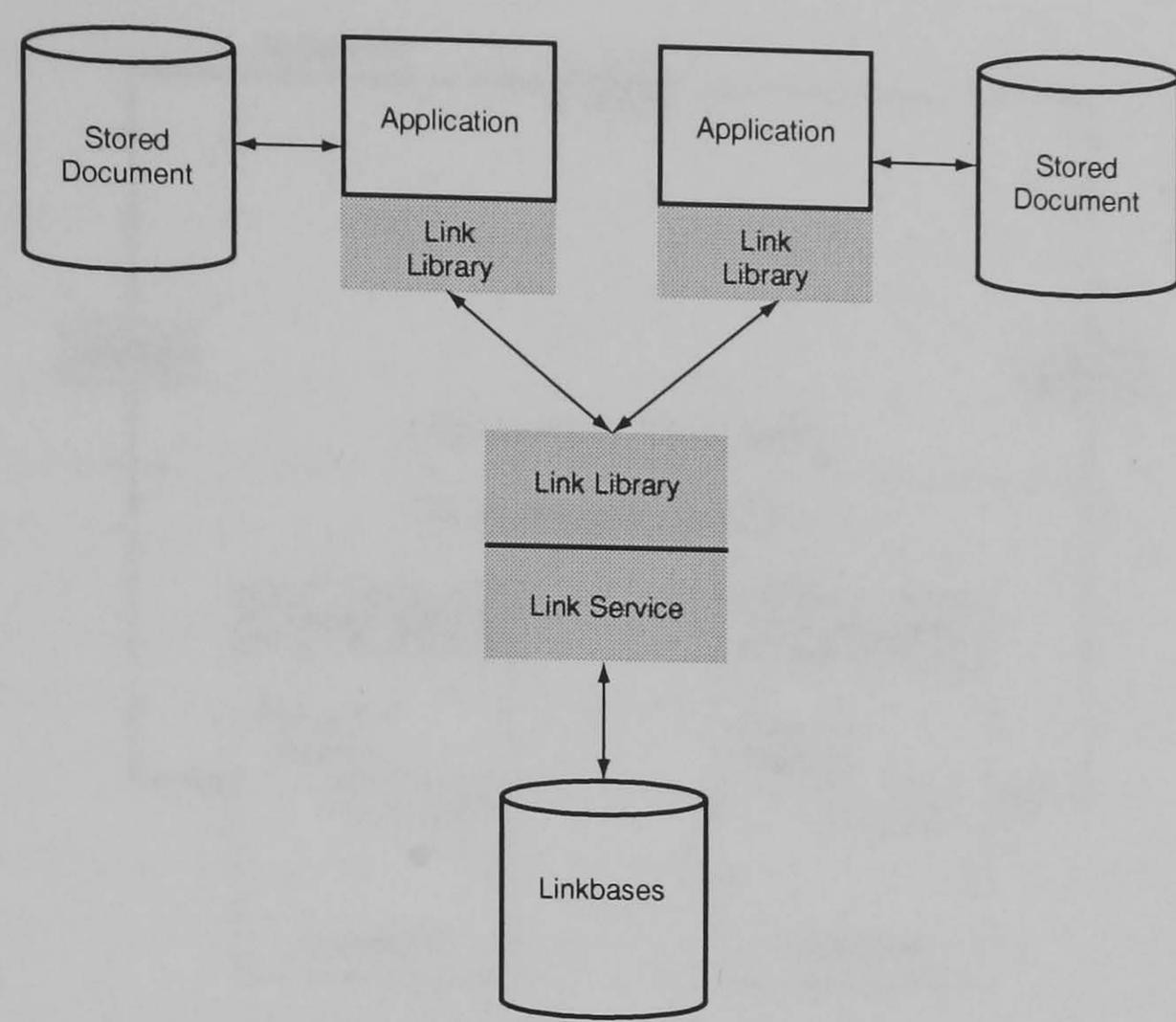


Figure B.1: Sun's Link Service

to improve consistency between applications.

The Sun Link Service is designed for use in distributed environments, and supports access to remote file systems using the distributed Sun workstation environment. Links are defined as untyped and directionless, but the system does support multiple branching links, allowing an anchor to reference multiple objects, and also be the destination of multiple links. Users must run a link server process on their local machine, and applications are required to register with the server if they wish to use the hypertext services, and the server will then inform applications of any state changes. The Link Service protocol is defined to be as simple and unrestrictive as possible, in an attempt to "...preserve the autonomy of individual tools, yet provide some measure of integration". Sun's Link Service was one of the earliest open system implementations, and has proved very influential in much of the research in this area, however the system does have a number of drawbacks, as there are no facilities for extending the semantics of hypertext links, and there is little support for distributed and collaborative applications.

B.2 SP1/2/3

The SP3 [LS94] system was developed from work on the SP1, SP2 and PROXHY architectures, designed for managing "hypermedia-in-the-large" by supporting the distribution of hypertexts across heterogeneous platforms and applications. The SP3 model adopts a computational view of hypertext, which defines links and anchors as behavioural entities. The SP3 model defines six elements to support hypertext services: applications, components, persistent selections, anchors, links and

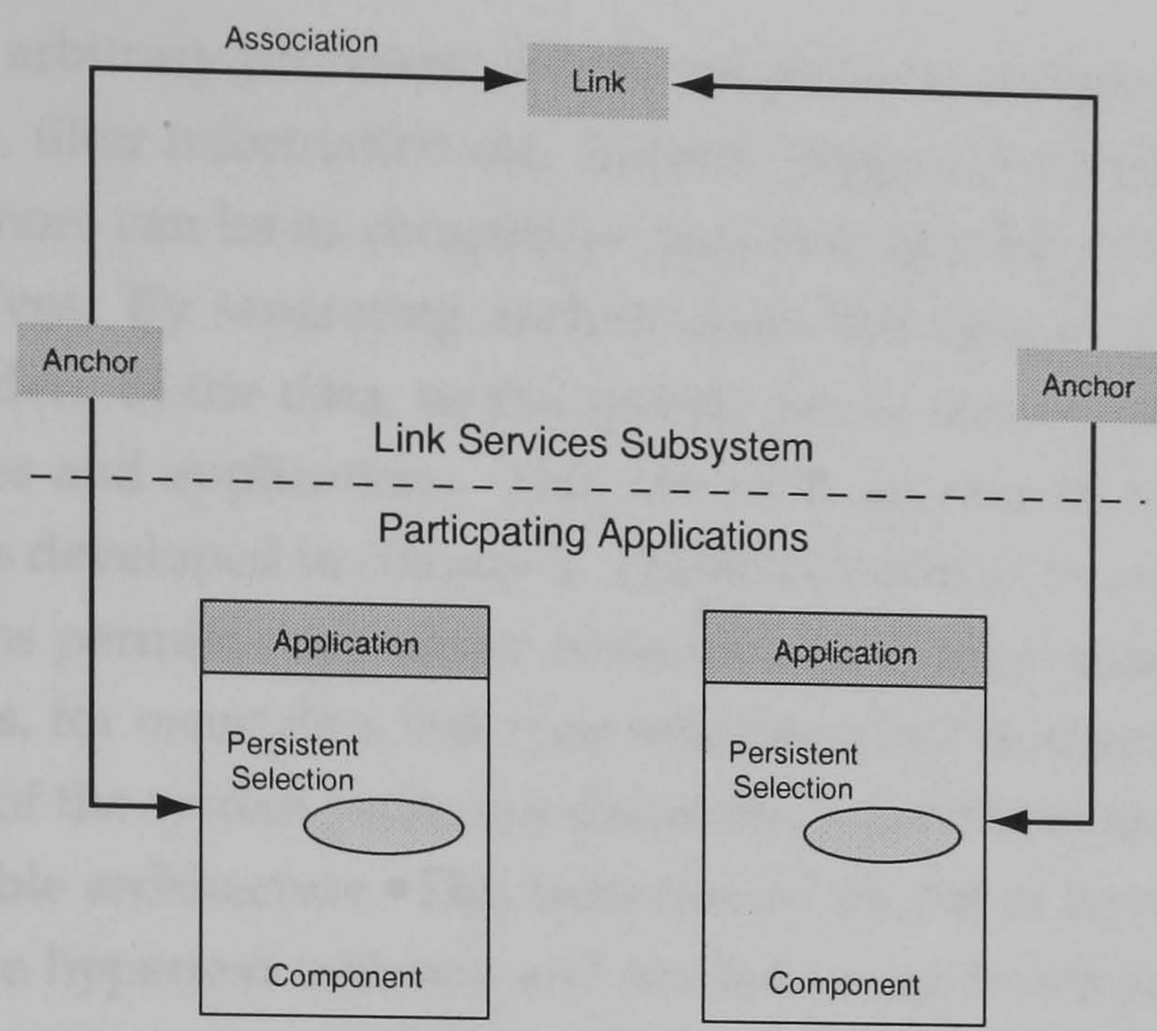


Figure B.2: The SP3 architecture

associations. Applications are the programs and tools which are used by the user to manipulate specialised information. Components are the actual data elements which are manipulated by these applications. Persistent selections are selections within the components which exist between sessions, and can be recalled at a later time. Anchors are associated with persistent selections and links with anchors, and are implemented as program processes. These links and anchors are tied together by associations, and it is these three elements which implement the hypertext behaviour (see figure B.2).

The SP3 system is built on top of the HB3 multi-user storage engine, which implements the versioning of objects and provides persistent and sharable storage. Applications participate in the SP3 hypertext services by interacting with the Link Services Manager, which requires them to handle persistent selections and respond to LSM requests. Applications are responsible for creating, deleting and displaying these persistent selections, and for manipulating the PST data structures. All message passing is performed using X Window System IPC facilities, and the development of an X Link Services Toolkit allows processes to implement client-server communication.

The SP3 model offers a very different view of hypertext than most other approaches to open hypertext systems, by defining a *computational, behavioural* view of hypermedia. The system abstracts information, structure and *behaviour* from a hypertext environment. This contrasts with previous systems which focused on information and structure, and encapsulated hypertext behaviour in the system itself. SP3 has a strong notion of anchors, in contrast to other systems which implement anchors as data structures to specify the end-points of links. SP3 anchors are

implemented as arbitrary processes, which can perform complex computations to customise views, filter information etc. Indeed, Leggett *et al.* point out that these behavioural anchors can be as complex as required, and can even be information systems themselves. By separating anchors from link objects, the linking mechanism is independent of the data, so the system can be incorporated to include arbitrary data types and applications. This notion of anchors as first-class objects is important, and is developed in chapter 3. The abstraction of structural information, using associations permits support for contexts, and allows more complex anchor-link relationships, for example a link may relate as many anchors as required. The modular nature of the system promotes distribution and network optimisation and supports a scalable architecture. This behavioural model of hypertext is a powerful view of future hypertext systems, and has led to the development of the *HOSS* hypermedia operating system [NLS96] at Texas A&M. The work on computational hypertext has heavily influenced some aspects of the HIPPO model, and is explored further in chapter 5.

B.3 Microcosm

Microcosm [DHHH92] is a model for open hypertext which has been developed at the University of Southampton. Like many other approaches to open hypertext, Microcosm separates the linking information from data objects, which does not impose any markup on the data itself — this allows existing tools and applications be integrated more easily into the Microcosm environment. All information is stored in separate link files called *linkbases*, and the hypertext services are supported by a number of autonomous communicating processes. Messages are passed through a chain of these filters, and each filter responds to the message as required (figure B.3). The Microcosm model supports three different forms of linking: *specific* links, *local* links and *generic* links:

1. Specific links refer to particular objects at a specific points in the source and destination documents.

2. Local links connect an object which appears anywhere in a *specific* document, to an object in a destination document.

3. Generic links join a particular object which appears anywhere in *any* document, with an object in a destination document.

The implementation of specific links most closely matches the linking policies in conventional systems, but the notion of local and generic links represents a departure from conventional linking practices. For example a generic link from the textual anchor *Paris* would become active in all documents within the Microcosm

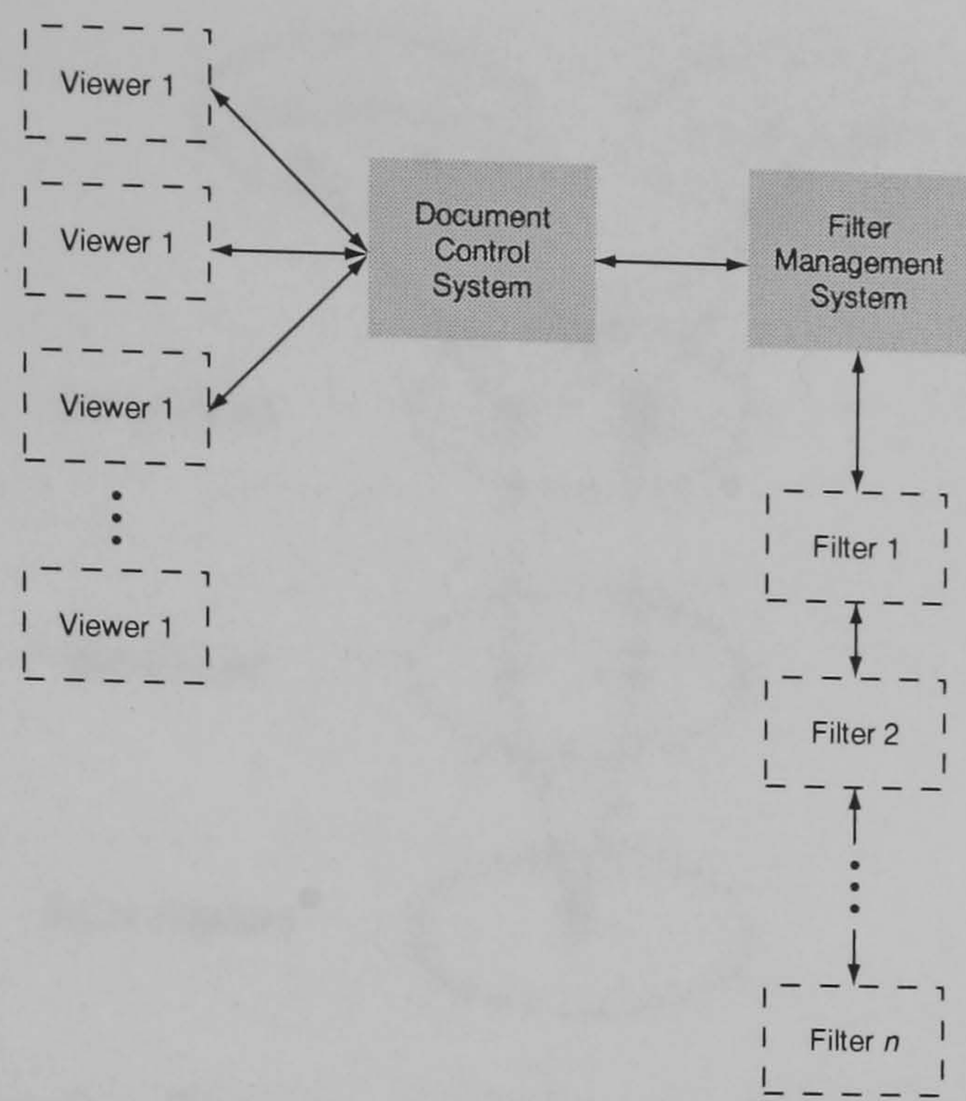


Figure B.3: The Microcosm model

domain, so all references to the word *Paris* would include a hypertext link. This might reference a geographical definition or a tourist guide for example, and is designed to help reduce the often overwhelming authoring effort required to incorporate information into current hypertext systems [FHHD90]. Any document which is included in the Microcosm system automatically absorbs any relevant generic links, and is seen as a powerful mechanism for expanding the hypertext environment.

