VISUALISATION OF HYPERMEDIA SYSTEMS: AN OPEN APPROACH

By

Mark James Weal

B.Sc.(Hons)

A thesis submitted for the degree of

Doctor of Philosophy

Department of Electronics and Computer Science,

University of Southampton,

United Kingdom.

August 2000

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

ELECTRONICS AND COMPUTER SCIENCE DEPARTMENT

Doctor of Philosophy

Visualisation of Hypermedia Systems:
An Open Approach

by Mark James Weal

Hypermedia systems are designed to allow links, or connections, to be made between different media objects. Key issues tackled in early hypermedia systems included developing tools to help guide users through the material and tools to help authors maintain the material that they create. The open approach to hypermedia emerged, where links were separated from the content of documents, allowing a more modular approach to hypermedia services. The ease of integration of tools in these open systems promoted the creation of many different types of navigational aids, designed to help users of the systems to access and maintain the information contained within them.

The openness and modular nature of such systems creates its own problems however. Users will often have to interact with a number of disparate interfaces to manipulate the navigational information. A new approach is presented which provides an open framework for these interfaces, allowing for a co-ordinated strategy and the modular addition of tools to help manage the screen interface and reduce the complexity of the interaction for users.

A second approach to the problem is to provide the different hypermedia information within a unifying visualisation. A novel framework is presented which allows more open access to the underlying navigational information of hypermedia systems. Visualisation tools can be connected to this framework in a modular fashion to provide flexible visualisations of the underlying information. By generating a number of different visualisations, the openness and flexibility of the visualisation framework approach is demonstrated.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Professor Wendy Hall, for all her help, encouragement and perseverance during the course of this Ph.D., and also for helping keep me employed within the stimulating and supportive Multimedia Research Group.

I am of course indebted to the many people who have contributed to the development of the Microcosm system which formed the foundations of much of the work presented here, in particular Ian Heath, Gary Hill, Rob Wilkins, Stuart Goose, Nick Beitner and Hugh Davis.

Many other members of the lab have provided me with thought provoking discussions and useful help and assistance, so I would also like to thank Danius Michaelides, Jonathan Dale, Gareth Hughes, Dave DeRoure, Luc Moreau and Mark Dobie.

Finally I must thank my family who have been supportive of me for so long and must have wondered if I would ever finish writing this thesis. I thank them most of all for never letting me know if they did.

Chapter 1

Introduction

The field of hypermedia is relatively young, tracing its roots back to the ideas of Vannevar Bush in the late forties. In this short space of time it has advanced to a point where providing access to hypermedia via the Internet is quite possibly the fastest growing industry on the planet. The World Wide Web, a hypermedia system with a single content base of information spanning the globe, has grown so fast that in less than ten years, seemingly half the adverts on television or in print contain URL entry points into the resource base.

Managing content and structure on this scale has been one of the problems that hypermedia researchers have focussed on and a key concept to come out of this research is the benefit of open approaches to hypermedia. In simple terms, the open approach separates the media content of the hypermedia from the structure that binds it together, ostensibly the links. By keeping the links separate from the data, the decision as to where to place the links can be deferred until the user views the information. This allows different structures to be overlaid on top of the same content and provides a mechanism for creating user tailored content that might be based on the user's specific interests. Although the separation of the links does lead to some problems, principally that of maintaining link integrity, it has provided new avenues for hypermedia designers to explore.

One of the areas of hypermedia that has received little attention is that of interface. Hypermedia systems have traditionally had their own ad hoc interfaces tied in to the underlying implementation of the hypermedia system. Open hypermedia systems allow the underlying information to be structured in many different ways, but often only through a single interface. Scalable interfaces which can be modified to account for user ability, hardware requirements and environment provide even more flexibility to such systems. This thesis seeks to examine the requirements of such open interfaces and the benefits they can provide to open hypermedia systems.

Opening the interface can enhance the presentation of the content, but in order to visualise the information in the systems the openness needs to be taken further. Even in systems that claim openness, often the information used by the system is closed and presented in proprietary interfaces. An example of this is the list of documents which the user has viewed using a World Wide Web browser. This information can be used to colour links to documents that the user has previously read, and the user can select from a list of documents if they wish to revisit them. Outside of these hardwired presentation mechanisms however the history information is sealed away and cannot be accessed by other parts of the system or indeed other applications wishing to make use of it. Without this freedom of access to the information truly open visualisations of the information are not possible.

1.1 Thesis Structure

Chapter 2 of this thesis will look at open hypermedia systems. An historical perspective is provided on hypermedia in general and a number of current hypermedia systems are examined. Navigation Tools within hypermedia systems are discussed and definitions provided for closed and open hypermedia. Concluding the chapter, a number of problems associated with the open approach are listed.

Chapter 3 takes a detailed look at the Microcosm open hypermedia system, which will form the foundation for much of the implementation discussed in later chapters. The openness of the system is examined, concentrating on the areas of architecture, message format, linking and functionality. Some of the navigation tools provided by the Microcosm system are discussed along with problems arising from the current architecture.

Chapter 4 looks at screen management issues. A brief history is given, and a number of different approaches to the problem are presented. Specific screen management issues that arise with open hypermedia systems are examined and the concept of open interfaces is discussed.

Chapter 5 presents a novel architecture for screen management called the Screen Handler Enabling Process, or SHEP for short. The design and implementation of the architecture is discussed in detail and examples of its use with real world problems are presented.

Chapter 6 looks at visualisation systems, both in the general case, and also how they relate to the field of open hypermedia. The use of metaphors in such systems is discussed along with a number of tools that exist for creating visualisations. A number of visualisation systems are discussed. Some of the problems of visualisation systems are described.

Chapter 7 presents a novel framework called Minerva, which is designed to help provide visualisations both of a general nature, but more specifically of open hypermedia systems. The design and implementation of the framework is discussed in detail. The incorporation into the framework of the standard hypermedia navigation tools discussed in chapter 2, is examined. Some novel navigational devices are suggested which might take advantage of the openness of the Minerva framework.

Chapter 8 looks at three visualisations of real world problems using the Minerva framework. Each presents its own unique issues and illustrates the openness of the framework.

Chapter 9 examines some of the limitations of the current implementation of the Minerva framework and discusses what future extensions and improvements might be made to the framework to overcome these problems.

Chapter 10 draws conclusions from the work presented in the previous chapters. The issues of open interfaces and open visualisations are summarised and possible ways forward suggested.

1.2 Declaration

Although this entire thesis is the author's personal view of the field, the original contributions are described in chapters 5,7 and 8. The work in this thesis has been conducted within a collaborative research group. It is all the author's own work, with the exception of that described in sections 5.6.1 and 5.9.2, where other members of the research group have in some way contributed.

Chapter 2

Open Hypermedia

2.1 Introduction

This chapter provides an overview of the history of hypertext and hypermedia and the theory behind it. Historical hypertext systems are discussed, illustrating the development of the field of hypertext. Current hypermedia systems are then covered, examining the methodology and aims behind them. The nature of links is examined as well as the wealth of navigation tools which have been devised to help assist users of hypermedia systems. A distinction is drawn between closed and open hypermedia systems and finally, some of the problems surrounding open hypermedia are discussed.

2.2 Historical Systems

Depending on your definition of hypertext, the date of birth of hypertext could vary by a few thousand years. Early versions of the Bible contained frequent cross-referencing and the Tamulad, an early religious text, made heavy use of annotations and nested commentaries.

In his article 'Hypertext: An Introduction and Survey'(Conklin, 1987), Conklin proposed that in order for something to be considered *true hypertext*, navigation had to be computer-supported. If we agree with this then we need only look back fifty years for the origins of hypertext.

2.2.1 Memex

In his seminal paper ‘As We May Think’, published in *Atlantic Monthly* in 1945 (Bush, 1945), Vannevar Bush wrote, in reference to the way library systems were organised,

> The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain...
> One cannot hope thus to equal the speed and flexibility with which the mind follows an associative trail, but it should be possible to beat the mind decisively in regard to the permanence and clarity of the items resurrected from storage.

The system that Bush went on to describe he called the “Memex”. It was a device in which all books, records and communications of an individual would be kept on Microfilm (being the only viable mass storage option at the time). Users could create trails through the information by linking two documents to each other. At a later date it would be possible to follow these trails through the material as if turning through the pages of the book.

He talked about the future, predicting

> Wholly new forms of encyclopaedias will appear, ready-made with a mesh of associative trails running through them.

Although unable to predict the rapid advances in digital technology, the ideas and working environments suggested by Bush can be seen in many of the systems around us today.

2.2.2 NLS and Augment

In 1963, Douglas Engelbart wrote ‘A Conceptual Framework for the Augmentation of Man’s Intellect’ (Engelbart, 1963). He proposed a system called H-LAM/T (Human using Language, Artefacts, and Methodology, in which he is Trained) which would seek to ‘amplify the native intelligence of the user’. These ideas were refined and eventually became NLS (oN Line System) which ran on complicated consoles that included a revolutionary input device known as a mouse.

NLS had three main components: a database containing the text fragments, view filters, which enabled the viewing of items from the database, and views, which were used to structure the information. The view filters could be used to carry out simple filtering such as only showing the file hierarchy to a certain depth, as well as more complex filtering such as only showing files which matched a specified search.

NLS evolved into a system called Augment, which was commercialised by McDonnell Douglas as a system for 'knowledge workers'(Engelbart *et al.*, 1973).

2.2.3 Xanadu

Ted Nelson, widely considered as a hypertext visionary, is credited with coining the term 'hypertext'. In developing his ideas for a unified literary environment, Nelson designed the Xanadu system (Nelson, 1980), named after the mythical place described in Samuel Taylor Coleridge's poem "Kubla Khan" .

The idea behind Xanadu was to greatly reduce storage requirements by extensive use of linking. The linking would serve in place of in-line quotes and would provide versioning, by only storing the original version of a document and links to the subsequent modifications.

Another idea that heavily influenced the design was that of maintainable copyright. Whenever a link referenced another piece of material, the copyright of the destination material could be readily established. This would allow for very accurate royalty payments, with material being paid for whenever it is referenced.

2.2.4 NoteCards

Xerox PARC's NoteCards (Halasz *et al.*, 1987)was one of the first widely used hypertext systems. It was developed as an information analyst's support tool to assist in gathering and assimilating information.

Users create a series of nodes, or notecards, which are then linked together to form a web. Although not strictly an open hypermedia system, as will be seen in a later section, NoteCards allows the creation of new node types in addition to the basic types. An integrated Lisp programming environment enables the open architecture of NoteCards to be extended to create new applications.

2.2.5 Sun Link Service

One of the first implementations of an open hypermedia link service was produced by Sun (Pearl, 1989). The idea behind the system was to separate the authoring and manipulation of links from the document management. This allowed the editing of documents to be carried out by the most appropriate applications with the links being added by a back-end link service process. This system was therefore one of the first hypertext systems not to rely on document mark-up.

As well as providing a link server program, a protocol was specified and a library provided which enabled third party applications to be integrated with the system. The flexible protocol enabled the applications to support as much of the service as they could, not restricting them to all or nothing compliance. By using native applications in the editing of documents and accessing of hypermedia functionality, users were presented with a familiar environment. The disadvantage of this is that different editors would provide different interfaces to the protocol, which might be confusing to users.

2.2.6 Intermedia

Developed at Brown University, the Intermedia Project (Yankelovich *et al.*, 1988) built on the early work of hypertext pioneers like Ted Nelson, such as the Hypertext Editing System and FRESS.

One of the goals of the project was to use hypertext systems within the classroom to support teaching. Documents of different media types could be added to the system and linked together. The system was designed to be both the authoring environment and also the end user browsing tool, enabling the students to add their own material and annotations to the system and link it into the courseware.

As more links and material were added to the system, they could quickly become too complex for users to comprehend, so to reduce this problem the concept of webs was introduced. Links belonged to specific webs that provided a context for the linking. Users could choose which web they wished to view and only those links would be displayed to them.

In addition to the basic document display and link following, the Intermedia Project implemented a number of navigational aids for the users of the system. Global maps provided the users with overviews of all the material, and local maps gave the user an idea of what material was linked to the document they were currently viewing. These navigational tools and others will be discussed in more detail in later sections.

2.2.7 Guide

The Guide system (Brown, 1987) was created as a research project at the University of Kent, Canterbury, in 1982. It established itself in the hypertext community and was successfully commercialised on the Macintosh and PC. Research was continued on the system and many interesting features were devised.

The Guide system was a closed hypertext system in that files had to be imported into the system. Links were then made within the documents themselves, appearing as underlined pieces of text. When users click on the link they are transported to the document at the other end of the link.

Because the text and links are all held within the system, new approaches to linking were possible. One of these was the replace link, where instead of taking the user to a new document, the document forming the destination of the link was embedded within the source document at the point of the link anchor. This technique of link following has the advantage of showing the linked information in context.

In order to aid navigation, Guide supported the creation of paths through the data set, which provided a structured, restricted view of the material for users to follow. The techniques described above both attempted to convey the structure of the information to the user without resorting to overview diagrams or maps.

2.2.8 Other Systems

There have been many other influential hypertext systems implemented, such as TextNet (Trigg *et al.*, 1986), KMS (Akscyn *et al.*, 1988) and Hyperties (Schneiderman, 1987). A detailed review of these systems can be found in (Conklin, 1987).

2.3 Existing Systems

Hypertext and hypermedia have now entered the mainstream. The sections below examine a number of currently used hypertext and hypermedia systems.

2.3.1 The World Wide Web

To start at the beginning, in 1969 the forerunner of the Internet came into existence. It was called the ARPANET. The idea behind it was to provide mechanisms for people to control computer systems and to access information remotely. In the first instance, it was the mechanisms for moving things from a to b that were concentrated on, and the nature and structure of the system was left uncontrolled. This meant that in order to access anything you had to know exactly where it was or you were going to have problems.

Twenty years later, in 1989, the Internet was firmly in place and there were servers all over the world. But the fundamental problem still existed. If you wanted to get information from a particular server you still needed to know an IP address or site name, which consists of dots and squiggles and a variety of other hieroglyphs. Once on the server it is then necessary to know the correct sequence of cryptic commands with which to retrieve the data. This was not a job for a novice computer user.

In 1989 Tim Berners-Lee was a software engineer at the Centre for European Particle Physics (CERN). He developed a hypermedia system that has now become universally known as the World Wide Web (WWW) (Berners-Lee, 1993a).

The Web, as it is often known, uses a system of URLs (Uniform Resource Locators) (Berners-Lee, 1992). These allow any arbitrary piece of information to be referred to by a single unique identifier and therefore give a standard addressing system.

In practice, this allows the whole Internet to be viewed as a single repository of information, rather than as a collection of discrete non-uniform storage facilities, each with their own access methods.

This doesn't mean to say that URLs are necessarily intuitive, after all,

```
http://www.iam.ecs.soton.ac.uk/
```

is not the easiest string in the world to remember but it does have a number of advantages.

A standardised protocol. Because the Web uses a standard protocol, browsers can be written which remove much of the retrieval effort from the user. Given that the user can remember the URL the browser is able to go to the remote machine, download the document, and display it for the user on the screen by using the HyperText Transfer Protocol (HTTP)(Berners-Lee, 1993b). This is made easier because all the servers and browsers are talking in the same language.