Microcosm adopts a *resource* based approach to hypertext, which means that users must select items of interest, and query the link services to see if a link exists for the object, rather than the more usual method of displaying links. However, Microcosm does support buttons in fully-aware viewers, which consist of visual link markers bound to specific links — these can be selected and traversed in the same way as conventional hypertext links. The system of filters which implement the Microcosm hypertext functionality are a very simple yet effective method for supporting link services. Users can dynamically configure the chain, and can incorporate arbitrary filters to provide additional services, for example, a filter has been implemented which maintains a history of user requests by monitoring the messages which are routed through the filters. Other filters allow users to compute links based on user input, or mimic filters generate messages which provide the user with a guided tour. Also, by including linkbases in the filter chain, users can easily add and remove sets of links, which implements a form of link contexts.

The separation of Microcosm linkbases does not impose markup on existing data formats, and allows the linking information to be processed externally, for example, computing a set of links which match a query. However, Microcosm does not develop the notions of anchors and links as much as the SP3 system, and does

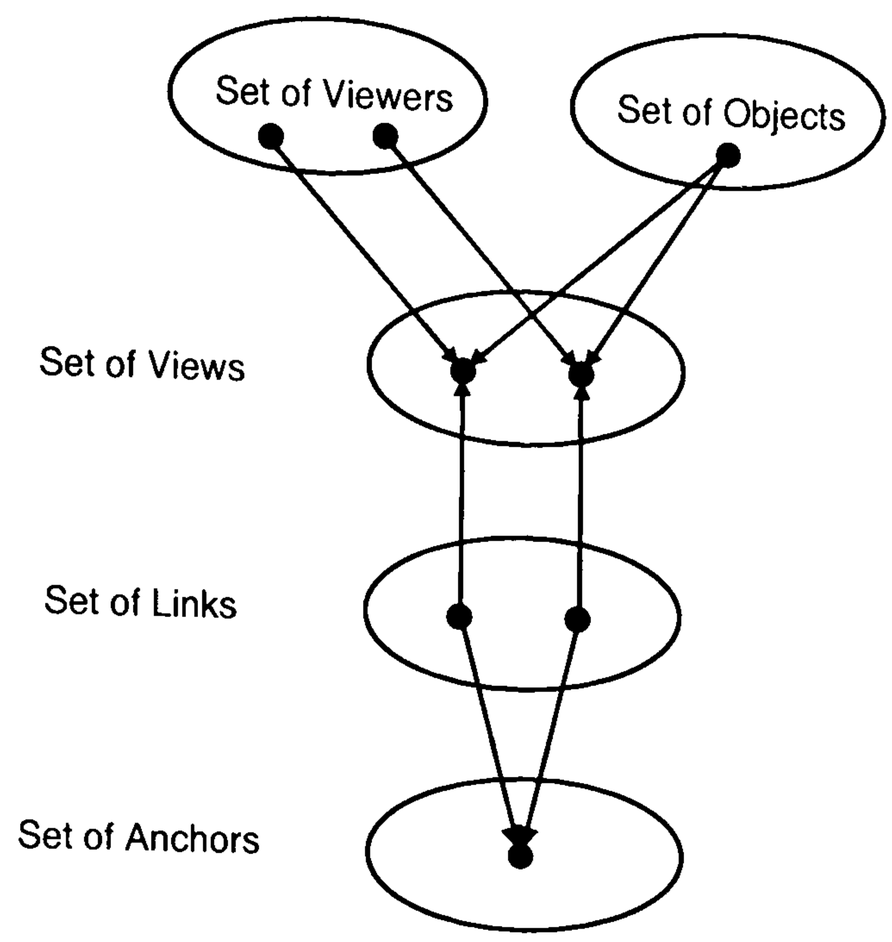


Figure B.4: The use of views in the Chimera system

not fully address the behavioural, computational aspects of hypertext. In addition, Microcosm requires the linkbases to manage persistent selections, which can give rise to consistency problems if applications edit anchors and links without notifying the linkbases. However, the ideas of communicating filters which can be dynamically configured has greatly influenced the work on HIPPO covered in chapter 5. Also, the separation of linking information, while not unique to Microcosm is used as the basis of chapter 4. The Distributed Link Service (available commercially as Webcosm [Web]) also provides an implementation of this linkbase model using the World Wide Web.

B.4 Chimera

Chimera [ATJ94] is a hypertext system designed for use in software development environments (SDEs), and for providing hypertext services across heterogeneous applications. Anderson *et al* [ATJ94] argue that hypertext anchors are more naturally associated with a view of an object, and so Chimera defines anchors with respect to views on objects rather than the objects themselves (figure B.4). In this way, anchors can be expressed in terms of presentation, run-time objects such as buttons etc.

The Chimera system is implemented as a client-server architecture, with the Chimera server providing hypertext services to other applications, and managing the routing of system messages. Chimera clients are responsible for defining the object and view concepts, and for supporting the manipulation of hypertext anchors. It is also the responsibility of the application to provide persistent storage of

object data — the Chimera server only manages hypertext information, and is not concerned with any application information.

Chimera chooses not to enforce a single object store but allows multiple storage mechanisms, so does not handle persistent storage of application object — this makes issues such as versioning more problematic, and must be a shared responsibility between Chimera and client applications. Also, Chimera does not address any collaborative issues — this is in part due to limitations in the Chimera architecture, although future versions of the system plan to address this restriction.

B.5 Dexter Hypertext Reference Model

The Dexter reference model[HS90] was the result of two workshops on hypertext theory held in October 1988 at the Dexter Inn, New Hampshire, which gathered together many important representatives from the hypertext community. Although the Dexter model does not strictly address open hypertext systems, it does attempt to explore the fundamental abstractions found in classical hypertext systems, and has become one of the most important and influential models to emerge from the field of hypertext research. Chapter 2 showed how these can be of great benefit in the hypertext field, because they allow us to reason about alternative abstractions, and use formal techniques to compare different systems. Reference models are precise and unambiguous, and provide a basis for developing standards for interoperability and interchange between systems.

The Dexter model addresses the dynamic, interactional aspects of hypertext, but focuses more on the structures involved in implementing hypertext systems. The model is specified precisely using the Z-specification language, and defines three layers: a *run-time* layer, a *storage* layer and a *within-component* layer, although the focus of the model is on the intermediate storage layer (figure B.5). The Dexter model acknowledges the widely varying definitions of fundamental concepts in different hypertext systems, and adopts less ambiguous terms such as *component*.

The storage layer addresses the basic node-link structures of a hypertext, and is composed of a network of components, interconnected by links. These components are treated as generic containers, and do not address any modelling of internal structures. A hypertext is defined as a finite set of components, and a set of resolver and access functions which are used to manipulate the hypertext. The within-component layer is used to model the content and internal structure of the hypertext data, but this is not elaborated further by the Dexter model. This gives the Dexter model the flexibility of not being limited to certain data types, an aspect which is seen in many implementations of contemporary open systems. Finally, the run-time layer is designed to support presentation mechanisms in the hypertext system, and captures the essentials of the dynamic and interactional aspects.

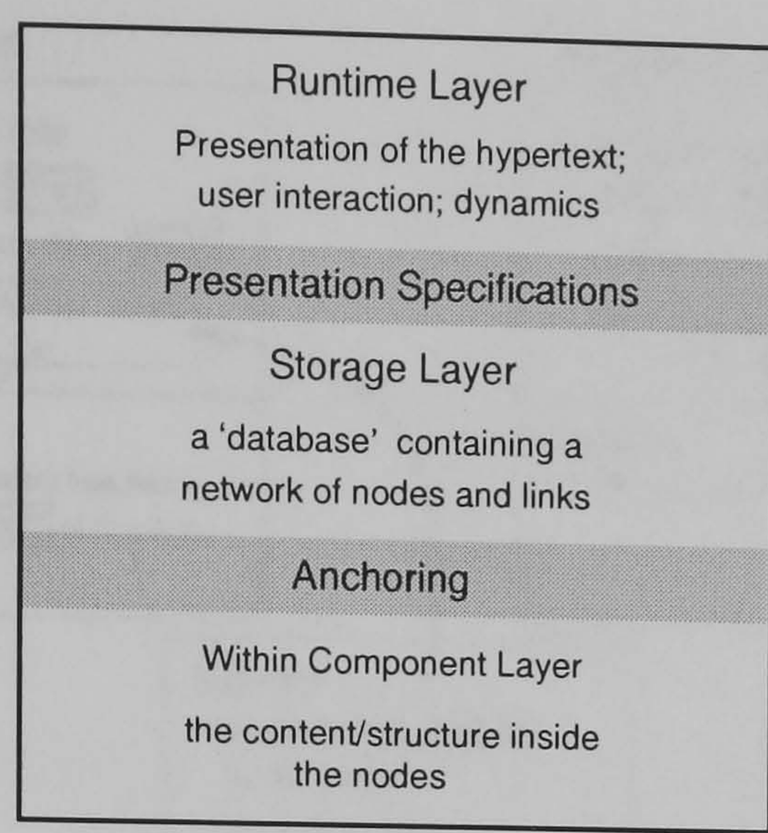


Figure B.5: The Dexter Reference Model

Dexter uses the anchoring mechanism to interface between the storage and within component layers, and uses presentation specifications as an interface between the storage and runtime layers (figure B.5). These interfaces are an important part of the model, because they allow a clean separation between layers. Presentation specifications specify how the component or network is presented to the user by the runtime layer, which may depend on the application or some other properties of the component. A hypertext link consists of anchor specifiers, which can contain arbitrary contents, and are interpreted by the particular application.

The Dexter model is perhaps more powerful and abstract than many classical hypertext systems, supporting features such as multiway links, links to links and computed links, and this has made the model difficult to implement. A number of systems have attempted to implement the Dexter model, or have been based largely on the model [GT94, RS92]. Since the model was designed, a number of problems and limitations have been identified with the model (problems with dangling links, reuse of anchors, composite nodes etc [GT94, LS94]). However, the Dexter reference model is a very powerful design, which captures many of the best ideas from classical systems, and has proved to be very influential in the hypertext community. Although the system was defined with respect to conventional monolithic systems, the model does exhibit many open design policies by distinguishing between objects belonging to hypermedia, and objects belonging to the applications. However, the model fails to develop some key abstractions such as link, anchor and composite semantics, and does not address key issues such as sharing, versioning etc.

B.6 MultiCard

MultiCard [RS92] is an implementation based on the Dexter reference model which supports the runtime, storage and within-component layers, and provides a pro-

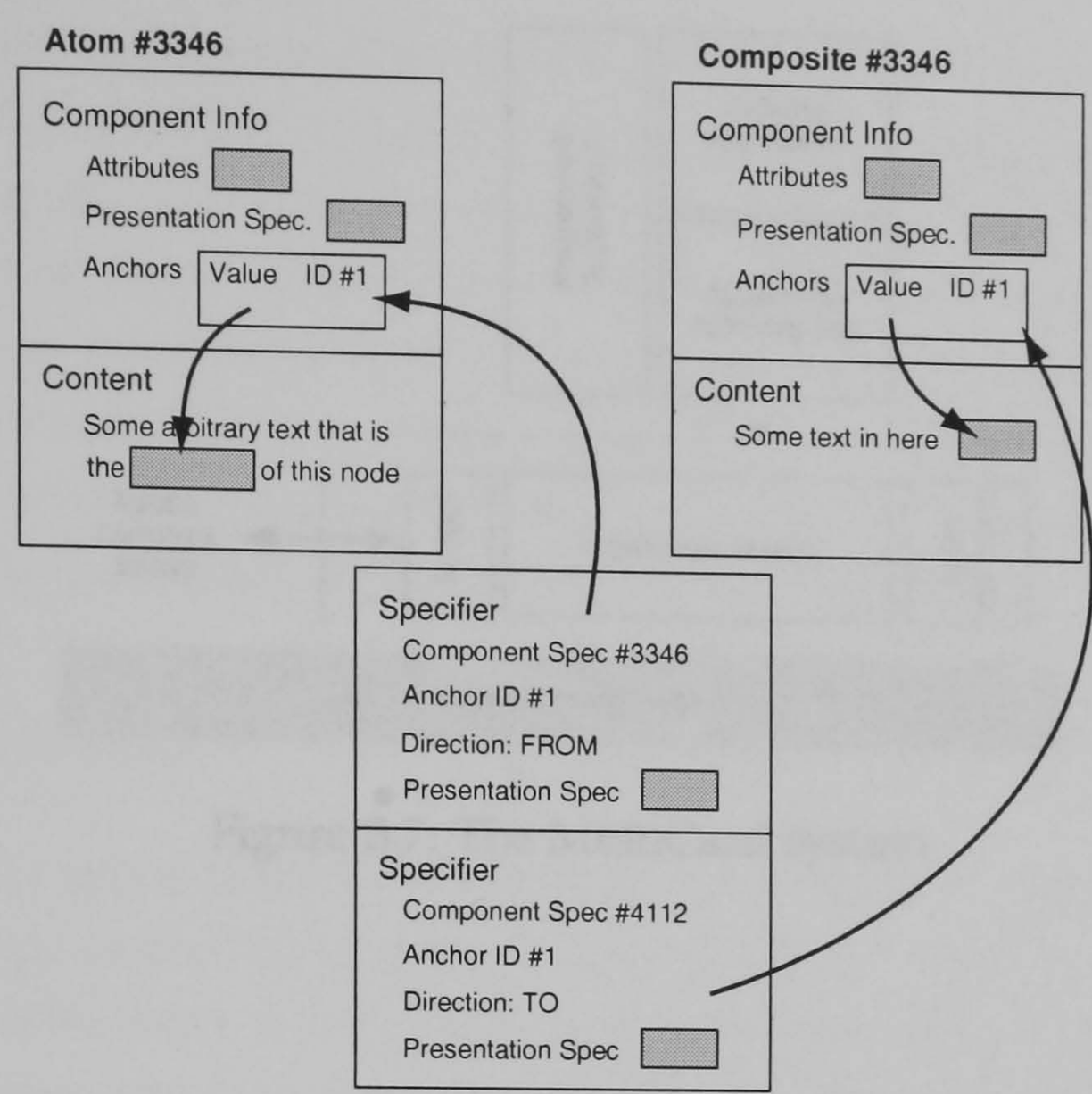


Figure B.6: An example using the Dexter model

tocol which implements the Dexter anchoring interface. The MultiCard system consists of a set of hypermedia classes, a distributed persistent storage system, an authoring and navigation tool, a communications protocol and a series of editor applications. This MultiCard architecture is shown in figure B.6. MultiCard manages the structure of nodes, and uses the M2000 protocol to access parts of documents, but as with many other open systems, the actual node content is managed by the editing applications. MultiCard anchor objects are responsible for carrying links, scripts and other objects, and define a sensitive area of a hypertext node. MultiCard extends the behaviour objects by attaching scripts to nodes, groups and anchors, which are used to manipulate editor contents, define functions etc.

Editors can provide different levels of support for the M2000 protocol, but must support the minimum requests to open and close nodes. Users then traverse links by clicking on anchors and MultiCard executes the appropriate script, although it is the responsibility of the editor to define what is meant by an anchor. This approach has much in common with the *computational* view of open hypertext which was discussed earlier, and seen in systems such as PROXHY, SP*x* etc. Using this model, MultiCard separates the hypertext structure from the application data, and provides a set of hypertext services to applications which comply with the M2000 protocol.