Simple hypermedia functionality Since every document has a unique address, it becomes easy to create documents that point to and reference other documents. This is a simple hypermedia link and is usually implemented in hypertext browsers so that the user only has to click on the highlighted link to view the destination document. This removes the need for the user to deal with the URL, since the only action required on their part is clicking on a highlighted piece of text.

A number of standard hypermedia tools could also be included in the browsers such as history devices and forward and back buttons. The extensive use of bookmarks helps to minimise the requirements for remembering the URLs.

Sites have been created which provide searchable indexes to documents on the Web. These indexes can be created by the submission of documents for inclusion in the index, or even by 'web crawling', where an intelligent agent moves around the web following links and adding documents that it finds to the index. Because of the variety of ways of collecting the document indexes, the indexes themselves can be keyword based, or content based.

The use of executable code located on the Web servers has also allowed for greater diversity in the linking mechanisms. It is possible to decide what link to follow at run time. For example, a set of resources can be configured to either text only or full graphics depending on an initial decision made by the user. With the use of scripts it need not be necessary to create two sets of documents and links. Instead, the server can customise the documents and links on the fly before they are sent to the user. Another example of this is scripts which use the co-ordinates selected in an image to decide which link to follow.

For reasons that will be covered shortly however, the Web is not an open hypermedia system in that the links are embedded in the documents. Some systems do attempt to improve on this as will be discussed in later sections.

2.3.2 Hyper-G and Harmony

Developed at the University of Graz in Austria, Hyper-G is an extension to the World-Wide-Web and was designed to move hypermedia systems away from simply being presentation systems and to allow more interactions from the users with the underlying data (Andrews *et al.*, 1995). The architecture is based around the concept of a hierarchy of collections that provide more structure than the more common node and link hypermedia design. Hyper-G has an open approach to linking, with the links being stored separate to the document collections. This allows for bi-directional links and also guaranteed link consistency. The object-oriented approach allows greater flexibility in authoring and maintenance and provides facilities for applying access permissions to links as well as the documents themselves. Hyper-G is a multi-user multi-author environment, relying on a back-end database that resolves all of the issues of concurrent updates and keeps track of user permissions. The architecture is client server, with readers and authors both using the same client to carry out their tasks.

Harmony is a client application for viewing Hyper-G that runs under UNIX/X11. As well as allowing the user to view the documents, the client can provide local maps that depict the document and its relation to other documents in the collection. This feature aims to help the user orient themselves within the hypertext by providing a sense of relative location. The viewer supports many different media types and has explicit support for multilingual user interfaces. This support goes beyond the content and includes the menus and buttons on the viewer itself.

2.3.3 Multicard

The Multicard hypermedia system provides a hypermedia toolkit that allows programmers to create and manipulate distributed hypermedia structures (Rizk, 1992). By providing a powerful back-end system, the intention is to allow developers to continue to use their existing editing and authoring tools while providing an underlying hypermedia service.

By using a communications protocol, known as the M2000, a variety of specialist editors can be utilised and assigned to specific hypermedia node types. The editors need not support the whole of the protocol, which allows the system to be flexible enough to incorporate many third party applications.

A separate hypermedia authoring tool is used to create the hypermedia structures of nodes and composites, but once authored, the M2000 compliant editors are used to browse the hypermedia structures and edit the content. In order to promote the openness of linking protocols, a link in Multicard is viewed as a message event that can include scripts. The use of a scripting language keeps the linking flexible and attempts to bring the API closer to the user.

The editor's user interface is not specified by the M2000 protocol and can be implemented in different ways. Although this obviously enables the native interface of third party applications to be used there can be a lack of coherence within the overall interface, with different methods being used to carry out the same underlying hypermedia task.

2.3.4 HyperDisco

The HyperDisco system was developed jointly by researchers at Aalborg University in Denmark and Texas A & M University (Wiil & Leggett, 1996). Wiil and Leggett state two main objectives met by the system, namely:

- to provide a platform to integrate existing and future distributed heterogeneous tools and data formats.

- to provide a platform to extend integrated tools to handle multiple collaborating users and multiple versions of shared artefacts.

The name HyperDisco is derived from *Hyper*media *Dis*tributed *Co*llaborative computing environment. The architecture consists of distributed HyperBase Management Systems (HBMS), tool integrators and third party tools. The tool integrator provides a separation between the third party tools and the underlying hypermedia services. Rather than forcing each tool to adhere to a single model of integration tools are allowed to have their own specialised model of integration which might include all or part of the hypermedia services on offer.

As with Multicard, scripts can be attached to hypermedia components to extend their behaviour. In order to keep the system as open as possible, multi user support is provided for concurrency control, locking and notification control mechanisms.

2.3.5 Chimera

The Chimera system was developed to help provide hypertext services within a software development environment (SDE) by a research team at the University of California, Irvine (Anderson *et al.*, 1994). The system was designed to help capture and visualise the relationships between objects in an SDE. The architecture is based around the idea of hypertext concepts. These include objects, viewers, views, anchors, links, attribute pairs and higher level concepts such as hyperwebs. These concepts are mapped onto the software development environment.

It uses a client server model, with the server being responsible for maintaining the concepts and hyperwebs and providing access to this information to the clients.

The Chimera system is open in that it supports a clearly defined API that allows access to the functionality from external applications. Developers are free to implement their own clients which can take advantage of the underlying hypertext functionality by registering an interest in one or more of the defined hypertext events of the system.

The openness of the Chimera system also facilitates the integration of third party tools and applications through the flexible API. One of the main problems cited with the system however is the lack of a consistent user interface to the hypertext due to the diversity of viewers that can be created for the system. The viewers can choose to implement the hypertext functionality in their own way, which leads to the lack of a uniform interaction style.

Another reason why Chimera fails to meet the requirements for a fully open hypermedia system is that the architecture is not extensible, as there are a pre-defined set of hypertext events that cannot be extended.

2.3.6 DeVise

The DeVise hypermedia framework (DHM) was developed at the Computer Science Department of Aarhus University in Denmark (K. Grønbæk and R. H. Trigg, 1992). The ideas behind the project were to provide tools to support co-operative system development and design, principally in the field of engineering. A number of requirements were established for the system, which included a shared database, platform independence, portability, extensibility and tailorability.

The Devise architecture is object oriented, with basic hypermedia objects supported such as anchors, links and composites. A multi-user object-oriented database was used to provide the necessary support for co-operative work.

The system was based on the Dexter Reference Model (Halasz & Schwartz, 1990), parting from the model in its support of dangling link structures, i.e. links which have a source anchor but no destination.

2.3.7 Microcosm

The Microcosm hypermedia link service(Fountain *et al.*, 1990) was developed at the University of Southampton and has since been successfully turned into a commercial product. Like the Sun Link Service, it avoids embedded mark-up by storing the links separately in linkbases. Chapter 3 of this thesis will look at the Microcosm system in more detail since it forms the underlying research framework on which the main work of this thesis was constructed.

2.4 Linking

Links form the basic building blocks of hypermedia systems. A link, in its simplest form, connects a source anchor and a destination anchor (often referred to as start and end anchors). Links usually carry with them the implication that the two anchors being joined have a semantic connection. Two classes of links can be readily identified, explicit and implicit.

Explicit Links

Explicit links have their start and end anchors fixed when they are created. Because the anchors are pre-defined, they can be easily highlighted by the system. By considering the granularity of the anchors, four categories of explicit link can be derived.

- Point to point links
- Point to document links
- Document to point links
- Document to document links

Each of these categories has their own particular use. Document to document links can be useful where the two documents are related as a whole, for example an image might be linked to a textual description of it. To take a different form of link, the name of a person occurring in a document might be linked to a paragraph about them in a large biographical index document. In this case, a point to point link would be most suitable.

The classifications above might suggest that links are uni-directional, but this need not be the case. Where a link is bi-directional, it is possible that one direction might be document to point and the other point to document.

Implicit Links

Links that have their start or end anchors dynamically generated by the system as a result of the user's actions are known as implicit links.

Three categories of implicit link can be identified based on the fixed or dynamic nature of the anchors.

- Fixed to dynamic
- Dynamic to fixed
- Dynamic to dynamic

The category which would complete the set is fixed to fixed, which has been identified above as an explicit link since both anchors are defined at the time of link creation.

An example of a fixed to dynamic link would be an anchor which causes a search to be launched. The result of the search is indeterminate when the link was authored as more material might be added to the system.

Dynamic to fixed might include links where the author enters a term in a dialogue which is matched against an index held in the system. The result could be the following of a link to a previously authored destination. The destination is fixed in advance by the author, but the act of entering the search term to be matched is dynamic.

Dynamic to dynamic would include searching mechanisms where the user created source of the link, i.e. the search term, is matched against an index and results in links being created to destinations suggested by the search. No authoring is required for these links and they can only be evaluated at search time.

There are two main drawbacks with implicit linking. Firstly, the concept of bi-directional linking is lost, in that one or both of the anchors are generated on the fly. Secondly, it becomes much harder to indicate to the user the anchors of the link. To give an example of the second case, a user might follow a link from a source anchor on the text 'Lord Mountbatten of Burma loved horses'. The destination is generated dynamically by a full text retrieval engine. Once the query has been made a document might be returned which contains the sentence

> Lord Mountbatten retained a number of servants in his house in Burma

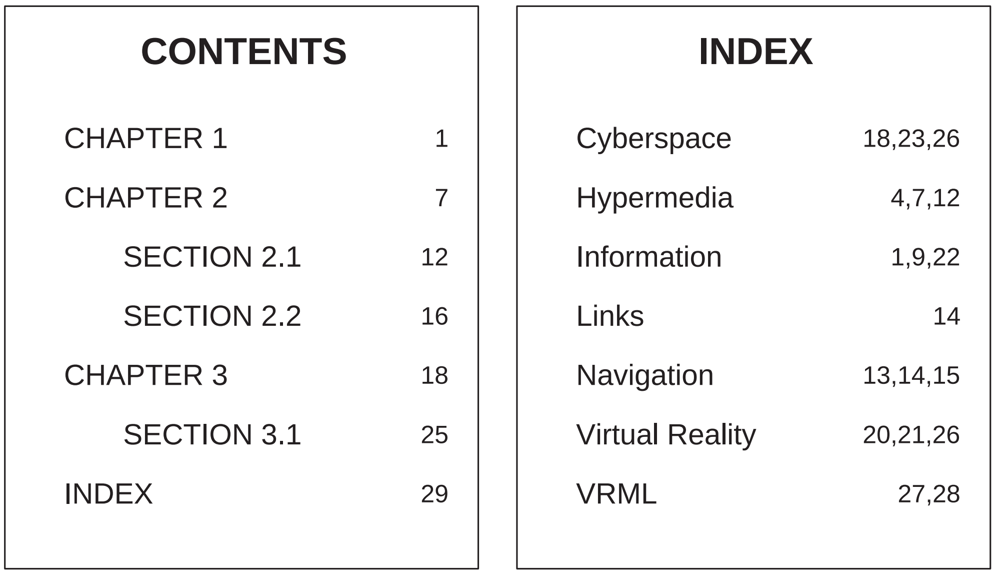
It is not clear what the system has matched as the destination anchor of this link, since not all of the text from the source anchor occurs in the sentence, and there is additional information that did not form part of the source anchor. This becomes worse when an entire document is presented as the match and it is not easy to discern the reason. Words that occur in the source anchor might be highlighted to offer some indication, but with complex search engines where synonyms are used there is no guarantee that *any* of the search terms would occur in a matching document. Because of this, often no indication is given to the user and they may be left wondering how the system decided this was an applicable document for them to view.

2.5 Navigation Tools

2.5.1 Direct Access Tools

Direct access tools, or DATs, provide a one to one mapping from a collection of unique representational objects to the documents within the document set. They can be divided into two categories based on the nature of the representational objects, textual DATs and graphical DATs.

Textual DATs



CONTENTS

CHAPTER 1	1
CHAPTER 2	7
SECTION 2.1	12
SECTION 2.2	16
CHAPTER 3	18
SECTION 3.1	25
INDEX	29

INDEX

Cyberspace	18,23,26
Hypermedia	4,7,12
Information	1,9,22
Links	14
Navigation	13,14,15
Virtual Reality	20,21,26
VRML	27,28

(a) A contents page. (b) An index.

Figure 2.1: Diagrams showing textual direct access tools.

Textual DATs present the user with textual descriptions of the objects from which to choose. A number of layouts are possible in the design of the tool.

The textual objects could be laid out hierarchically adding extra semantic information about the relationships between documents. An example of this might be an interface to a filing system.

Figure 2.1(a) shows a contents page. This is a direct access tool in that the user of a book can go directly to a particular chapter or section of the book using the page number given on the contents page. In a similar way, the index provides direct access to the fine grain of the information, by allowing the user to locate individual references to keywords within the body of the text. The index in Figure 2.1(b) is flat, but it is equally possible to create hierarchical indexes. The contents page illustrated is hierarchical, providing additional structural information about the document.

Graphical DATs

As the name implies, graphical DATs provide direct access to the underlying information in the form of graphs or diagrams. There are many types of graphical DATs including time lines, charts and maps.

The term map can encompass a wide variety of tools. In the context of a hypermedia system the map might visually show linking information (Yankelovich *et al.*, 1988) or may show the relationship between documents by their proximity on the map. This latter form is referred to as an affinity map (Pintado & Tsichritzis, 1990). A third type of map might present the information using more than two dimensions (Gloor, 1991).

The scope of the map can be varied, giving us either local maps, with a small scope, or global maps where the scope includes everything. The sections below describe local and global maps.

2.5.2 Local Maps

Local maps are used to present the user with information about documents that are within their local context. By restricting the scope to the local area, the map becomes less complicated. In the case of a hypermedia system this might be all the documents within one link from the user's current location. The map will display representations of the documents as well as the links between them. The links shown will often be just the links from the focal document to the other documents, but there is no reason why the map might not indicate all the links between all the documents represented on the map. Directional arrows can be used to indicate the direction of the link, or where appropriate that it is bi-directional. In theory there is no reason why such a map cannot be extended to show more than this first level of connections. The idea of levels of detail was included in the Intermedia system (Yankelovich *et al.*, 1988). In practice, however, a two dimensional map can quickly become too cluttered to be of use.

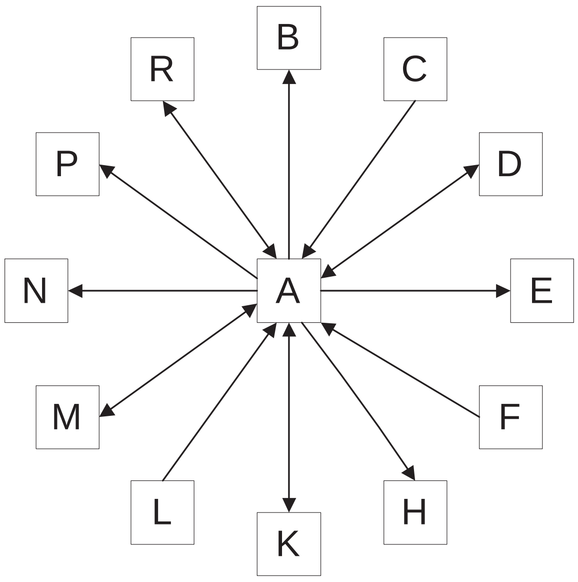


Figure 2.2: A local map.