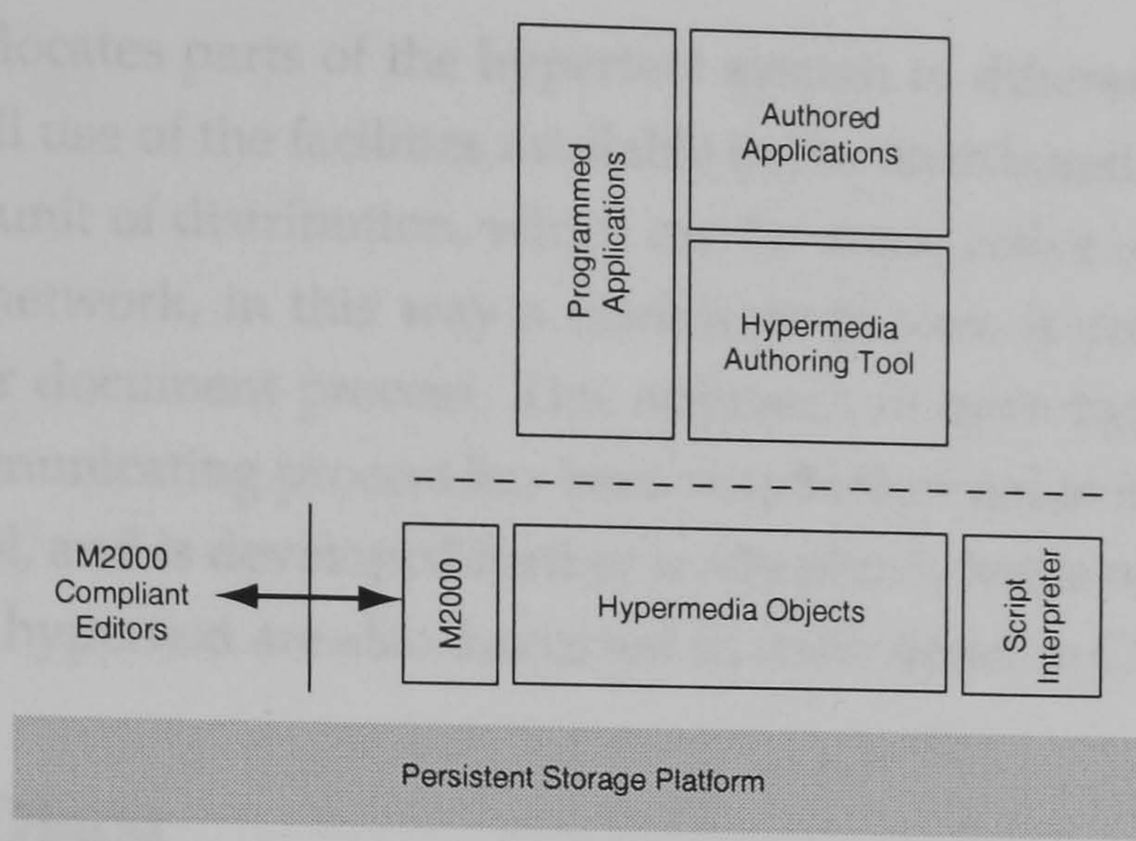


Figure B.7: The MultiCard system

B.7 D^2

The Distributed Documents architecture (D^2)[HGC94] continues work on open systems, by providing a set of generic hypertext services in a distributed environment. The architecture exploits a networked environment to provide access to remote information and services by distributing application services as well as the hypertext documents themselves. The D^2architecture is based on the notion of service provision and consumption — services are accessed by objects using a series of well-defined interfaces, and it is these co-operating objects which provide the hypertext services. The D^2model defines a number of basic entities: documents, nodes, links and anchors. The D^2model also defines view documents which support the concept of dynamic documents. These view documents may contain sets of links and nodes, but also execute a computation to produce references to other nodes and links. In this way, view documents support active documents which introduce an additional form of structuring to the conventional document object.

D^2defines three separate layers: the *storage* layer, the *hypertext* layer and the *presentation* layer. The storage layer consists of storage objects such as database managers, file systems etc, which provide the storage and retrieval services for the system. The presentation layer provides the services for displaying and presenting nodes, using presentation objects. These presentation services should support the display of anchors and links, and be capable of supporting composite nodes. The layer also provides services for displaying an entire document, using document presentation objects.

An architecture which is suitable for open computing must be simple to implement in a distributed environment, consisting of heterogeneous platforms across a network. Most current systems implement distribution using a single, centralised server, which does not take full advantage of the distributed paradigm. The D^2

model instead allocates parts of the hypertext system to different areas of the network, making full use of the facilities available in the distributed environment. The document is the unit of distribution, which can be made active on any workstation throughout the network, in this way a document process is completely independent of any other document process. This approach to open hypertext, as a set of distributed, communicating process has been very influential in the development of the HIPPO model, and is developed further in chapter 5. Some of the issues arising from distributed hypertext are also discussed in more detail in Chapters 2 and 6.

B.8 HyperDisco

The HyperDisco project aimed to provide an open hypertext platform, to support the flexible integration of existing tools. The application supports many of the design criteria outlined previously, such as computation, concurrency, distribution, versioning etc, but also focuses on the problems of flexible integration and extension of tools. Unlike most systems, which provide a fixed model of integration, HyperDisco allows tools to integrate at different levels. An application can have its own particular model of integration, and support its own specific protocol for accessing hypertext services. This builds on some of the earlier approaches to tool integration taken by MultiCard and Microcosm etc which allowed tools to provide different levels of hypertext support.

HyperDisco provides two layers of hypertext services – an *integration layer* to provide basic hypertext linking services, and a *data model* layer which implements basic storage services for hypertext objects. The HyperDisco implementation is based around an object-oriented paradigm, and the two layers are implemented as object hierarchies (figure B.8). These classes implement basic hypertext abstractions, and tools integrate with HyperDisco by extending and tailoring the class behaviour. Tools communicate with HyperDisco using an object-oriented message passing mechanisms, to service requests and invoke HyperDisco operations.

The HyperDisco system supports a diverse range of node media types, and allows new types to be added by specialising the class hierarchy. Multi-headed, directional are provided, and arbitrary computations can be associated with link traversal. HyperDisco supports many different aspects of hypertext behaviour – composites, access control, versioning of components and also some degree of collaboration between users – however, it is the approaches to tool integration and extensibility which are most interesting.

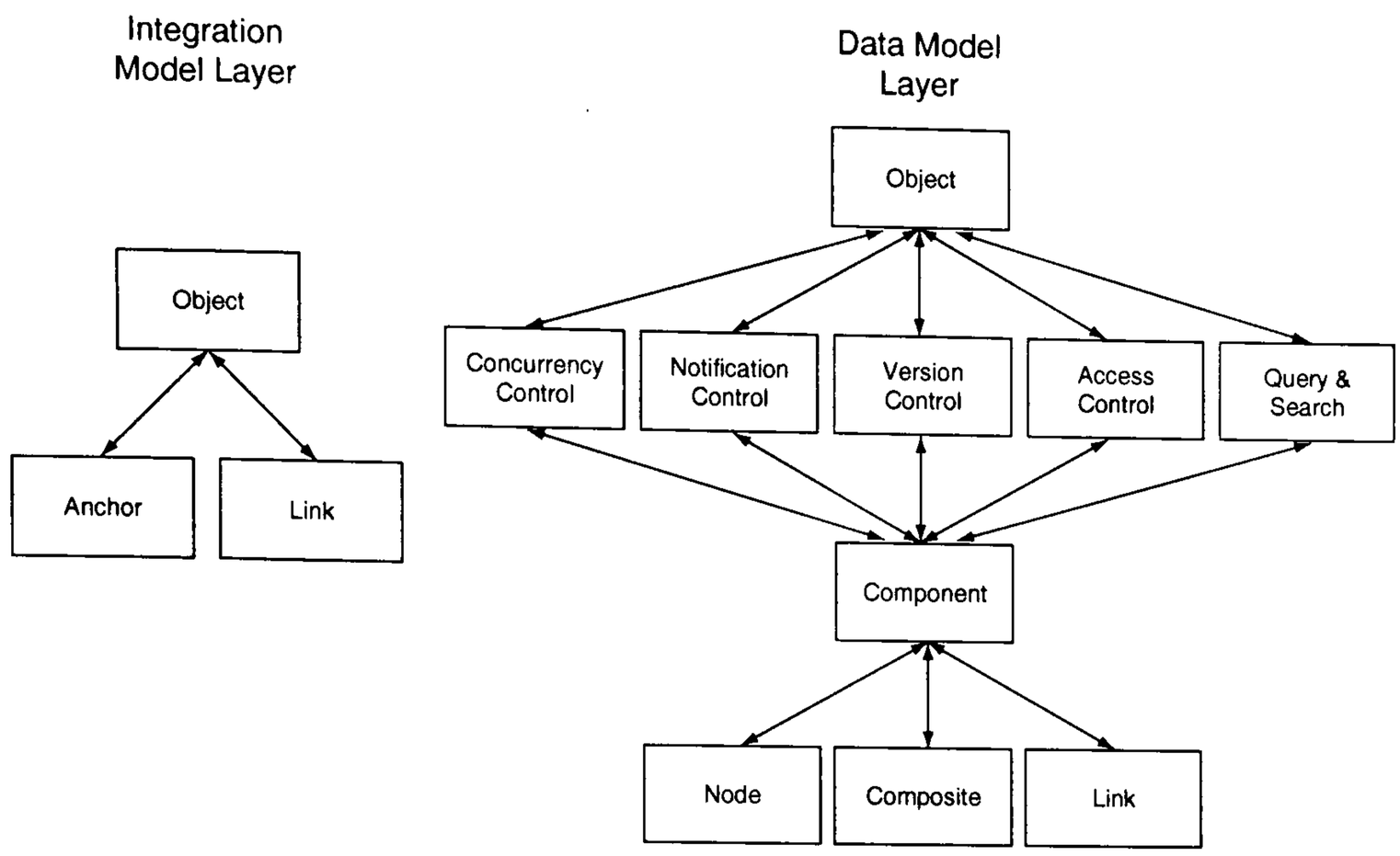


Figure B.8: HyperDisco class hierarchies

B.9 The Trellis Model

Most hypertext implementations and formal models focus on the structural properties of hypertext, and ways in which hyper-links can be supported. However, very few models address the *browsing semantics* of a hypertext, and the way that a user interacts with the hypertext. This is the main purpose of the Trellis hypertext model, which provides a formalism based on Petri nets, for both describing hypertexts and reasoning about their behaviour. Petri nets are directed graphs which uses *tokens* to decide whether transitions can be traversed. For example, a transition between two nodes can only be enabled, if the source node has a token marking; the absence of a token prevents the transition being fired.

This approach using petri nets builds on the work of directed graphs which are widely used in the hypertext community, but also allow the model to express and manage the browsing paths through a hypertext. Particular branches can be enabled or restricted, by careful placing of tokens and transitions. The model can control how a user moves through a hypertext and which paths they can choose. In this way, the petri net model provides the descriptive power of directed graphs, but also a mathematically precise abstract machine for execution.

The Trellis model views a hypertext, not as a passive structure, but as an automata which is *executed*. The model allows authors to specify the precise browsing semantics of the hypertext, and to define the ways in which the user can interact. There is also a substantial body of petri net theory which can be used to analyse and evaluate hypertexts which are defined using this model. The Trellis model can identify which nodes which can be reached, and which nodes cannot be reached by a browser. The petri net formalism is inherently concurrent, so can be used to ex-

press concurrent browsing paths and sychronisation issues. The petri net model can also be extended in a number of ways by using coloured tokens and timed nets to support access control, guided tours etc. The Trellis model is a very powerful model which is notable for its attention to the browsing semantics of a hypertext. The generalised model also offers a useful model of computational, behavioural hypetext systems, which has some analogy to the work developed in chapter 5.

B.10 The Hypertext Design Model (HDM)

The *Hypertext Design Model* (HDM) is a model for describing hypertext applications which already exist, or are being developed, in a system independent manner. While the Dexter model address very specific abstractions and aims to include the underlying primitives of particular systems, HDM is a more generalised model. HDM does not make as many assumptions about the nature of nodes, links, anchors etc, and does not assume any specific browsing semantics. This generalised model is intended to form the basis of hypertext design tools and hypertext development environments in the future.

The model defines a number of components – *entities, components* and *units* – which represent various abstractions of the underlying information. HDM also supports a number of different link types – perspective, structural and application links – which support the different roles of linking in a hypertext. These can include multiple representations of the same node, or some domain-specific relationships etc. HDM provides a number of other powerful modelling concepts such as hypertext outlines and anchor types to group multiple links etc. However, one of the most interesting areas of the model is its approach to browsing semantics.

The HDM model defines an abstract schema for a hypertext structure, yet it can also be used to create the hypertext applications themselves, by generating instances of the schema. This is done by defining the browsing semantics which map the underlying model to an actual application instance; this defines all the dynamic behaviour and the way in which a user interacts with the system (presentation information, link traversal behaviour, link visibility etc). In this way, the same HDM model can be used to generate different versions of an application, by providing multiple browsing mappings. The HDM model provides a number of *default* browsing sematics which are considered appropriate for a simple hypertext application, which minimises the effort involved in generating applications. The separation of browsing semantics from the hypertext structure definition is a very powerful aspect of the HDM model, and shares some of the behavioural concerns of other models such as the Trellis model discussed earlier.

B.11 MAX

Many hypertext systems are based around static, fixed problem domains in which the information remains largely constant over time. However, many disciplines require a more dynamic view of hypertext, in which the information sources are often not known in advance. Components can change and are often computed in response to user actions. In these cases, it is not feasible to define all nodes and links in advance, and these require a dynamic implementation of hypertext ideas.

Bieber explored the area *Decision Support Systems* [Bie91], and looked at ways in which these could be augmented with hypertext functionality. The MAX prototype identifies the following main components in a dynamic system:

1. Back-end Application Manager
   The Back-end Application Manager acts as layer between the decision support applications and the main hypertext interface layers. This back-end subsystem manages the knowledge bases and applications, and provides support for data management. Dyanmic information and requests are passed between the knowledge applications and the main hypertext engine. The other two layers are then responsible for processing the data, to dynamically create hypertext documents.

2. Interface Control Subsystem (CS)
   The CS handles all communication between the user interface and the back-end applications. The queries and reports which used to interact with applications are passed to the CS, and are compiled into (display-independent) interactive documents. These dynamically compiled documents are then passed on to the main interface, where they are presented to the user.

3. Display Subsystem (DS)
   The DS receives display-independent documents from the Interface Control Subsystem, which have been generated in response to user queries and actions. The DS is then responsible for formatting this information into a form which is suitable for display. The DS maintains an internal representation of all elements, to allow user actions to be processed (eg. Which marker is selected etc). The DS also maintains configuration-specific templates, formatting information, session histories etc.

The key difference between dynamic problem domains such as decision support systems, and conventional, static environments, is that the hypertext structure is not known in advance. Insetad, the hypertext engine must infer hypertext links, and dynamically construct hypertext objects at runtime. This task is supported by the Interface Control Subsystem described previously, which receives data from the

back-end application manager, and must identify suitable link endpoints. This can be done by examining the data stream for embedded keywords, which are placed by the back-end to identify hypertext objects. A more generalised approach is achieved using *bridge laws*; these are predicate rules which can be used to map application objects on to hypertext objects. These bridge laws can be used to generate hypertext objects for entire groups of application objects – in this way, a rich hypertext can be created using only a small number of bridge rules. In addition, MAX also supports explicit, user-defined links and task-oriented filters whcih allow the hypertext to be customised to the user. The MAX prototype is an important system which explores the role of hypertext in dynamic environments, and examines many of the issues which arise from adding hypertext functionality to third-party applications.