Because the local map can often be generated dynamically there is no guarantee that the location of documents on the map will stay the same as the system is modified. A typical implementation of a local map would arrange connected documents in a circular pattern around the focal document of the map. The relative positions of documents have no significance other than to show they are attached to the central document. An example of such a local map is shown in Figure 2.2.

2.5.3 Global Maps

A global map is a diagrammatic representation of all the documents and links within a hypermedia application. If the number of documents or links is large then the global map becomes difficult to generate and use (Yankelovich *et al.*, 1988).

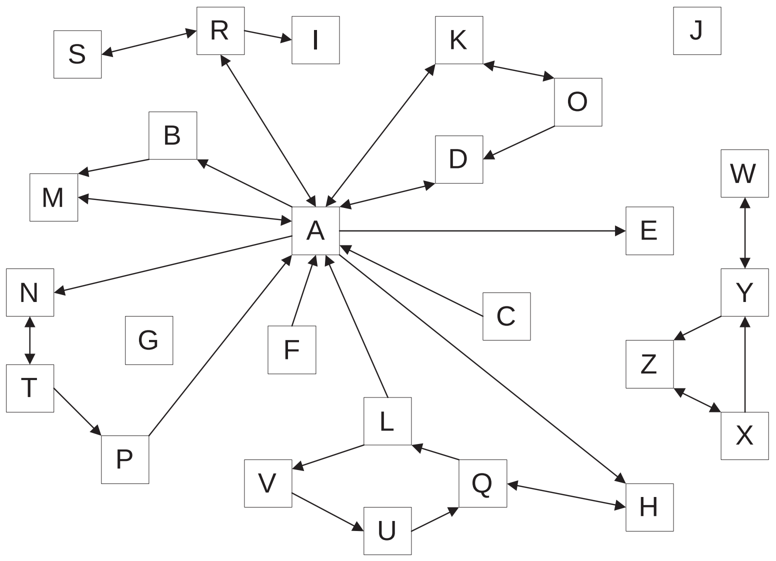


Figure 2.3: A global map.

Figure 2.3 shows an example of a global map. Only twenty-six documents are depicted on the map, yet already it is beginning to become cluttered. As in the local map, links are shown as lines with arrows denoting the direction of the links. A number of features can be clearly made out on the global map which were not previously identifiable on the local map. These are listed below.

Start points : The documents labelled F and C cannot be accessed from other documents via links. This implies that unless the user starts at these documents, they will never reach them using just link following as a navigation method. For this reason they are referred to as start points. It may be that this is the intended function of the documents, for example a contents page may well appear on a global map as a document with links leading from it, but none leading to it.

End points : Two of the documents shown on the global map (E and I) are end points within the hypermedia document set since once the user arrives at the documents they cannot follow any more links. These might prove to be potential problems, or could reflect the nature of the material. For example the documents might contain glossary definitions and thus once the user has read the document they are likely to backtrack to their previous position and carry on browsing from there. This does of course imply that link following is not the sole form of navigation within the system.

Islands : The documents G and J are not connected to the rest of the dataset by any links. For this reason they would not show up on any local maps unless they are the centre of the map. The Documents W, X, Y and Z could also be considered as an island even though they are connected to each other by links. Using just links, islands can be inaccessible to the user, and global maps are a useful tool for highlighting this potential problem to authors.

2.5.4 The History Tool

History tools record information about which documents a user has visited while they have been using the hypermedia system and often the order in which they were visited. The information can then be presented to the user in a number of different ways. Commonly, it is in the form of a list of documents sorted chronologically. Usually the head of the list will contain the most recently visited documents. The user is shown which documents they have seen but not how they got from one document to the next. An additional feature of a history tool might be to indicate to a user that they have visited a document more than once before.

The principle aim of the history tool is to allow the user to retrace their steps and keep a sense of continuity (analogous to flipping back through a book.) Users may decide they need to review previously read material or alternatively, when offered a choice of links, wish to filter out those documents that they have visited before.

History tools have also been implemented as a trail of breadcrumbs (Bernstein, 1988). A highlight is used to indicate that a user has been to a document before. The length of the breadcrumb trail is limited to a fixed number of most recently visited nodes. The idea behind this is to reduce the chances of the trails becoming incoherent as more and more documents are viewed.

2.5.5 Guides, Tours and Trails

Guides use some form of heuristics to provide the user with a dynamically changing tour. Guides suggest new documents to view based upon knowledge of the data set and information gathered either actively or passively from the user. Information is actively obtained by asking questions. Information is passively collected by monitoring the user's interactions with the system, for example, which links they follow or which documents they have already seen.

The simplest method for displaying this information is to provide the user with a list of files to see next. The guide can rank these documents based on its current information, placing the most important document in its opinion at the top of the list. Providing the user trusts the information the guide is presenting, it removes much of the navigational overhead from them. It may not always be obvious to the user why the guide has suggested a document however, since many of the guide's decision making processes are hidden.

With large systems the possible number of choices of where to go next can often exceed the optimal 'seven plus or minus two' (H. Gleitman, 1986). A principle objective of guides is to reduce the number of choices presented to the user to a more manageable number.

The guides discussed above automatically produce navigational information for the user. Tours are usually authored by domain experts as a means of introducing users to information that they are not familiar with (Nielsen, 1995)(Bieber *et al.*, 1997). Much of the work on tours has centred on the educational environment where the expert knowledge of teachers is encapsulated for use by students when approaching a new area of study (Shipman *et al.*, 1998)

Trails differ from tours in that they are created as a by product of the user moving through the material rather than being explicitly authored (DeRoure *et al.*, 1998). Trails could be likened more to 'beaten paths' as opposed to the laid out walkways of tours.

2.5.6 Search Engines

It is difficult to say whether search engines should be considered a navigation tool within a hypermedia system or an alternative approach. They are included here for completeness, and because in reality most hypermedia systems will include a search engine as part of their system.

It could be argued that the if the material contained in the hypermedia system is fully linked together then a search engine is redundant. This may be true where the size of the content is small and the linking is very heavily structured. However, in practice, even with automated link creation, where hypermedia systems are large a search engine can provide a quick mechanism to find specific material.

Also, quite often the linking of material in a hypermedia dataset is structured to some specific purpose. Where a user wishes to make use of the material but has different objectives, the links provided may be inappropriate and often a search mechanism can allow them to find the material more easily.

Because of the scale of the World Wide Web, a number of search engines have been created for it including commercial products such as Alta Vista, as well as freely available tools such as Harvest and Swish.

2.6 Definition of Closed Systems

To define what constitutes an open system, we can start by examining what makes most of the previously discussed systems closed. A number of features can be identified.

- In most closed systems, the application files need to be imported into the system. This often places restrictions on the type of files that can be used and may require the file to be converted to a proprietary file format.
- It is often difficult to extend the basic link model. This reduces the flexibility of the system and restrains both the author and user to operating within the confines of the supported linking mechanisms.
- Link anchors will usually be denoted by symbolic representations embedded within the documents. Commonly referred to as embedded mark-up, this enforces the use of particular formats. A good example of this is the World Wide Web where anchors are embedded within the HTML, for example

  ```
  <A HREF="destination.html">Source anchor</A>
  ```

Peter Brown (Brown, 1987) summed up some of the problems with closed hypertext systems in the following way:

> A second reason for failure is that the tool is an island itself and cannot be combined with other tools. Those of us that expect the whole world to rewrite its documentation to fit the needs of our new hypertext systems are unlikely to have our expectations fulfilled. Instead we

> must capture existing documents and have some way - even if crude - of automatically imparting structure to it. We must also work with existing tools such as spell checkers or encryption programs. This is to some extent a research area but more, I expect, a question of curbing some of our wilder aspirations so that, following a recurrent theme of this paper, we fit the world as it is rather than the world as we would like it to be.