B.12 Hyper-G

Hyper-G has been developed as a large-scale open hypertext system, using a distributed architecture to provide hypertext services [AKM95]. The focus of Hyper-G has been to provide a hypertext model for managing large collections or documents, and to provide effective support for multiple users. Hyper-G provides a number of navigational paradigms to help the user explore a hypertext:

- collections
  Hyper-G documents can be organised into separate groups, known as *collections*. This allows related documents to be arranged into separate aggregations, which can then be used to form the basis of content searches, access control etc.

- hyperlinks
  Hypertext links are maintained separately from the underlying node content in the same way as some other open hypertext systems (eg. Microcosm [DHHH92], Intermedia [YHMD88]). The advantage of separate linking information is discussed further in chapter 4.

- attribute and content search
  Hyper-G also allows a set of keywords to be associated with documents and collections, so that users can perform searches on the document space. Hyper-G documents are also automatically indexed, so that the document contents can be searched. These searches can be used in conjunction with document collections to limit the scope of any queries.

In addition to the rich structuring services provided by Hyper-G, the model also provides a number of distributed services. Hypertext collections can be distributed

across a number of Hyper-G servers, and replication services are provided to update remote servers and to maintain consistency throughout the hypertext.

Users can connect using a special-purpose Hyper-G client (Harmony, Amadeus etc), or can use an existing web browser. The servers will also provide some integration with existing information systems such as Gopher, WAIS, FTP and other WWW servers. Hyper-G also offers a number of other features such as multi-lingual support, user access control and data storage services. The Hyper-G system is available as a commercial product known as HyperWave [Mau96].

B.13 MHEG

Some of the earlier discussions explored the idea of formal hypertext models, and showed how these could be used to reason about abstract hypertext ideas. These have led to the development of hypertext interchange standards which can be used to communicate hypertexts between heterogeneous systems. MHEG [MHEG97] is an international ISO standard developed by the Multimedia Hypermedia Experts Group which defines the representation and encoding of hypertext objects, for interchange between applications and services. Unlike many other approaches, the MHEG standard focuses on the runtime presentation of hypertext objects, and the interaction of the user with the hypertext.

MHEG assumes a minimal environment to ensure that the hypertext can be viewed on a variety of platforms. It makes a distinction between actual objects, and views of objects which are defined during the runtime presentation of the object. MHEG provides support for synchronisation, which allows the detailed coordination of hypermedia objects, and MHEG hypertext links provide references to external objects, which are interpreted appropriately by the application. MHEG also separates the contents of objects from the applications and defines anchors as presentation objects, and although the MHEG standard has been developed independently from the Dexter reference model, the two models are very similar.

The MHEG standard provides a means of communicating hypertext presentations between diverse platforms, and has a very simple notion of hypertext links. This contrasts with other approaches such as HyTime, described in the follwing section, which provides a richer environment aimed at the long-term storage of more complex hypertexts. In this way, it is hoped that presentational standards such as MHEG can complement more expressive models such as HyTime [RvOHB97].

B.14 Hypermedia/Time-based Structuring Language (HyTime)

HyTime [GNKN97] is another ISO standard designed for representing the structure and relationships of multimedia, hypertext and time-based documents. HyTime

does not address any of the issues arising from user interfaces, query languages or data notations, but does provide a way of describing document structures and the relationships between different parts of documents. Unlike the MHEG standard which focuses on the more presentational issues, HyTime provides a more expressive model for storing hypertext structures. The HyTime standard provides a very flexible means of addressing and locating objects in the hypertext, which allows the definition of complex hyperlinks. For example, HyTime allows the addressing of objects by specifying an offset from an object, by specifying a position in a hierarchical structure, or by using a query to specify the required object etc. HyTime also supports linking between documents, the specification of spatial information, and the co-ordination and manipulation of temporal data.

The HyTime standard developed from work done with the *Standard Generalised Markup Language* (SGML) [SGML85] used for describing the generic structure of documents. Document structures are defined using a Document Type Definition (DTD) which describe the elements in a hypertext, and how they relate to each other. HyTime is defined as a series of modular "architectural forms", which allows systems to support only those modules which are required for a particular application. The standard defines five HyTime modules (see figure B.14):

1. base module

   The HyTime "base module" provides the fundamental facilities for representing and addressing objects and must be supported by all HyTime applications.

2. location address module

   This defines the different kinds of addressing mechanisms supported by the HyTime standard, and allows the referencing of objects using some arbitrary measurement scheme. For example, an object address might use offsets of letters in a textual element, or a path through a hierarchical tree. The module also provides addressing using semantic constructs, which requires the interpretation of data objects by the application, for example "the red square in the middle".

3. hyperlink module

   This is used to support the notion of hypertext linking and to establish relationships between objects. The module defines a number of forms for defining arbitrary link types – hyperlink, contextual, aggregate, variable and independent. In this way, HyTime can support a diverse range of flexible linking mechanisms.

4. scheduling module

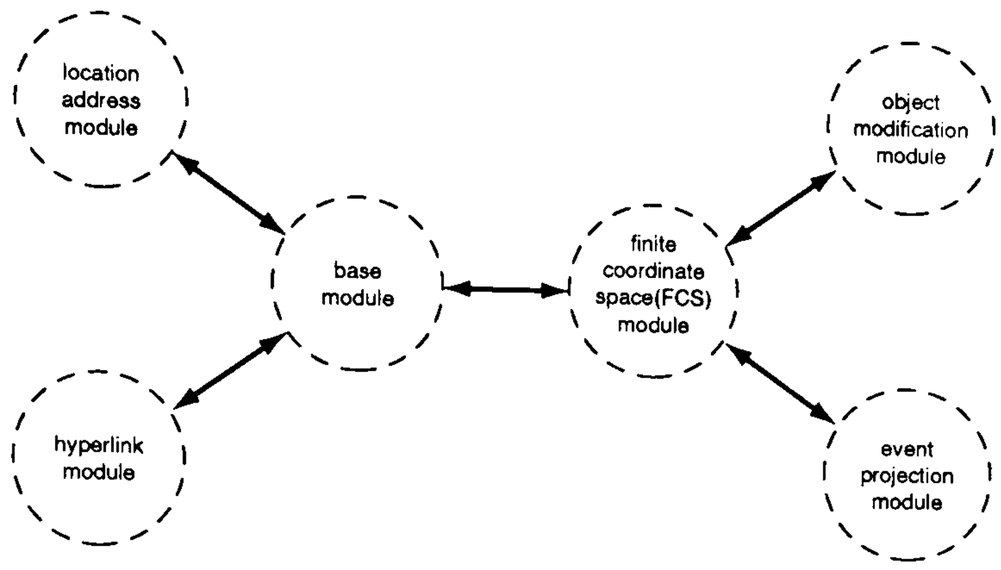


Figure B.9: HyTime modular architecture

This optional module is used to schedule objects, and define their positions relative to other objects. Objects are projected on to the axes of *finite coordinate spaces*, and measurements can be in terms of spatial or temporal units. The module defines a number standard units of measurement.

5. rendition module

   This final optional module can be used in conjunction with the scheduling module, to express *object modification* and *event projection*. Object modification allows changes to objects and the processing of information objects to be controlled during rendition (although HyTime does not define the semantics of these changes). Event projection is used to map objects between different coordinate spaces (for example, projecting an object in a virtual time space on to a real time space).

HyTime provides a flexible means of describing hypertext documents, and offers a useful standard for hypertext interchange. There have been few implementations of HyTime technologies, and developer's have been reluctant to adopt the HyTime standard. This is due in part, to the complexity of the standard and the difficulties of implementing HyTime engines. However, the HyTime model does offer some interesting and powerful modelling ideas, and HyTime may prove to be useful in the development of open hypertext systems.

B.15 Portable Document Format

The Portable Document Format (PDF)[BC93] has been developed by Adobe Systems, as a page description language for exchanging documents across platforms. This enables users to share traditional electronic documents with no knowledge of the resources and applications available to the recipient. Users create and browse PDF documents using the Acrobat suite of applications (figure B.15), and generate PDF files from existing legacy PostScript[Ado90] formats or directly from the native

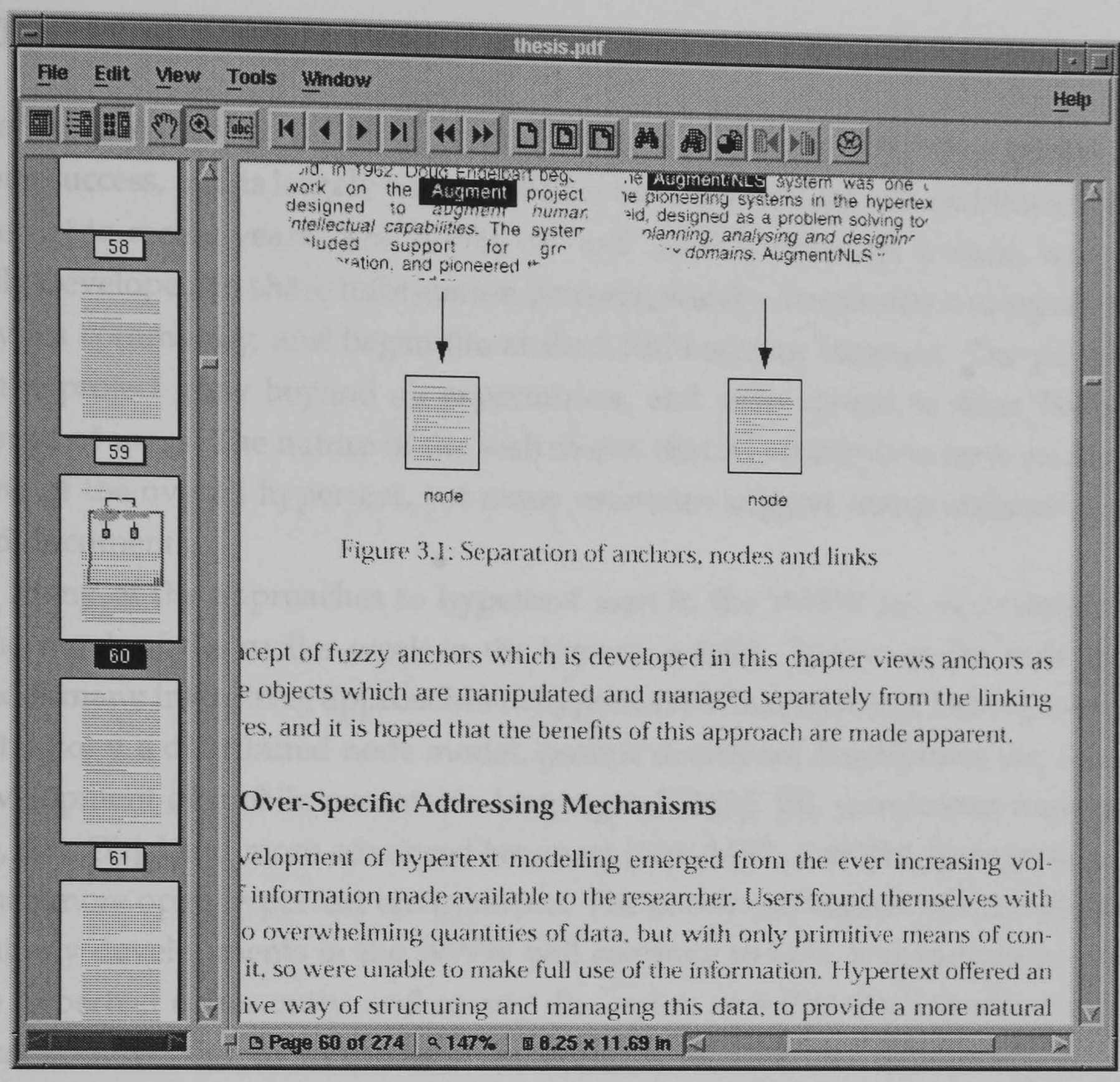


Figure B.10: Using Acrobat Exchange to view PDF documents

applications used to create the original document. The PDF format is derived from the PostScript page description language, and adopts a similar syntax and imaging model.

PDF is designed as a means of exchanging electronic documents between heterogenous platforms, although the standard does support some simple hypertext functionality. Users can connect specific areas of a page to other parts of a document using uni-directional links, and documents can be interfaced with the World Wide Web. Applications which support the PDF format have been developed to operate seamlessly in a WWW environment, and the PDF format is becoming increasingly popular as a platform for document interchange. While the Portable Document Format offers only very simple hypertext behaviour, the success of the technology may mean that it will play an important rôle in the development of hypertext interchange standards.

B.16 World Wide Web

The World Wide Web [WWWa] is a hypertext environment which has gained enormous success, and is largely responsible for the explosive growth and interest in the Internet in recent years. The WWW or *"web"* as it is variously known, was originally developed to share information between widely distributed colleagues in the physics community, and began life at the CERN science institute. The popularity of the project grew beyond all expectations, and soon spread to other fields and groups of users. The nature of the web means that it is difficult to form an accurate view of the overall hypertext, yet many estimates suggest many millions of users and documents.

Many of the approaches to hypertext seen in the WWW are very simple, and echo much of the earlier work in the hypertext field. However, the web also includes many interesting approaches to hypertext such as dynamic linking using CGI technology, a distributed node model, generic document descriptions etc. Also, the development of mobile executable languages [GM95, JS], component frameworks [JB, Act, Objc] and more advanced browsers [Net, MSE, Sun95b], have resulted in a much more open hypertext environment. The enormous success of the web, means that any developments in the WWW will continue to have a significant impact on the hypertext community, and so any discussion of influential hypertext research must acknowledge the presence of the World Wide Web.

The WWW was designed as a distributed hypertext system, based around a simple client-server model. Users request hypertext nodes from a remote document server, using a simple client known as a *web browser*. Documents are uniquely identified in the network domain using a *Universal Resource Locator* (URL) which provides a fixed, unambiguous address for each web document. Hypertext nodes are defined using a generic markup language called HTML (*Hypertext Markup Language*), which describes the content and node structure in a platform specific way (figure B.16). This simple idea allows WWW documents to be processed and viewed on a diverse range of platforms, and is one of the many reasons for its success. The distributed aspects of the web are also discussed in more detail in section 2.2.

The web supports a simple linking model, in which links are embedded in the node contents. Hyperlinks are uni-directional, but can reference elements either in the same documents, or remote objects which reside elsewhere in the network domain. The first WWW browsers such as *Lynx* supported simple textual nodes, but the emergence of graphical browsers such as Netscape Navigator [Net], Microsoft Internet Explorer [MSE] and HotJava [Sun95b] support an increasing range of graphics formats B.16. These web browsers also support simple navigational tools such as backtracking features and bookmark lists to record places of interest.

It is important to note some of the more open aspects of the World Wide Web

```
<title>
This is a HTML document
</title>

<body>
<h1>This is a main heading</h2>

<p>
A HTML document uses markup tags
to define each logical element.
</p>

<p>
The document can also include
hypertext links.

<a href="http://www.ep.cs.nott.ac.uk>
This is a link.
</a>
</p>
```

Figure B.11: An example HTML definition

model, which have helped integrate the web with the user community. One of the most important contributions was the ability to support diverse media types, by using *helper applications*. Many browsers would provide native support for textual nodes and some simple images, but if an unknown media type was encountered, then this would be passed to the appropriate application. This third party would be responsible for interpreting the node data, which allows the web to support many node formats. In addition to helper applications, the WWW was also designed around existing standards and tools. Unlike many other approaches to open hypertext, the web does not require a complex hyperbase or database service to manage the storage of objects. While this can limit the functionality of the system, it greatly simplifies the task of adding new users and documents into the WWW. Users need only a simple web browser to access web documents, and authors can publish documents by running a web server. The World Wide Web can accomodate new users and authors with the minimum amount of effort and cost, and it is this which is perhaps the main reason for the phenomenal success of the web.

The WWW also introduces the ideas of computation into the hypertext model, which were discussed earlier in section 2.1.3 when the ideas of open hypertext were outlined. The initial approach taken by web developers was to use the *Common Gateway Interface* to extend the functionality of the web. Authors could provide scripts and programs which could be indirectly attached to links, and executed by users. This simple form of computation was widely used in the web to achieve diverse tasks – dynamic links, submit queries, provide user information or any number of other computations. However, the limitations of this approach which were restricted to the simple exchange of data, have led to the development of richer execution models. Some web browsers offer their own scripting languages (eg.

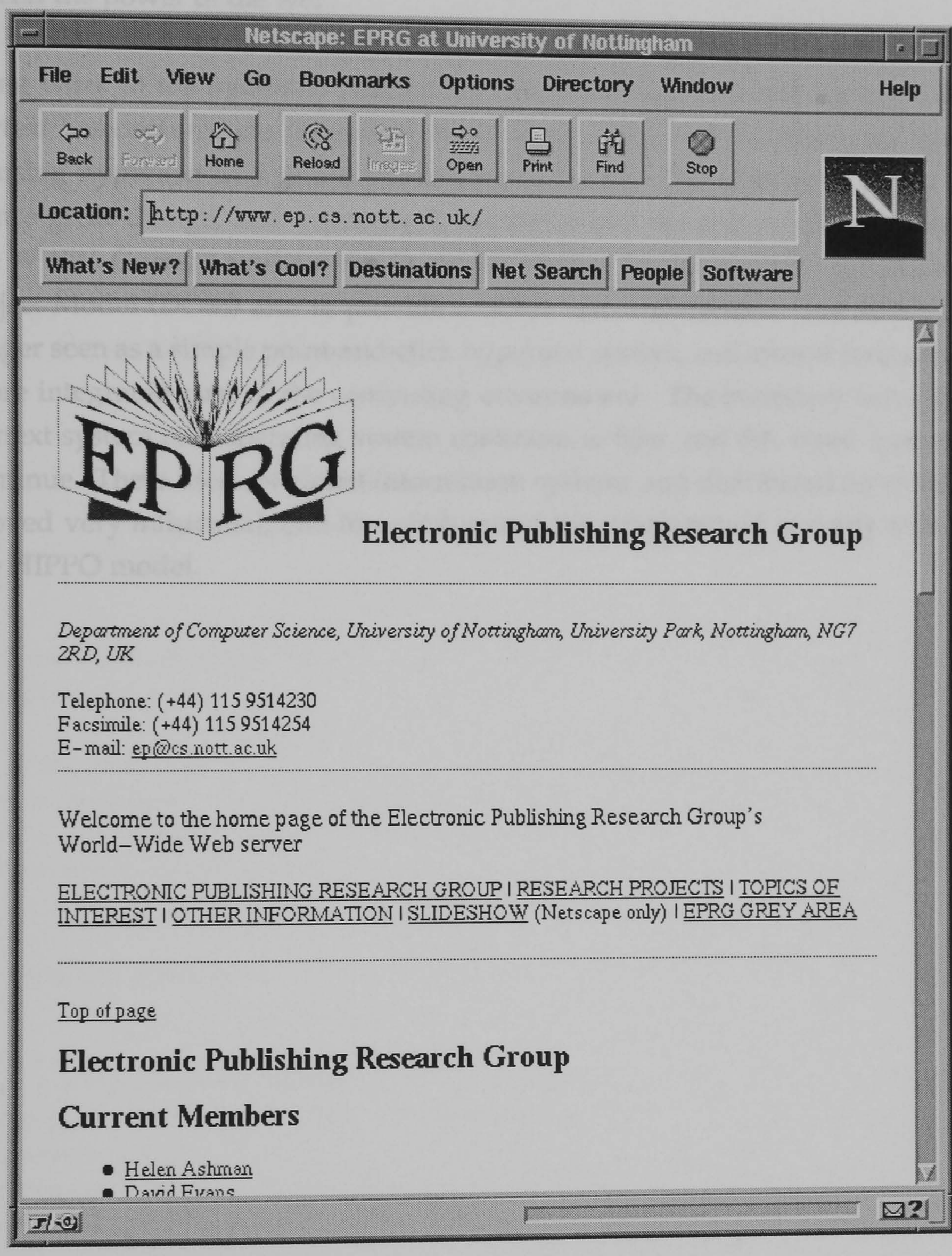


Figure B.12: Netscape Navigator web browser

JavaScript [JS]), which can respond to user actions and events. The Java language [GM95] offers a more powerful programming language which operates inside a *virtual machine*, and can execute on heterogeneous platforms. Also, the development of component architectures such as Java Beans [JB] and Active X [Act] continues to extend the power of the web.

The WWW offers a very simple hypertext model, and ignores much of the existing work in the hypertext community. However, this minimal approach to hypertext has captured the interest of the IT community, and has shown the benefits of using hypertext in a global information network. The web continues to experience great change, and recent work has developed the computational aspects of the WWW. Current work on the *Extensible Markup Language (XML) and* Document Object Model (DOM) aim to provide a richer web architecture. The WWW is no longer seen as a simple point-and-click hypertext system, and instead forms a much more integrated part of the computing environment. The boundary between hypertext system and operating system continues to blur, and this trend looks set to continue. These ideas of global information systems and distributed services have proved very influential, and have influenced the development of many aspects of the HIPPO model.

Appendix C

Fuzzy Anchor Specification

C.1 Lexical Specification

```
%{
/*
 * flex rules for fuzzy anchors
 *
 * Paul Newton
 */

/* marks end of input */
int flex_eof = 0;

/* error recovery */
static tokenpos = 0;
static int lineno = 1;
static char linebuf[500];
%}

%s NEED_ARG NEED_DATA

%%
\n.* { /* subsequent lines */
  strcpy(linebuf, yytext+1); /* save next line */
  lineno++;
  tokenpos = 0;
  yyless(1); /* pass back everything except \n */
 }

 [\t ] /* ignore whitespace */;

 id                    { BEGIN NEED_ARG; return ID; }
 page                  { BEGIN NEED_ARG; return PAGE; }
 dest                  { BEGIN NEED_ARG; return DEST; }
 data                  { return DATA; }
 xres                  { BEGIN NEED_ARG; return XRES; }
```

```
yres                    { BEGIN NEED_ARG; return YRES; }
range                   { BEGIN NEED_ARG; return RANGE; }
--                      { BEGIN 0; return EOR; }
=                       { return yytext[0]; }
<NEED_ARG>[^ ^\n]*      { BEGIN 0;
                          yylval.sym = strdup(yytext);
                          return STRING;
                        }
[0-9]+                  { yylval.num = atoi(yytext);
                          return INTEGER;
                        }
<<EOF>>                 { flex_eof = 1; return THE_END; }
```

C.2 Grammar

```
%{
/*
 * yacc grammar for fuzzy anchors
 *
 * pkn, 10/2/97
 */

/* current anchor */
static FAD* yyNewFAD = NULL;
%}

/* current token */
%union {
  char *sym;
  int num;
}
%token <sym> STRING
%token <num> INTEGER
%token ID PAGE DEST RES DATA XRES YRES RANGE THE_END EOR

%start entry
%%

entry:      THE_END { yyNewFAD = NULL; YYACCEPT; }
          | id page dest resolution range data {
              YYACCEPT;
            };

id:         ID '=' STRING {
              yyNewFAD = new FAD;
              yyNewFAD->FADsetID(atoi($3));
              /* reset indices */
              addAnotherElem(NULL, 0);
            }
```

```
page:        PAGE '=' STRING {
               yyNewFAD->FADsetPage(atoi($3));
             }

dest:        DEST '=' STRING {
               yyNewFAD->FADsetDest($3);
             }

resolution: xres yres

xres:        XRES '=' STRING {
               yyNewFAD->FADsetXres(atoi($3));
             }

yres:        YRES '=' STRING {
               yyNewFAD->FADsetYres(atoi($3));
             }

range:       RANGE '=' STRING {
               yyNewFAD->FADsetRange(atoi($3));
             }

data:        DATA { yyNewFAD->FADallocData(); } data_elem

data_elem:   INTEGER {
               addAnotherElem(yyNewFAD, $1);
             }
           | data_elem INTEGER {
               addAnotherElem(yyNewFAD, $2);
             }
```

Appendix D

Linkbase Specification

D.1 Lexical Specification

```
%{
/*
 * flex rules for linkbases
 *
 * Paul Newton
 */

/* marks end of input */
int LTlb_flex_eof = 0;

/* error recovery */
static tokenpos = 0;
static int lineno = 1;
static char linebuf[500];
%}

%s NEED_ARG NEED_DATA

%%
\n.* { /* subsequent lines */
  strcpy(linebuf, LTlb_yytext+1); /* save next line */
  lineno++;
  tokenpos = 0;
  yyless(1); /* pass back everything except \n */
}

[\t ] /* ignore whitespace */;

linkbase          { BEGIN NEED_ARG; return LINKBASE; }
strength          { BEGIN NEED_ARG; return LINK_STRENGTH; }
srcDoc            { BEGIN NEED_ARG; return SRC_DOC; }
srcAnchor         { BEGIN NEED_ARG; return SRC_ANCHOR; }
destDoc           { BEGIN NEED_ARG; return DEST_DOC; }
```

```
destAnchor          { BEGIN NEED_ARG; return DEST_ANCHOR; }
--                  { BEGIN 0; return EOR; }
=                   { return LTlb_yytext[0]; }
<NEED_ARG>[^ ^\n]*  { BEGIN 0;
                      LTlb_yylval.sym = strdup(yytext);
                      return STRING;
                    }
<<EOF>>             { LTlb_flex_eof = 1; return THE_END; }
```

D.2 Grammar

```
%{
/*
 * yacc grammar for linkbases
 *
 * pkn, 3/12/96
 */

/* current link */
static LTlink *yyNewLink = NULL;
%}

/* current token */
%union {
  char *sym;
  int num;
}
%token <sym> STRING
%token LINKBASE LINK_STRENGTH SRC_DOC SRC_ANCHOR
%token DEST_DOC DEST_ANCHOR THE_END EOR

%start entry
%%

entry:      THE_END { yyNewLink = NULL; YYACCEPT; }
          | start linkbase link_strength
            src_doc src_anchor dest_doc dest_anchor {
              YYACCEPT;
            };

start:        EOR {
                yyNewLink = new LTlink;
              }

linkbase:     /* empty */
            | LINKBASE '=' STRING {
                yyNewLink->setLinkbase($3);
                free($3);
              };
```

```
link_strength:  /* empty */
              | LINK_STRENGTH '=' STRING {
                  yyNewLink->setLinkStrength(atof($3));
                  free($3);
                }

src_doc:        /* empty */
              | SRC_DOC '=' STRING {
                  yyNewLink->setSrcDoc($3);
                };

src_anchor:     SRC_ANCHOR '=' STRING {
                  yyNewLink->setSrcAnchor(atoi($3));
                };

dest_doc:       /* empty */
              | DEST_DOC '=' STRING {
                  yyNewLink->setDestDoc($3);
                };

dest_anchor:    /* empty */
              | DEST_ANCHOR '=' STRING {
                  yyNewLink->setDestAnchor(atoi($3));
                };
```

Appendix E

Linkbase Tree Hierarchy Specification

E.1 Lexical Specification

```
%{
/*
 * lexer for linkbase inheritance tool
 *
 * Paul Newton
 */

/* error recovery */
static int lineno = 0;
%}

%%
#.* { lineno++; }; /* comments */

linkbase            { return LINKBASE; }
public|private      {
            LTinherit_yylval.sym = strdup(LTinherit_yytext);
            return SPECIFIER;
                    }
url                 { return URL; }
filename            { return FILENAME; }
build               { return BUILD; }
[(){}:;,]           { return LTinherit_yytext[0]; }
[-:~./_a-zA-Z0-9]*[/_a-zA-Z0-9]  {
            LTinherit_yylval.sym = strdup(LTinherit_yytext);
            return ID;
                                }
[\t ]               ; /* ignore whitespace */
\n                  lineno++;
```

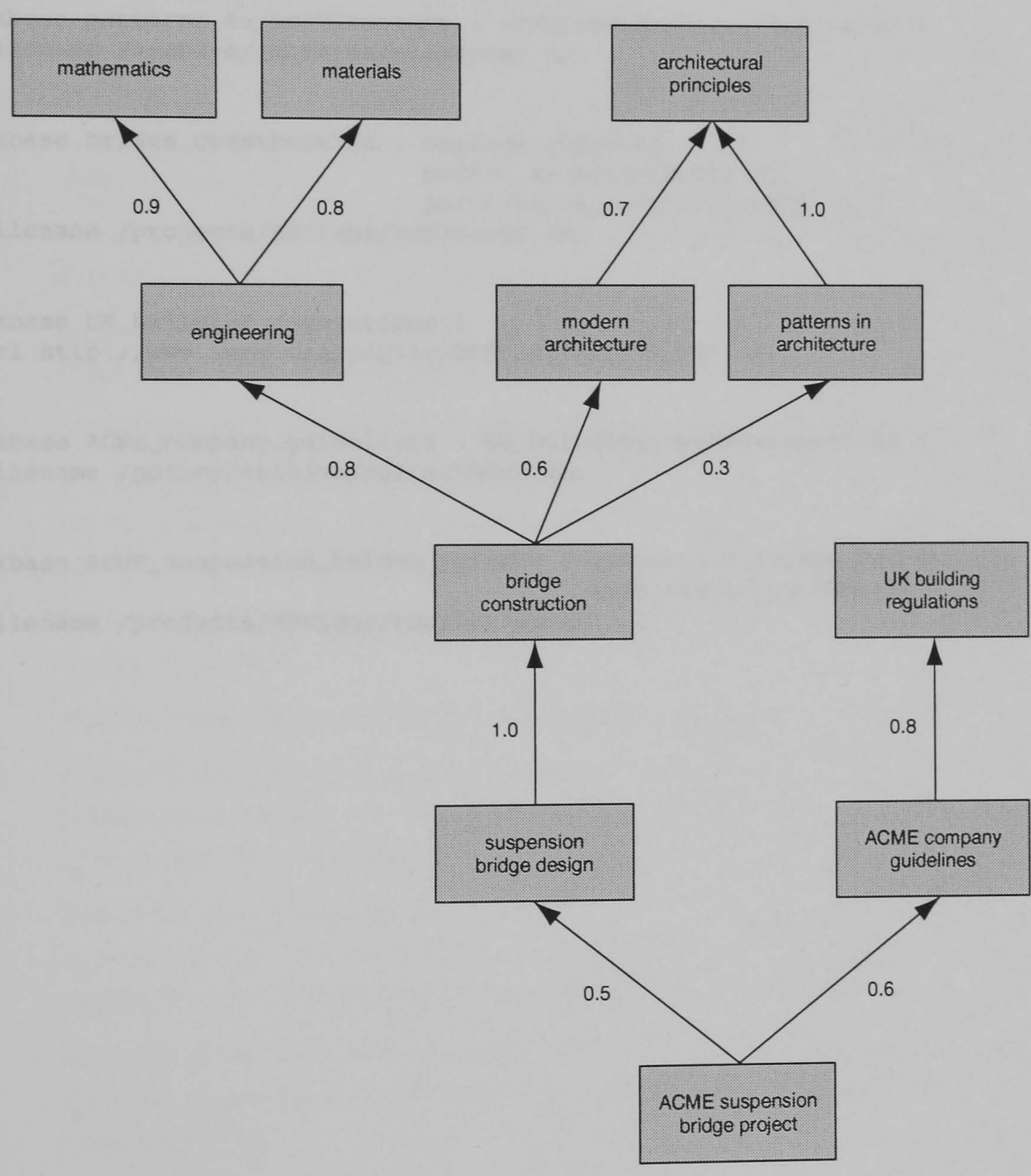
E.2 Grammar

```
%{
/*
 * parser for linkbase inheritance tools
 *
 * pkn, 25/10/96
 */

/* current tree and tree node */
static LTtree yyLTtree;
static LTtreeE *yyNewNode;
%}

/* current token */
%union {
  char *sym;
}

%token <sym> ID SPECIFIER LB_REF
%token LINKBASE FILENAME URL BUILD

%%
linkLTtree:       /* empty */
                | linkLTtree linknode
                | linkLTtree buildnode

linknode:         newLinkbase ':' inheritSpec contents ';'
                | newLinkbase contents ';'

newLinkbase:      LINKBASE ID
                  {
                    yyNewNode = new LTtreeE($2);
                    free($2);
                    yyLTtree.addNode(yyNewNode);
                  }

inheritSpec:      SPECIFIER ID
                  {
                    yyNewNode->addParent(yyLTtree.getNode($2),
                                         BaseAccess($1));
                    free($1);
                    free($2);
                  }
                | SPECIFIER ID '(' ID ')'
                  {
                    yyNewNode->addParent(yyLTtree.getNode($2),
                                         BaseAccess($1),
                                         atof($4));
                    free($1);
```

```
                    free($2);
                    free($4);
                  }
                | inheritSpec ',' SPECIFIER ID
                  {
                    yyNewNode->addParent(yyLTtree.getNode($4),
                                         BaseAccess($3));
                    free($3);
                    free($4);
                  }
                | inheritSpec ',' SPECIFIER ID '(' ID ')'
                  {
                    yyNewNode->addParent(yyLTtree.getNode($4),
                                         BaseAccess($3),
                                         atof($6));
                    free($3);
                    free($4);
                    free($6);
                  }

contents:         '{' lbLoc '}'

lbLoc:            FILENAME ID ';'
                  {
                    yyNewNode->setFilename($2);
                    free($2);
                  }
                | URL ID ';'
                  {
                    yyNewNode->setURL($2);
                    free($2);
                  }

buildnode:        BUILD ID ';'
                  {
                    LTtreeE *node = yyLTtree.getNode($2);
                    yyLTtree.buildTree(node);
                    free($2);
                  }
                | BUILD ID ID ';'
                  {
                    LTtreeE *node = yyLTtree.getNode($2);
                    yyLTtree.buildTree(node, $3);
                    free($2); free($3);
                  }
```