Malcolm expands on this when referring to the needs of industrial strength hypermedia systems (Malcolm *et al.*, 1991).

> Systems should be designed that enable engineers to link data created with their own tools rather than by having to use special hypermedia editors.

2.7 Definition of Open Systems

In his thesis, Davis (Davis, 1995) attempted to define what makes a hypermedia system open. He condensed the popular opinion in the field into six key areas in which hypermedia systems should be open.

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

1. *Size:*
   It should be possible to import new nodes, links, anchors and other hypermedia objects without any limitation, to the size of the objects or to the maximum number of such objects that the system may contain, being imposed by the hypermedia system.
   This is essential if systems are to be able to provide Industrial Strength Hypermedia (Malcolm *et al.*, 1991). Systems should also be able to cope with dynamic information that is regularly revised and updated. Where systems are large in size, it is also quite likely that distribution of the dataset will be required, an issue discussed in detail by Hill (Hill, 1994).
2. *Data Formats:*

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

With new formats appearing all the time, and existing formats continually undergoing review and improvement, an open hypermedia system must be able to adapt to these changes and incorporate a wide variety of information. A system that restricts users to using HTML 1.0 when the rest of the world operates using HTML 4.0 would not be tolerated.

3. *Applications:*

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

   Ironically enough, it is often the systems with embedded mark-up which provide the best access to the link information as it is readily available from the documents themselves. In many open hypermedia systems, claiming to be so by virtue of their link separation, the links are stored in a private linkbase which is inaccessible from outside the system. This restricts integration with other services.

4. *Data Models:*

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

   A number of different data models have been proposed for hypermedia systems, notably the Dexter Reference Model (Halasz & Schwartz, 1990)(Halasz & Mayer, 1994). Problems have been highlighted with the Dexter model however (K. Grønbæk and R. H. Trigg, 1992) and it is doubtful that an all encompassing data model could be found. It is important therefore that an open hypermedia system be flexible enough to incorporate different data models.

5. *Platforms:*

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

   Cross platform portability has been dramatically improved during recent years with the rapid advance of distributed programming languages such as Java and simple communication protocols such as HTTP. This not only enables systems to be used on different platforms, but also allows different parts of systems to run on different platforms. A link service might be operating on a large UNIX based file system, communicating its hypermedia functionality to a client on a Windows machine.

6. *Users:*
   The system must support multiple users, and allow each user to maintain their own private view of the objects in the system.
   A private view should extend past simple storage of bookmarks, including such things as display preferences and also personal choices as to the sort of information to be presented with.

It should be noted that none of the systems described previously meet all six criteria set out above, but many are considered open as they obey the majority of the criteria. Moving from closed to open systems does solve some of the problems that were encountered with early hypermedia systems. There are unfortunately new problems introduced by open systems that have to be tackled.

2.8 Problems with Open Hypermedia

Many systems address the requirements of openness proposed by Davis and others, namely scalability, open data formats, application integration, data models, platform independence and user preferences. Indeed, Microcosm has been described as a system that boasts an open architecture, open message formats, open linking strategies and open functionality.

One area in which many hypermedia systems fall down is that they have inherently closed interfaces. The information is provided by each individual module in their own GUI's, which are often unconnected to other interfaces of the system.

2.9 Conclusions

The relatively new field of hypermedia has progressed rapidly in the last decade. The visionary ideas of people like Vannevar Bush and Ted Nelson are becoming a reality in the form of hypermedia systems like the World Wide Web.

Some of the problems inherent in closed hypermedia systems are being answered by the open approach that stores the links separately from the documents and allows them to be processed independently. This modular approach also promotes the creation of many different navigation tools, designed to assist the user in their tasks.

In Chapter 3, the Microcosm system will be examined, focussing on the open approach it has taken and the tools and systems that have been created within it.

Chapter 3

Microcosm

3.1 Introduction

Microcosm was originally designed to overcome a number of problems that were identified as being common to the majority of hypertext systems at the time when work on the system began in early 1989 (Fountain *et al.*, 1990)(Heath, 1992). Briefly, these problems were :

- The authoring effort required to create and maintain links in large sets of documents.
- The closed nature of many hypertext systems.
- The constraints that proprietary document formats impose when considering the portability of information from one hypertext system to another.
- The problems of dealing with information stored on read-only media.

To solve these problems Microcosm was designed to be an open hypermedia system whose openness shows itself in a number of different ways. These inlcude :-

- Open architecture
- Open message format
- Open linking
- Open functionality

3.2 Open Architecture

Figure 3.1 illustrates the Microcosm architecture. Four distinct types of module can be identified, the document viewers, the Document Control System (DCS), the Filter Manager System (FMS) and the filters. Underlying all these modules is a Document Management System (DMS). Each of these components will be described in turn.

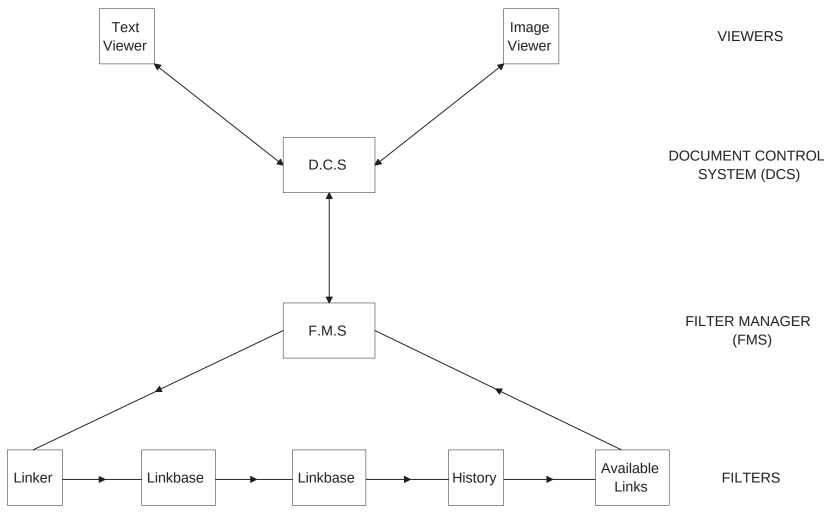


Figure 3.1: The Microcosm architecture of viewers and filters.

3.2.1 Viewers

Microcosm viewers are organised by document type, allowing different viewers to be plugged into the system. In order to promote openness by allowing any arbitrary document type to be used with Microcosm, three classes of viewer are catered for.

Fully aware viewers. Specially written for the Microcosm system, these viewers understand the Microcosm message passing protocol, and may have specific Microcosm features attached to them such as the ability to highlight button links within documents.

Partially aware viewers. Some applications may be customised to allow for some degree of Microcosm awareness. An example would be Microsoft Word, which has an extensive macro language. Using this language, menu items were added which allowed the user to select a piece of text in the document and then select 'follow link' on the menu, triggering the hypermedia activity.

Unaware viewers. Some media types can only be viewed in specialised applications, which offer no services for customisation. The DCS can launch these applications to view the documents but is unable to communicate with the application once it is running. To try and provide some degree of functionality, Microcosm can be set to monitor the clipboard. If the user sends a piece of text to the clipboard, Microcosm can treat it as a selection and use it as part of a "follow" link message. This enables a small degree of hypermedia functionality even from completely unaware applications.

3.2.2 Document Control System (DCS)

The Document Control System, or DCS, is responsible for starting up the document viewers as and when they are required. It also serves as the main communication point for the viewers and all messages passed to, or from, the viewers are routed through the DCS. The DCS also sends messages to and receives messages from the Filter Manager; providing the link between the document interface and the underlying hypermedia functionality.