Appendix F

Linkbase Tree Example



Appendix F: Linkbase Tree Example

```
linkbase mathematics {
  url http://www.ucl.ac.uk/public/appliedMathII.lb;
};

linkbase materials {
  url http://www.british-library.org/lb/science/ref/metalProperties.lb;
};

linkbase engineering : mathematics(0.9), materials(0.8) {
  url http://www.acme.com/local/acmeEng.lb;
};

linkbase architectural_principles {
  url http://www.tekon.org/basicArch.lb;
};

linkbase modern_architecture: architectural_principles(0.7) {
  url http://www.british-library/org/lb/history/ref/modernArch.lb;
};

linkbase patterns_in_architecture : architectural_principles(1) {
  filename /archive/patterns/alexander.lb;
};

linkbase bridge_construction : engineering(0.8),
                               modern_architecture(0.6),
                               patterns_in_architecture(0.3) {
  filename /projects/EBridge/ref/suspB.lb;
};

linkbase UK_building_regulations {
  url http://www.ieee.org/public/Oct93/12AB9/33_3XA.lb;
};

linkbase ACME_company_guidelines : UK_building_regulations(0.8) {
  filename /policy/archive/currentACME.lb;
};

linkbase ACME_suspension_bridge_project : suspension_bridge_design(.5),
                                          ACME_company_guidelines(.6) {
  filename /projects/EBridge/current/EB.lb;
};
```

Bibliography

[ABC+97] Sharon Adler, Anders Berglund, James Clark, Istvan Cseri, Paul Grosso, Jonathan Marsh, Gavin Nicol, Jean Paoli, David Schach, Henry S. Thompson, and Chris Wilson. A Proposal for XSL, August 1997. Work in progress. The latest documentation is available from the W3C WWW site (URL: http://www.w3.org).

[ACDC96] Helen Ashman, Tim Cawley, Scott Davis, and Greg Chase. Issues in the Use of External and Remote Services in Hypermedia Systems. In *Workshop on Incorporating Hypertext Functionality into Software Systems (HTFII)*, 1996. Held in conjunction with Hypertext 1996 Conference.

[Acr] Documentation on the Acrobat suite of software products is available from the Adobe WWW site (URL: http://www.adobe.com).

[Act] ActiveX and DCOM documentation available from the Microsoft WWW site (URL: http://www.microsoft.com).

[Ado90] Adobe Systems, Inc. *PostScript Language Reference Manual*, 2nd edition, 1990.

[AKM95] Keith Andrews, Frank Kappe, and Hermann Maurer. Hyper-G and Harmony: Towards the Next Generation of Networked Inofrmation Technology. In *Proceedings of CHI '95*, ACM, Denver, May 1995.

[Ale77] Christopher Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[Alt] The AltaVista WWW site. http://www.altavista.digital.com.

[AM84a] Robert M. Akscyn and Donald L. McCacken. ZOG and the USS Carl Vinson: Lessons in System Development. In *Proceedings of the First IFIP Conference on Human-Computer Interaction*, pages 901–906, London, England, September 1984. Elsevier Science Publishers.

[AM84b] Robert M. Akscyn and Donald L. McCracken. The ZOG Approach to Database Management. In *Proceedings of the Trends and Applications Conference: Making Databases Work*, pages 217–225, Gaithersburg, Maryland, May 1984.

[AM93] Robert M. Akscyn and Donald L. McCracken. Design of Hypermedia script languages: the KMS experience. In *Proceedings of the 1993 Hypertext Conference*, pages 268–269, November 1993.

[AMY88] Robert M. Akscyn, Donald L. McCracken, and Elise A. Yoder. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*, 31(7):820–835, July 1988.

[And97] Kenneth M. Anderson. A Critique of the Open Hypermedia Protocol. In *Proceedings of the 3rd Workshop on Open Hypermedia Systems*, 1997. Held in conjunction with Hypertext 1997 Conference.

[App87] Apple Computer Inc. *Macintosh HyperCard User's Guide*, 1987.

[ATJ94] Kenneth M. Anderson, Richard N. Taylor, and E. James Whitehead Jr. Chimera: Hypertext for Heterogeneous Software Environments. In *Proceedings of the European Conference on Hypertext (ECHT '94)*, pages 94–107, Milano, September 1994.

[AV94] H.L. Ashman and J.L.M. Verbyla. Dynamic Link Management Via the Functional Model of the Link. In *Proceedings of the Basque International Workshop on Information Technology*, Feb 1994.

[AVC94] H.L. Ashman, J.L.M. Verbyla, and T. Cawley. Hypermedia Management in Large-Scale Information Systems Using the Functional Model of the Link. In *Proceedings of the Fifth Australasian Database Conference*, pages 247–257, Jan 1994.

[BC93] Tim Bienz and Richard Cohn. *Portable Document Format Reference Manual*. Adobe Systems Incorporated, Mountain View, California, June 1993.

[BE94] C. Boyle and A.O. Encarnacion. MetaDoc: an adaptive hypertext reading system. *User Models and User Adapted Interaction*, 4(1):1–19, 1994.

[Bea94] I. Beaumont. User modeling in the interactive anatomy tutoring system ANATOM-TUTOR. *User Models and User Adapted Interaction*, 4(1):21–45, 1994.

[Ber88] M. Bernstein. The bookmark and the compass: Orientation tools for hypertext users. *ACM SIGOIS Bulletin*, 9(4):34–45, October 1988.

[Bie91] Michael Bieber. Issues in modeling a dynamic hypertext interface for non-hypertext systems. In *Proceedings of the 1991 Hypertext Conference*, pages 203–217, December 1991.

[BJ87] Jay Bolter and Michael Joyce. Hypertext and Creative Writing. In *Proceedings of Hypertext 1987*, pages 41–50. ACM, November 1987.

[Bol91] J.D. Bolter. *Writing Space: The Computer, Hypertext, and the History of Writing*. Lawrence Erlbaum Associates, 1991.

[Boo94] Grady Booch. *Object-Oriented analysis and design*. Benjamin-Cummings, 1994.

[Bro88] P.J Brown. Linking And Searching Within Hypertext. *Electronic Publishing: Origination, Dissemination and Design*, 1(1):45–53, April 1988.

[Bro89] P.J. Brown. Do We Need Maps To Navigate Round Hypertext Documents? *Electronic Publishing: Origination, Dissemination and Design*, 2(2):91–100, July 1989.

[Bro92] P.J Brown. UNIX Guide: Lessons From Ten Years' Development. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 63–70, Milano, November 1992. ACM.

[Bro94] Peter Brown. Adding Networking to Hypertext: Can it be done transparently? In *Proceedings of the European Conference on Hypertext (ECHT '94)*, pages 51–58, September 1994.

[Bru96] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modelling and User Adapted Interaction*, 6(2-3), 1996.

[BSW96] P. Brusilovsky, E. Schwarz, and G. Weber. ELM-ART: An intelligent tutoring system on World Wide Web. In *Third International Conference on Intelligent Tutoring Systems (ITS '96)*, 1996.

[Buf96] John F. Buford. Evaluating HyTime: an examination of implementation experience. In *Proceedings of the 7th ACM Conference on Hypertext*, pages 105–115. ACM, March 1996.

[Bus45] Vannevar Bush. As We May Think. *Atlantic Monthly*, July 1945.

[BWAH96] Ajit Bapat, Jürgen Wäsch, Karl Aberer, and Jörg M. Haake. HyperStorM: An Extensible Object-Oriented Hypermedia Engine. In *Proceedings of Hypertext 1996*, pages 203–214. ACM, March 1996.

[CB89] Jeff Conklin and Michael L. Begeman. gIBIS: A tool for all reasons. *Journal of the American Society for Information Science*, 40(3):200–213, 1989.

[CBDW94] L.A. Carr, D.W. Barron, H.C. Davis, and W.Hall. Why use HyTime? *Electronic Publishing: Origination, Dissemination and Design*, 7(3):163–178, September 1994.

[CBY89] Timothy Catlin, Paulette Bush, and Nicole Yankelovich. InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration. In *Proceedings of the 1989 Hypertext Conference*, pages 365–378, November 1989.

[CCS92] Silvano Pozzi Cefriel, Augusto Celentano, and Luisa Salemme. ALIVE: A Distributed Live-Link Documentation System. *Electronic Publishing: Origination, Dissemination and Design*, 5(3):131–142, September 1992.

[CG88] Brad Campbell and Joseph M. Goodman. HAM: A General Purpose Hypertext Abstract Machine. *Communications Of The ACM*, 31(7):856–861, July 1988.

[Cha93] Daniel T. Chang. HieNet: A User-Centered Approach for Automatic Link Generation. In *Proceedings of the 1993 Hypertext Conference*, pages 145–158, November 1993.

[CHP88] Donald D. Chamberlin, Helmut F. Hasselmeier, and Dieter P. Paris. Defining Document Styles for WYSIWYG Processing. In *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography (EP '88)*, pages 121–137, April 1988.

[Con87] Jeff Conklin. Hypertext: An Introduction And Survey. *Computer*, pages 17–40, September 1987.

[CS88] T.J.O Catlin and K.E Smith. Anchors For Shifting Tides: Designing a Seaworthy Hypermedia System. In *Proceedings of the 12th International Online Meeting*, pages 15–25, London, 1988.

[CT91] Marco A. Casanova and Luiz Tucherman. The Nested Context Model for Hyperdocuments. In *Proceedings of the 1991 Hypertext Conference*, pages 193–201, December 1991.

[dBH92] Paul de Bra and Geert-Jan Houben. An Extensible Data Model for Hyperdocuments. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 222–231, Milano, November 1992.

[DCE] Open Group. DCE documentation available from the Open Group WWW site (URL: http://www.opengroup.org).

[DeR89] Steven J. DeRose. Expanding the Notion of Links. In *Proceedings of the Hypertext 1989 Conference*, pages 249–257, November 1989.

[DES93] National Institute of Standards and Technology. Data Encryption Standard (DES). Federal Information Processing Standards publication, December 1993.

[DHHH92] Hugh Davis, Wendy Hall, Ian Heath, and Gary Hill. Towards an Integrated Information Environment with Open Hypermedia Systems. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 181–190. ACM, December 1992.

[Dij68] E.W. Dijkstra. Goto statement considered harmful. *Communications of the ACM*, 11(3):147–148, March 1968.

[DKH94] Hugh C. Davis, Simon Knight, and Wendy Hall. Light Hypermedia Link Services: A Study of Third Party Application Integration. In *Proceedings of the 1994 ECHT Conference*, pages 41–50. ACM, September 1994.

[DLR96] Hugh Davis, Andy Lewis, and Antoine Rizk. OHP: A Draft Proposal for a Standard Open Hypermedia Protocol (Levels 0 and 1: Revision 1.2). In *Proceedings of the 2nd Workshop on Open Hypermedia Systems*, 1996. Held in conjunction with Hypertext 1996 Conference.

[dRCP93] F. de Rosis, B. De Carolis, and S. Pizzutilo. User tailored hypermedia explanations. In *INTERCHI '93 Adjunct Proceedings*, pages 169–170, 1993.

[DS86] Norman Delisle and Mayer Schwartz. Neptune: A Hypertext System For CAD Applications. In *Proceedings of the ACM SIGMOD International Conference On Management Of Data*, pages 132–143, Washington D.C, May 1986.

[DSSSL96] International Standards Organisation. Document Style Semantics and Specification Language (DSSSL). ISO/IEC IS 10179:1996, 1996.

[Eng84a] Douglas C. Engelbart. Authorship Provisions In Augment. In *Proceedings of the 1984 COMPCON Conference*, pages 465–472, San Francisco, February 1984. IEEE.

[Eng84b] Douglas C. Engelbart. Collaboration Support Provisions in Augment. In *Proceedings of the 1984 AFIPS Office Automation Conference*, pages 51–58, Los Angeles, February 1984.

[Eng95] Douglas C. Engelbart. Toward augmenting the human intellect and boosting our collective IQ. *Communications of the ACM*, 38(8):30–31, August 1995.

[ET94] Paul M. English and Raman Tenneti. Interleaf Active Documents. *Electronic Publishing: Origination, Dissemination and Design*, 7(2):75–87, June 1994.

[FC89] Mark E. Frisse and Steve B. Cousins. Information retrieval from hypertext: Update on the Dynamic Medical Handbook Project. In *Proceedings of Hypertext 1989*, pages 199–212. ACM, November 1989.

[FC91] Mark F. Frisse and Steve B. Cousins. Models for Hypertext. *Journal of the American Society for Information Science*, 43(2):183–191, 1991.

[Fei88] Steven Feiner. Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structure. In *Conference on Office Computing Systems*, pages 205–212, 1988.

[Fei90] Steven K. Feiner. Authoring Large Hypermedia Documents With IGD. *Electronic Publishing: Origination, Dissemination and Design*, 3(1):29–46, February 1990.

[FHHD90] Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh Davis. Microcosm: An Open Model for Hypermedia With Dynamic Linking. In *Hypertext: Concepts, Systems and Applications. The Proceeedings of the European Conference on Hypertext*, pages 298–311, France, 1990. Cambridge University Press.

[Foua] Free Software Foundation. The Bison general-purpose parser generator. Available from GNU software archives.

[Foub] Free Software Foundation. The Flex lexical analyzer generator. Available from GNU software archives.

[Fri87] Mark Edwin Frisse. Searching for Information in a Hypertext Medical Handbook. In *Proceedings of Hypertext 1987*, pages 57–66. ACM, November 1987.

[FS89] Richard Furuta and P. David Stotts. Programmable Browsing Semantics in Trellis. In *Proceedings of Hypertext 1989*, pages 27–42. ACM, November 1989.

[FS90] Richard Furuta and P. David Stotts. A functional meta-structure for hypertext models and systems. *Electronic Publishing: Origination, Dissemination and Design*, 3(4):179–205, November 1990.

[FSD92] Richard Furuta, P. David Stotts, and Gregory D. Drew. Experiences with a Client-Server-Based Architecture for a Distributed Structured Hypertext System. In *Proceedings of Electronic Publishing*, pages 113–125, Apr, 1992. Cambridge University Press.

[Fur86] G. Furnas. Generalized Fisheye Views. In *Proceedings of CHI'86, Human Factors in Computing Systems*, pages 16–23, April 1986.

[GB80] Ira P. Goldstein and Daniel G. Bobrow. A layered approach to software design. Technical Report CSL-80-5, Xerox PARC, Xerox Corporation, December 1980.

[GDHR97] Stuart Goose, Jonathan Dale, Wendy Hall, and David De Roure. Microcosm TNG: A Distributed Architecture to Support Reflexive Hypermedia Applications. In *Proceedings of Hypertext 1997*, pages 226–227. ACM, April 1997.

[GH95] M. Gonschorek and C. Herzog. Using hypertext for an adaptive help system in an intelligent tutoring system. In *7th World Conference on Artificial Intelligence in Education*, pages 274–281, 1995.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[GHMS93] Kaj Grønbæk, Jens A. Hem, Ole L. Madsen, and Lennert Sloth. Designing Dexter-Based Hypermedia Systems. In *Proceedings of the 1993 Hypertext Conference*, pages 25–38, November 1993.

[GHMS94] Kaj Grønbæk, Jens A. Hem, Ole L. Madsen, and Lennert Sloth. Hypermedia Systems: A Dexter-based Architecture. *Communications of the ACM*, 37(2):65–74, February 1994.

[GKM93] Kaj Grønbæk, Morten Kyng, and Preben Mogensen. CSCW Challenges: Cooperative Design In Engineering Projects. *Communications of the ACM*, 36(4):67–77, June 1993.

[GLD97] Stuart Goose, Andy Lewis, and Hugh Davis. OHRA: Towards an Open Hypermedia Reference Architecture and a Migration Path for Existing Systems. In *Proceedings of the 3rd Workshop on Open Hypermedia Systems*, 1997. Held in conjunction with Hypertext 1997 Conference.

[GM95] James Gosling and Henry McGilton. The Java Language Environment—A White Pape r. Technical report, Sun Microsystems, Mountain View, California, May 1995. Available from the JavaSoft WWW site
(URL: http://java.sun.com).

[GMP96] Franca Garzotto, Luca Mainetti, and Paolo Paolini. Information Reuse in Hypermedia Applications. In *Proceedings of Hypertext 1996*, pages 93–104. ACM, March 1996.

[GMP97] Franca Garzotto, Luca Mainetti, and Paolo Paolini. Designing Modal Hypermedia Applications. In *Proceedings of Hypertext 1997*, pages 38–47. ACM, April 1997.

[GNKN97] Charles F. Goldfarb, Steven R. Newcomb, W. Eliot Kimber, and Peter J. Newcomb. Information processing – Hypermedia/Time-based Structuring Language HyTime – 2nd edition. Technical Report ISO/IEC JTC 1/SC 18 WG8 N1920rev, International Standards Organisation (ISO), May 1997.

[Gol90] Charles Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[Gol97] Gene Golovchinsky. What the Query Told the Link: The Integration of Hypertext and Information Retrieval. In *Proceedings of Hypertext 1997*, pages 67–74. ACM, April 1997.

[GP93] Franca Garzotto and Paolo Paolini. HDM – A model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993.

[Gro94a] Kaj Grønbæk. Building tailorable hypermedia systems: the embedded-interpreter approach. In *Proceedings of Object Oriented Programming Systems, Languages and Applications (OOPSLA) 1986*, Aarhus University, October 1994. SIGPLAN.

[Gro94b] Kaj Grønbæk. Composites in a Dexter-based Hypermedia Framework. In *Proceedings of ECHT 1994*, pages 59–69. ACM, September 1994.

[Gru93] G. Grunst. Adaptive hypermedia. *Adaptive user interfaces: Principles and practice*, pages 269–283, 1993.

[GS87] Irene Greif and Sunil Sarin. Data Sharing in Group Work. *ACM Transactions On Office Information Systems*, 5(2):187–211, April 1987.

[GT94] Kaj Grønbæk and Randall H. Trigg. Design Issues for a Dexter-based Hypermedia System. *Communications of the ACM*, 37(2):41–49, February 1994.

[Haa92] A. Haake. CoVer: A contextual version server for hypertext applications. In *Proceedings of the European Conference on Hypertext (ECHT '92)*, pages 43–52. ACM, November 1992.

[Haa97] Jörg Haake. Colaboration via OHS: A Scenario. In *Proceedings of the 3rd Workshop on Open Hypermedia Systems*, 1997. Held in conjunction with Hypertext 1997 Conference.

[Hal87] Frank G. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. In *Proceedings of Hypertext 1987*, pages 345–365. ACM, November 1987.

[Hal94] Wendy Hall. Ending the tyranny of the button. *IEEE Multimedia*, 1(1), 1994.

[HBG96] H. Hohl, H.D. Böcker, and R. Gunzenhäuser. Hypadapter: An adaptive hypertext system for exploratory learning and programming. *User Models and User Adapted Interaction*, 6, 1996.

[HF92] Dan Heller and Paula M. Ferguson. *Motif Programming Manual*, volume 6A. O'Reilly and Associates, Inc., 1992.

[HGC94] A. Hatzimanikatis, I. Gaviotis, and D. Christodoulakis. Distributed Documents: An Architecture for Open Distributed Hypertext. *Electronic Publishing: Origination, Dissemination and Design*, 7(1):35–48, March 1994.

[HH94] Gary Hill and Wendy Hall. Extending the Microcosm Model to a Distributed Environment. In *Proceedings of the European Conference on Hypertext (ECHT '94)*, pages 32–40. ACM, September 1994.

[HHL+92] Jorg Haake, Jorg Hannemann, Andreas Lemke, Wolfgang Schuler, Helge Schutt, and Manfred Thuring. SEPIA: A Cooperative Hypermedia Authoring Environment. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 12–22, Milano, November 1992.

[Hic93] D. L. Hicks. *A version control architecture for advanced hypermedia environments*. PhD thesis, Department of Computer Science, Texas A & M University, 1993.

[HKr+96] K. Höök, J. Karlgren, A. Wærn, N. Dahlbäck, C.G. Jansson, K. Karlgren, and B. Lemaire. A glass box approach to adaptive hypermedia. *User Models and User Adapted Interaction*, 6, 1996.

[HKRC92] Bernard J. Haan, Paul Kahn, Victor A. Riley, and James H. Coombs. IRIS Hypermedia Services. *Communications Of The ACM*, 35(1):36–51, January 1992.

[HMT87] Frank G. Halasz, Thomas P. Moran, and Randall H. Trigg. Notecards In A Nutshell. In *Proceedings of Human Factors in Computing Systems*, pages 45–52. ACM, 1987.

[Hoe89] T. Hoeber. *The OPEN LOOK Graphical User Interface Style Guide*. Sun Microsystems, May 1989.

[HS90] Frank Halasz and Mayer Schwartz. The Dexter Hypertext Reference Model. Submitted to the NIST Hypertext Standardisation Workshop, Gaithersburg, January 1990.

[HTTP] Network Working Group. Hypertext Transfer Protocol – HTTP/1.1, January 1997. Request for Comments (RFC): 2068.

[IT89] Peggy M. Irish and Randall H. Trigg. Supporting Collaboration in Hypermedia: Issues and Experiences. *Journal of the American Society for Information Science*, 40(3):192–199, 1989.

[Joh75] S.C. Johnson. Yacc – Yet Another Compiler-Compiler. Technical Report CS Technical Report No. 32, Bell Laboratories, July 1975.

[JB] Sun Microsystems. JavaBeans API Specification v1.01. Available from the JavaSoft WWW site (URL: http://java.sun.com).

[JS] JavaScript documentation can be found on the Netscape Communications WWW site (URL: http://www.netscape.com).

[Kac90] Charles J. Kacmar. *PROXHY: A Process-oriented extensible hypertext architecture.* PhD thesis, Texas AM University, 1990.

[Kac95] Charles J. Kacmar. A process approach for providing hypermedia services to existing, non-hypermedia applications. *Electronic Publishing: Origination, Dissemination and Design*, 8(1):31–48, March 1995.

[KFC93] C. Kaplan, J. Fenwick, and J. Chen. Adaptive hypertext navigation based on user goals and context. *User Models and User Adapted Interaction*, 3(3):193–220, 1993.

[Kil94] Haim Kilov. On Understanding Hypertext: Are Links Essential? *Software Engineering Notes*, 19(1):30, January 1994.

[KL91] Charles J. Kacmar and John J. Leggett. PROXHY: A Process-Oriented Extensible Hypertext Architecture. *ACM Transactions on Information Systems*, 9(4):399–419, October 1991.

[KN93] J. Kohl and C. Neuman. The Kerberos Network Authentication Service v5, September 1993. Request for Comments (RFC): 1510.

[KWO90] Uffe Kock Wiil and Kasper Østerbye. Experiences with HyperBase – A multi-user backend for hypertext applications with emphasis on collaboration support. Technical Report R 90-38, Department of Computer Science, University of Aalborg, October 1990.

[Lan90] Danny Lange. A Formal Model of Hypertext. In *NIST Hypertext Standardisation Workshop*, pages 145–166, January 1990.

[Lan92] George P. Landow. *Hypertext: The Convergence of Contemporary Critical Theory and Technology*. Johns Hopkins University Press, 1992.

[Lel92] Alain Lelu. Hypertext paradigm in the field of information retrieval: a neural approach. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 112–121. ACM, December 1992.

[Les75] M.E. Lesk. Lex – A Lexical Analyzer Generator. Technical Report CS Technical Report No. 39, Bell Laboratories, October 1975.

[Lis87] Barbara Liskov. Data Abstraction and Hierarchy. In *Proceedings of Object Oriented Programming Systems, Languages and Applications (OOPSLA) 1987 (Addendum)*, MIT Laboratory of Computer Science, October 1987. SIGPLAN.

[LK91] John J. Leggett and Ronnie L. Killough. Issues in Hypertext Interchange. *Hypermedia*, 3(3):159–186, 1991.

[LS93] Daryl T. Lawton and Ian E. Smith. The Knowledge Weasel Hypermedia Authoring System. In *Proceedings of the 1993 Hypertext Conference*, pages 106–117, November 1993.

[LS94] John J. Leggett and John L. Schnase. Dexter With Open Eyes. *Communications of the ACM*, 37(2):77–86, February 1994.

[Mal91] Kathryn C. Malcolm. Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise. In *Proceedings of the 1991 Hypertext Conference*, pages 13–23, December 1991.

[Mau96] Hermann Maurer. *HyperWave – The Next Generation Web Solution*. Addison-Wesley, 1996.

[MBD+90] Raymond J. McCall, Patrick R. Bennett, Peter S. D'Oronzio, Jonathan L. Ostwald, Frank M. Shipman III, and Nathan F. Wallace. PHIDIAS: Integrating CAD Graphics into Dynamic Hypertext. In *Hypertext: Concepts, Systems and Applications. The Proceeedings of the European Conference on Hypertext*, pages 152–165, France, 1990. Cambridge University Press.

[MC93] Susan Michalak and Mary Coney. Hypertext and the Author/Reader Dialogue. In *Proceedings of the 1993 Hypertext Conference*, pages 174–182, November 1993.

[MC94] N. Mathé and J. Chen. A user-centred approach to adaptive hypertext based on an information relevance model. In *4th International Conference on User Modeling*, pages 107–114, 1994.

[Mey86] Norman Meyrowitz. Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia system. In *Proceedings of Object Oriented Programming Systems, Languages and Applications (OOPSLA) 1986*, ACM, Brown University, Institute for Research in Information and Scholarship, October 1986. SIGPLAN.

[Mey89] Norman Meyrowitz. The Missing Link: Why We're All Doing Hypertext Wrong. *The Society of Text*, pages 107–114, 1989.

[MHEG97] International Standards Organisation. MHEG Part 5. ISO/IEC IS 13522-5, 1997.

[MI89] Catherine C. Marshall and Peggy M. Irish. Guided Tours and On-Line Presentations: How Authors Make Existing Hypertext Intelligible for Readers. In *Proceedings of Hypertext 1989*, pages 15–26. ACM, November 1989.

[MIC94] Catherine C. Marshall, Frank M. Shipman III, and James H. Coombs. VIKI: Spatial Hypertext Supporting Emergent Structure. In *Proceedings of ECHT 1994*, pages 13–23. ACM, September 1994.

[MIRJ91] Catherine C. Marshall, Frank M. Shipman III, Russell A. Rogers, and William C. Janssen Jr. Aquanet: a hypertext tool to hold your knowledge in place. In *Proceedings of Hypertext 1991*, pages 261–275. ACM, December 1991.

[Mos] Documentation on the Mosaic web browser is available on the NCSA WWW site (URL: http://www.ncsa.uiuc.edu).

[Mou91] Stuart Moulthrop. Beyond the Electronic Book: A Critique of Hypertext Rhetoric. In *Proceedings of Hypertext 1991*, pages 291–298. ACM, December 1991.

[Mou92] Stuart Moulthrop. Toward a Rhetoric of Informating Texts. In *Proceedings of ECHT 1991*, pages 171–180. ACM, November 1992.

[MR92] Catherine C. Marshall and Russell A. Rogers. Two Years Before the Mist: Experiences with AquaNet. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 53–62, Milano, November 1992.

[MS95] Catherine C. Marshall and Frank M. Shipman III. Spatial Hypertext: Designing for Change. *Communications of the ACM*, 38(8):88–97, August 1995.

[MS96] A. Micarelli and F. Sciarrone. A case-based toolbox for guided hypermedia navigation. In *Fifth International Conference on User Modeling (UM '96)*, pages 129–136, 1996.

[MSE] Information on the MicroSoft web browser can be found on the MicroSoft WWW site (URL: http://www.microsoft.com).

[Nel93] Theodor Nelson. *Literary Machines*. Mindful Press, 1993.

[Nel95] Theodor Holm Nelson. The Heart of Connection: Hypermedia Unified by Transclusion. *Communications of the ACM*, 38(8):31–33, August 1995.

[Net] Information on the Netscape Navigator web browser can be found on the Netscape Communications WWW site (URL: http://www.netscape.com).