3.2.3 The Filter Manager (FMS)

The Filter Manager maintains a chain of filters, which provide the core hypermedia functionality of the Microcosm system. It serves as a communication channel between the DCS (and therefore the viewers) and the filters. Although shown in the diagram as a chain where filters are connected directly to each other, the messages are in fact routed via the Filter Manager to the next filter in the chain. This aids modularity by removing the need for the filters to be aware of each other. Filter chain topology and message passing are discussed in more detail in later sections.

As messages are passed to the Filter Manager, it passes them down the filter chain. Any messages passed by the last filter are sent by the Filter Manager to the document control system. These in turn, might be forwarded to the appropriate viewer if that is who the message was intended for.

Any messages generated by filters in the chain are passed on to the filters in the chain *below* the filter which generated the message. Any filters in front of the sending process will not receive the message. This does prohibit a filter from sending a message to a filter lower down in the filter chain and receiving a reply to that message. Although restrictive, this practice does greatly reduce the chance of message loops appearing within the chain.

3.2.4 Filters

The filter processes provide all the hypermedia functionality in Microcosm. When a filter connects to the system by registering with the Filter Manager, it is placed within the filter chain. It then sits there receiving messages, which it can examine, and according to the message type, handle in a number of possible ways. It can :-

- ignore the message and pass it on.
- carry out some processing and then pass the message on.
- carry out some processing and then discard the message.
- carry out some processing and modify the message.
- carry out some processing and create new messages.

All the filter functionality can be structured around this message passing scheme. More comprehensive examples of filters are given in later sections.

3.2.5 Document Management System (DMS)

The Document Management System serves as an underlying database of documents which can be accessed from all parts of the Microcosm system. It was added to the system for two principle reasons.

Firstly, it allows a level of abstraction from the underlying file system by providing unique identifiers for all of the documents in the system. The rest of the Microcosm system uses these identifiers to refer to the documents rather than their physical location in the file system. This means that documents can be easily moved around using the Document Management System and any links or references to the document will remain valid. Maintaining link integrity is one of the major problems of open hypermedia systems and anything that helps with this task should be exploited.

```
\Action DISPATCH
\DocType TEXT
\DocName 100.23.34.54.56.34.23.12352
```

Figure 3.2: An example Microcosm message to dispatch a document.

Secondly, the level of indirection between the Microcosm system and the file system allows for the modular inclusion of distributed files. Initially, Microcosm only dealt with files on the local file system, but through modification of the Document Management System only, remote access to files held on a network, or even on the web can be seamlessly incorporated without major recompilation of the system (Hill, 1994).

3.3 Open Message Format

The Microcosm message passing system provides a very flexible and powerful method for communicating between modules in an open system. It does however have its quirks and idiosyncrasies. Because it is an underpinning tool in the research described later, it is worth covering it in some detail here.

At the simplest level, a Microcosm message is a series of tag and value pairs. The content of messages is open in that a message may contain any arbitrary tags. Although there are no required tags in a message, virtually all Microcosm messages will contain an action tag that indicates the type of the message. Example action tags are *FOLLOW.LINK*, *CREATE.LINK* and *CLOSE.FILE*.

Figure 3.2 shows a typical Microcosm message. The example message contains three tag value pairs. The first is the Action tag. The *DISPATCH* action is a request to launch a document. If the DCS receives this message it will launch a viewer to display the document. The DocType tag indicates to the DCS what type the document is and it can choose the appropriate viewer to display the document. The DocName is in the form of a UniqueID that is mapped by the Document Management System (DMS) to a path and filename.

The message format is open in that any arbitrary information can be placed in a message. Tags might be added by specific modules and contain information which only they are in a position to interpret. The other modules in the system will happily ignore this information.

There are a number of problems that can occur with this form of message passing system in conjunction with a filter chain approach.

3.3.1 Some Messages Only Need to be Processed Once

Because of the way messages are routed through the filter chain, it is possible for two filters to process the same message. In some cases this is inappropriate. For example, when a new link is created a message is sent through the filter chain asking for the link to be stored in a linkbase. If there is more than one active linkbase in the chain and the message was passed through as normal, the link would be stored in every active linkbase. This is clearly not the desired behaviour of the system. In order to ensure the link is only stored in one linkbase, a number of possibilities exist :-

- The first linkbase to receive the message processes it and then removes the message from the chain. This is what happens in the current Microcosm system. The user knows that any links created will be placed in the first linkbase in the filter chain. In order to add links to the other linkbases, the order of the filters needs to be altered.
- The first linkbase to receive the message processes it and stores the link. Before passing the message on however, it is modified to reflect the fact that the link has already been stored. This provides additional notification for the other filters which might process the message in a way that doesn't lead to the creation of the link, for example notifying the system that the link has indeed been created (which will be expanded on in the next section).
- A third alternative would be to target the link creation message at a particular linkbase. This could be achieved by adding an additional tag into the create link message which indicates which linkbase should create the link. When this linkbase receives the message and recognises its name it stores the link. Other linkbases in the chain would ignore the message. This method does not require the message to be modified and allows it to continue the length of the filter chain. It does however require an addressing system for filters.

3.3.2 There is No Way of Knowing if there is a Recipient

One problem with this un-addressed message passing is that there is no way of knowing if there is a recipient for the message. A create link message can be sent by the link creation tool, but if there is no linkbase in the chain the message will pass along the whole chain, being ignored, and finally disappear. Unless the message has a response message associated with it, the link creation tool will be unaware that the link was not stored. There are a number of approaches to alleviating this problem :-

- The ability to reply directly to the sender of the message could be implemented. Once a message is processed, the processor replies to the sender indicating that it has carried out its task. This could result in a number of replies for a single message if the message is processed by more than one filter. A direct reply would be required to circumvent the natural topology of the filter chain, which precludes messages passing to filters in the chain which precede the sender. Without knowing how long it takes to process the message, the sender would have to guess how long to wait before assuming the message has not been processed.
- Unprocessed messages are returned to sender. If the filter manager is made aware of what messages are removed from the chain when they are processed, it could make sure than any messages that arrive at the end of the filter chain unprocessed are returned to the sender of the message.
- Filters register what messages they will process and this information is made available to other processes in the system. The link creation tool could then check that something exists to process the create link message and store the link. This has been partially implemented in Microcosm in that filters can register menu items for viewers to indicate what actions the user could successfully carry out. The registration process gives rise to alternative topologies that will be discussed in a later section.
- A final alternative would be to provide a dynamic start up system for filters. If the filter manager were aware of which filters could process messages, it could start up a filter to handle a message which currently has nothing to actively processes it. This scheme is exploited in the direct communication topology discussed later in this chapter.

3.4 Open Linking

One of the original distinctions between a closed and open hypermedia system was whether the links were embedded in the documents. In Microcosm the links are stored in separate link databases, or linkbases.

Examples of implicit links in the Microcosm system are generic links and computed links. In the case of generic links, the destination is fixed, with the start anchor being defined dynamically by the user. For example, the user selects the word 'Membrane' in a document and asks the system to find them some links. A link was previously authored as a generic link on the word 'membrane' so the link is matched as the source anchor. The name of the source document is irrelevant for a generic link, where only the selection is matched. The user is then returned the destination of the link, in this case perhaps a biology dictionary that contains the definition of a membrane.

Computed links can have a dynamically generated destination based on a user defined start anchor. Here, if the user selected the word 'membrane' and asked the system to compute links on the selection, the full text retrieval system will examine its index and dynamically create a set of useful destinations based on a correlation between the selection and items in the index. These links only exist at the time of calculation and if new documents are added to the system and the index regenerated the same query might not produce the same set of matches. In the Microcosm system, a full text retrieval process known as the computed linker (Li *et al.*, 1992) is used to generate implicit links whose end anchors are derived from a selection made by the user.