[New97a] Paul Newton. Hypertext using fuzzy anchors. In *Proceedings of Hypertext and Hypermedia: Products, Tool and Methods (H2PTM '97)*, pages 353–366. Hermes, September 1997.

[New97b] Paul Newton. Incorporating Hypertext using Fuzzy Components. In *Workshop on Incorporating Hypertext Functionality into Software Systems (HTFIII)*, 1997. Held in conjunction with Hypertext '97.

[NFS89] Sun Microsystems, Inc. NFS: Network File System Protocol Specification, March 1989. Request for Comments (RFC): 1094.

[Nie90] Jakob Nielsen. Through Hypertext. *Communcations Of The ACM*, 33(3):296–310, March 1990.

[NIS] Sun Microsystems. Network Information Service Plus (NIS+): An Enterprise Naming Service. Available from the Sun Microsystems WWW site (URL: http://192.9.48.5/solaris/wp-nisplus/).

[NK89] James M. Nyce and Paul Kahn. Innovation, Pragmaticism and Technological Continuity: Vannevar Bush's Memex. *Journal of the American Society for Information Science*, 40(3):214–220, 1989.

[NLS96] Peter J. Nürnberg, John J. Leggett, and John L. Schnase. Hypermedia Operating Systems: A New Paradigm for Computing. In *Proceedings of Hypertext 1996*, pages 194–202. ACM, 1996.

[NLS97] Peter J. Nürnberg, John J. Leggett, and Erich R. Schneider. As We Should Have Thought. In *Proceedings of Hypertext 1997*, pages 96–102. ACM, April 1997.

[Noi93] Emanual G. Noik. Exploring Large Hyperdocuments: Fisheye Views of Nested Networks. In *Proceedings of Hypertext 1993*, pages 192–205. ACM, November 1993.

[Nye92] Adrian Nye. *Xlib Programming Manual*, volume 1. O'Reilly and Associates, Inc., 1992.

[Obja] Object Management Group, Inc. Common Facilities Architecture v4.0. Available from the OMG WWW site (URL: http://www.omg.org).

[Objb] Object Management Group, Inc. CORBAservices: Common Object Services Specification. Available from the OMG WWW site (URL: http://www.omg.org).

[Objc] Object Management Group, Inc. The Common Object Request Broker Architecture and Specification v2.1. Available from the OMG WWW site (URL: http://www.omg.org).

[Obj97] Object Management Group, Inc. A Discussion of the Object Management Architecture. Available from the OMG WWW site (URL: http://www.omg.org), January 1997.

[ODA85] International Standards Organisation. Office Document Architecture (ODA) and Interchange Format. ISO/IEC 8613 Parts 1-8, 1985.

[OHSa] The Open Hypermedia Systems Working Group WWW site (URL: http://www.csdl.tamu.edu/ohs).

[OKW96] Kasper Østerbye and Uffe Kock Wiil. The Flag Taxonomy of Open Hypermedia Systems. In *Proceedings of Hypertext 1996*, pages 129–139. ACM, March 1996.

[ON94] Kasper Østerbye and Kurt Nørmark. An Interaction Engine for Rich Hypertexts. In *Proceedings of ECHT 1994*, pages 167–176. ACM, September 1994.

[Ope] OpenDoc documentation available from the IBM WWW site (URL: http://www.software.ibm.com/ad/opendoc).

[Par91] H. Van Dyke Parunak. Don't Link Me In: Set Based Hypermedia for Taxonomic Reasoning. In *Proceedings of Hypertext 1991*, pages 233–242. ACM, December 1991.

[PD90] R. Pausch and J. Detmer. Node popularity as a hypertext browsing aid. *Electronic Publishing: Origination, Dissemination and Design*, 3(4):227–234, November 1990.

[Pea89] Amy Pearl. Sun's Link Service: A Protocol For Open Linking. In *Proceedings of the 1989 Hypertext Conference*, pages 137–146, November 1989.

[PLGU95] T. Pérez, P. Lopisteguy, J. Gutierrez, and I. Usandizaga. HyperTutor: From hypermedia to intelligent adaptive hypermedia. In *Proceedings of the World conference on educational multimedia and hypermedia (ED-MEDIA '95)*, pages 529–534, 1995.

[PT90] Xavier Pintado and Dennis Tsichritzis. SaTellite: Hypermedia Navigation by Affinity. In *Proceedings of the European Conference on Hypertext (ECHT 1990)*, pages 274–287. ACM, November 1990.

[PYS90] Murugappan Palaniappan, Nicole Yankelovich, and Mark Sawtelle. Linking active anchors: a stage in the evolution of hypermedia. *Hypermedia*, 2(1):47–66, 1990.

[QV86] Vincent Quint and Irène Vatton. Grif: An Interactive System for Structured Document Manipulation. In J.C. van Vliet, editor, *Text Processing and Document Manipulation – Proceedings of the International Conference*, pages 200–213, April 1986.

[Ras87] Jef Raskin. The Hyper in Hypertext. In *Proceedings of Hypertext 1987*, pages 325–330. ACM, November 1987.

[RS92] Antoine Rizk and Louis Sauter. Multicard: An Open Hypermedia System. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 4–10, Milano, November 1992.

[RSG97] Gustavo Rossi, Daniel Schwabe, and Alejandra Garrido. Design Reuse in Hypermedia Applications Development. In *Proceedings of Hypertext 1997*, pages 57–66. ACM, April 1997.

[RvOHB97] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. Cooperative use of MHEG-5 and HyTime. In *Proceedings of Hypertext and Hypermedia: Products, Tool and Methods (H2PTM '97)*, pages 57–73. Hermes, September 1997.

[Sal89] G. Salton. *Automatic Text Processing – The Transformation Analysis and Retrieval of Information by Computer*. Addison Wesley, 1989.

[SCG89] Frank M. Shipman, R. Jesse Chaney, and G. Anthony Gorry. Distributed Hypertext for Collaborative Research: The Virtual Notebook System. In *Proceedings of the 1989 Hypertext Conference*, pages 129–135, November 1989.

[SF89] P. David Stotts and Richard Furuta. Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*, 7(1):3–29, January 1989.

[SGML85] International Standards Organisation. Standard Generalized Markup Language (SGML). ISO/IEC IS 8879, 1985.

[Shn87] Ben Shneiderman. User interface design for the Hyperties electronic encyclopædia. In *Proceedings of Hypertext 1989*, pages 189–194. ACM, November 1987.

[SL89] John L. Schnase and John J. Leggett. Computational Hypertext in Biological Modelling. In *Proceedings of Hypertext 1989*, pages 181–197. ACM, November 1989.

[SLH+93] John L. Schnase, John J. Leggett, David L. Hicks, Peter J. Nuernberg, and J. Alfredo Sanchez. Design and Implementation of the HB1 Hyperbase Management System. *Electronic Publishing: Origination, Dissemination and Design*, 6(1):35–63, June 1993.

[SLH94] John L. Schnase, John J. Leggett, and David L. Hicks. Open Architectures For Integrated Hypermedia-Based Information Systems. In *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, pages 386–395, January 1994.

[SLHS93] John L. Schnase, John J. Leggett, David L. Hicks, and Ron L. Szabo. Semantic Data Modeling of Hypermedia Associations. *ACM Transactions on Information Systems*, 11(1):27–50, January 1993.

[SM96] Sun Microsystems. Java Object Serialization Specification v1.2. Available from the JavaSoft WWW site (URL: http://java.sun.com), December 1996.

[SM97] Sun Microsystems. Java Remote Method Invocation Specification v1.42/jdk 1.2 beta 1. Available from the JavaSoft WWW site (URL: http://java.sun.com), October 1997.

[SM98a] Sun Microsystems. Java Message Service v0.91. Available from the JavaSoft WWW site (URL: http://java.sun.com), March 1998.

[SM98b] Sun Microsystems. Java Transaction Specification v1.0. Available from the JavaSoft WWW site (URL: http://java.sun.com), March 1998.

[SM98c] Sun Microsystems. JavaSpace Specification v0.999. Available from the JavaSoft WWW site (URL: http://java.sun.com), March 1998.

[SM98d] Sun Microsystems. JNDI: Java Naming and Directory Interface v1.1. Available from the JavaSoft WWW site (URL: http://java.sun.com), January 1998.

[Sny86a] Alan Snyder. Commonobjects: An overview. *ACM SIGPLAN Notices*, 21(10):19–28, October 1986.

[Sny86b] Alan Snyder. Encapsulation and Inheritance in Object-Oriented Programming Languages. In *Proceedings of Object Oriented Programming Systems, Languages and Applications (OOPSLA) 1986*, Hewlett-Packard Laboratories, October 1986. SIGPLAN.

[SP82] Zaw-Sing Su and Jon Postel. The Domain Naming Convention for Internet User Applicationst, August 1982. Request for Comments (RFC): 819.

[Spi88] Robert Spinrad. Dynamic Documents. *Harvard University Information Technology Quarterly*, 7(1):15–18, 1988.

[SR95] Daniel Schwabe and Gustavo Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 38(8):45–46, August 1995.

[SS90] H.A. Schütt and N. Streitz. HyperBase: A hypermedia engine based on a relational database management system. In *Proceedings of the European Conference on Hypertext (ECHT 1990)*, pages 95–108, November 1990.

[SS91] John B. Smith and F. Donelson Smith. ABC: A Hypermedia System for Artifact-Based Collaboration. In *Proceedings of the 1991 Hypertext Proceedings*, pages 179–192, December 1991.

[SSS93] Douglas E. Shackleford, John B. Smith, and F. Donelson Smith. The Architecture and Implementation of a Distributed Hypermedia Storage System. In *Proceedings of the 1993 Hypertext Conference*, pages 1–13, November 1993.

[Sta] Richard Stallman. *GNU Emacs Manual*. Free Software Foundation, 6.0 edition.

[Sto91] P. David Stotts. Dynamic Adaptation of Hypertext Structure. In *Proceedings of Hypertext 1991*, pages 219–231. ACM, December 1991.

[Str91] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1991.

[Str96] N. A. Streitz. Designing Collaboration Environments Based on the Common Ground of Hypermedia and CSCW. Tutorial held in conjunction with Hypertext '96, March 1996.

[Sun95a] Sun Microsystems. *The ONC+ Developer's Guide*, 1995.

[Sun95b] Sun Microsystems, Inc. The Hotjava Browser: A White Paper. Available from Sun Microsystems, 1995.

[SWF87] John B. Smith, Stephen F. Weiss, and Gordon J. Ferguson. A Hypertext Writing Environment and its Cognitive Basis. In *Proceedings of Hypertext 1987*, pages 195–214. ACM, November 1987.

[TB90] Douglas B. Terry and Donald G. Baker. Active Tioga Documents: An Exploration of Two Paradigms. *Electronic Publishing: Origination, Dissemination and Design*, 3(2):105–122, May 1990.

[TBR93] Frank Wm. Tompa, G. Elizabeth Blake, and Darrell R. Raymond. Hypertext by Link-Resolving Components. In *Proceedings of the 1993 Hypertext Conference*, pages 118–130, November 1993.

[TD92] K. Tochtermann and G. Dittrich. Fishing for Clarity in Hyperdocuments with Enhanced Fisheye-Views. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 212–221. ACM, November 1992.

[TMH87] Randall H. Trigg, Thomas P. Moran, and Frank G. Halasz. Adaptability and Tailorability in NoteCards. In *Proceedings of INTERACT 1987*, pages 723–728. Elsevier Science Publishers, 1987.

[Tri83] Randall H. Trigg. *A Network-Based Approach to Text Handling for the Online Scientific Community*. PhD thesis, University of Maryland, 1983.

[Tri86] Randall H. Trigg. TEXTNET: a Network-Based Approach to Text Handling. *ACM Transactions on Office Information Systems*, 4(1):1–23, January 1986.

[TSH86] Randall H. Trigg, Lucy A. Suchman, and Frank G. Halasz. Supporting Collaboration In Notecards. In *Proceedings of Computer Supported Cooperative Work*, pages 153–162, Austin, Texas, December 1986.

[Uni93] International Telecommunications Union. The Directory – overview of concepts, models and service, 1993.

[URL94] Internet Engineering Task Force. Uniform Resource Locators (URL), 1994. Request for Comments (RFC): 1738.

[Wat97] Carolyn Watters. Link Levels in an Open Hypertext View. In *Workshop on Incorporating Hypertext Functionality into Software Systems (HTFIII)*, 1997. Held in conjunction with Hypertext '97.

[Web] The Webcosm WWW site
(URL: http://www.webcosm.com).

[Whi97] E. James Whitehead Jr. An Architectural Model for Application Integration in Open Hypermedia Environments. In *Proceedings of Hypertext 1997*, pages 1–12. ACM, April 1997.

[Wii91a] Uffe Kock Wiil. Issues in the Design of EHTS: A Multiuser Hypertext System for Collaboration. Technical Report R 91-24, Department of Mathematics and Computer Science, University of Aalborg, June 1991.

[Wii91b] Uffe Kock Wiil. Using Events as Support for Data Sharing in Collaborative Work. In *Proceedings of the International Workshop on CSCW*, pages 162–176, 1991.

[WL92] Uffe Kock Wiil and John J. Leggett. Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems. In *Proceedings of the 4th ACM European Conference on Hypertext (ECHT 1992)*, pages 251–261. ACM, November 1992.

[WL93] Uffe Kock Wiil and John J. Leggett. Concurrency Control in Collaborative Hypertext Systems. In *Proceedings of the 1993 Hypertext Conference*, pages 14–24, November 1993.

[WL96] Uffe Kock Wiil and John J. Leggett. The HyperDisco Approach to Open Hypermedia Systems. In *Proceedings of the Hypertext 1996 Conference*, pages 140–148, Washington, DC, March 1996.

[WWWa] The World Wide Web Consortium WWW site (URL: http://www.w3.org).

[WWWb] Information on URIs, URLs and URNs available from the World Wide Web Consortium WWW site (URL: http://www.w3.org/addressing).

[WWW98a] World Wide Web Consortium (W3C). Cascading Style Sheets, level 2, May 1998. The latest documentation is available from the W3C WWW site (URL: http://www.w3.org).

[WWW98b] World Wide Web Consortium (W3C). Document Object Model Specification v1.0, April 1998. Work in progress. The latest documentation is available from the W3C WWW site (URL: http://www.w3.org).

[WWW98c] World Wide Web Consortium (W3C). Extensible Markup Language (XML v1.0, February 1998. The latest documentation is available from the W3C WWW site (URL: http://www.w3.org).

[WWW98d] World Wide Web Consortium (W3C). Hypertext Markup Language (HTML) 4.0 Specification, April 1998. The latest documentation is available from the W3C WWW site (URL: http://www.w3.org).

[Yah] The Yahoo! WWW site. http://www.yahoo.com.

[YHK93] W. Yeong, T. Howes, and S. Kille. X.500 Lightweight Directory Access Protocol, July 1993. Request for Comments (RFC): 1487.

[YHMD88] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. Intermedia: The Concept and Construction of a Seamless Information Environment. *Computer*, pages 81–96, January 1988.

[You90] Laura De Young. Linking Considered Harmful. In *Proceedings of the European Conference on Hypertext (ECHT 1990)*, pages 238–249. ACM, November 1990.

[Zad65] L.A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.

[Zel89] Polle T. Zellweger. Scripted Documents: A Hypermedia Path Mechanism. In *Proceedings of Hypertext 1989*, pages 1–14. ACM, November 1989.

[ZR97] Hada Ziv and Debra J. Richardson. Adding Uncertainty to Hypertext Models of Software Systems. In *Workshop on Incorporating Hypertext Functionality into Software Systems (HTFIII)*, 1997. Held in conjunction with Hypertext '97.