There are a number of substantial benefits of storing links separate to the documents.

The original documents are not modified. This removes any restrictions on document types and allows the users of the system to edit the documents using the most appropriate editors. Furthermore, as the links need not be embedded in the document it is possible to use read only media such as data on DVD-ROMs or CD-ROMs. This would not be possible otherwise without making copies of the data which could be modified.

Multiple link databases can be applied to documents. As the links are stored separately, many different linkbases can be created and applied to the same document set. For example, users might choose to have their own personal linkbase, which only they would use. As the links are not embedded, other users viewing the documents would not be presented with those links. Linkbases can also be used to provide different tutorials based on the same dataset. By choosing which linkbase to use, different paths will be presented through the data as the documents are viewed. This would require copies of the datasets if constructed using a closed hypermedia system.

Link information can be analysed separately. By having the link information separated from the documents rather than distributed throughout the dataset it makes it easier to carry out analysis on the information. This central information resource could prove a powerful tool for expert systems to use when proffering suggested viewing material.

There are a number of problems that can be identified with open linking strategies however.

Possibility of link inconsistency. Where link information is stored which relies on document content information, such as a character offset into the document, the possibility exists for the document to become out of synch with the links. By editing a paragraph in a text document a link's offset may change, or at worst, the source anchor for the link could be removed entirely. As the link only exists within the linkbase, the system might not be aware of this inconsistency until it tries to place the link. To overcome this, link aware editors could be created which update the linkbases to reflect the changes made to the documents, but with the possibility of many linkbases distributed throughout the system, even this does not provide a total answer to the link integrity problem. Issues of link integrity are discussed in much greater detail in the work of Davis (Davis, 1995).

Need to update local knowledge. While users are using the system they can be considered to view a snapshot of the information. If a link is added to the system which has a source in a document that is currently displayed, the system needs to inform the viewer of the changes that have occurred. This also applies to navigation tools that may need to be made aware of modifications to the underlying information. For example, a linkbase might wish to be informed that a document has been removed from the system. This would allow it to search its database and remove any links that refer to the document, or at least flag them to avoid returning them as the result of a query. The separation of link data from documents makes the update issue more difficult.

3.5 Topology

A number of different topologies have been tried with the Microcosm architecture. Each has its own advantages and disadvantages. The sections below describe the topologies and look at some of the pros and cons associated with them.

3.5.1 The Basic Filter Chain

The basic filter chain topology is the one discussed in section 3.2.3. As shown in figure 3.3, the filters are placed in a single chain, with the messages passing down the entire chain. Some messages will invariably be blocked by the modules and other messages spawned during the processing. Rather than discuss this topology in detail, it will serve as a reference to compare the alternatives below to.

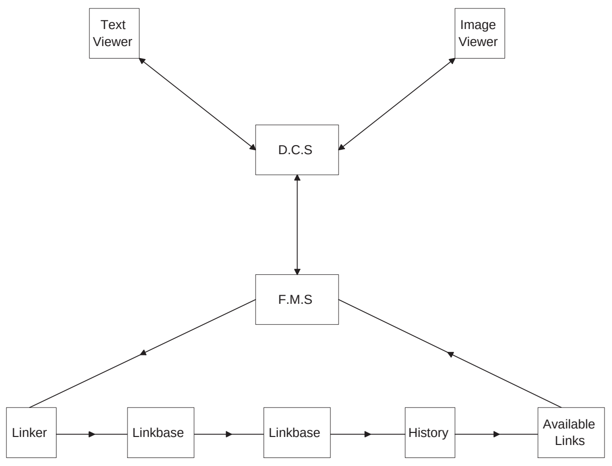


Figure 3.3: The original filter chain topology used in Microcosm.

3.5.2 The Direct Communication Topology

As the name implies, the direct communication model removes the filter manager entirely and allows all the modules in Microcosm, be they filters or viewers, to talk directly to each other. With no permanent lines of communication, Figure 3.4 illustrates some of the possible communication patterns that might take place using the direct communication model. The labelled arrows indicate the messages being passed between the modules.

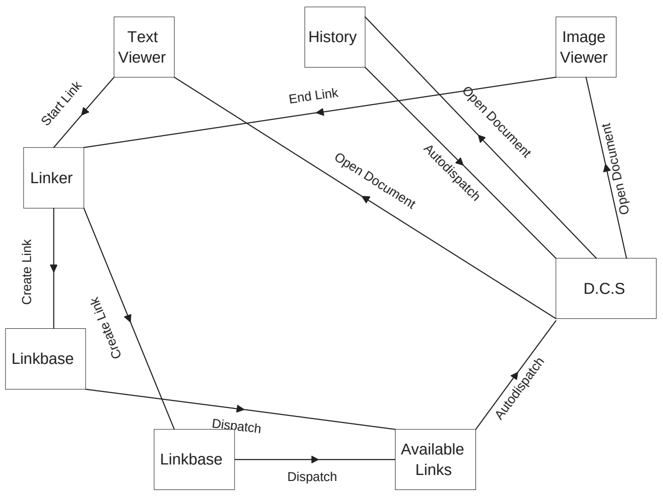


Figure 3.4: The direct communication topology.

The Microcosm modules register with a central communications broker that routes the messages. As part of the registration process, the modules supply a name, and various topics on which they wish to communicate. These roughly equate to the message actions of the standard Microcosm set-up. As part of the registration process the modules are assigned a unique identifier. Once connected to the system the modules can communicate with each other via the broker.

There are four ways in which processes can communicate using the direct communication model :-

Broadcast. This involves sending the message to every process that is registered with the communications system. A *SHUT.DOWN* message would be a good example of this, being used to clear up when the session is finished.

Delivery by topic. This is where the message is passed to every process that has registered to talk about a particular topic i.e. *FOLLOW.LINK*.

Delivery by name. The message is sent all processes registered with the given name. The name could be obtained from the communication process, or might be included in a previously received message to allow for direct reply.

Delivery by unique number. Rather than use the process name, the sender might use the unique number, which is allocated at registration time. Where a process processes more than one type of message this can provide an easy way of differentiating between the received messages. The process can register multiple times and use the unique number to differentiate between the messages it receives. This is the only form of message passing which is guaranteed to be process to process as more than one process can register for a give topic or even name.

This topology greatly reduces the number of messages that are sent around the system since processes are no longer reliant on returning messages via the filter manager. The average distance a message travels is far shorter since filters that are not registered for the message will not receive it.

The system also allows for more sophisticated message passing. A filter might put a request out on broadcast and be answered by another filter in the system. Once the original filter has the reply, it can use the information contained within it to open a direct communication path with the other filter. In this way, the direct communication process can act as a basic form of brokering service, connecting processes that wish to communicate.

The direct communication model also allows for the Microcosm system to be used in different ways. With the original topology it was necessary for all the filters to be started in advance. With the direct communication system more fluid start-up could be achieved. The user could open a file in their file system that causes the Microcosm viewer to be launched. The viewer then asks for any button links in that document. When the broker receives the *FIND.BUTTONS* message and discovers it has no processes to deliver it to, it could consult a list of processes and start up the relevant process to handle the message. Providing default handlers are given for the possible messages, the system will start up as and when it is required.

Unlike the original filter manager topology however, it is harder to filter out messages from the communication system. Under the original topology, a filter could be written which sits between the linkbases and the dispatcher, which removed links according to some pre-determined heuristics to reduce the cognitive load on the user. One hundred *DISPATCH* messages would be sent by the linkbases and only ten would appear at the dispatcher due to the filtering mechanism.

Under the direct communication model, one hundred *DISPATCH* messages would be sent by the linkbases and all one hundred would arrive at the dispatcher. If the specialised filter were added to the system, it could register to receive the *DISPATCH* messages, but would receive them in parallel to the dispatcher and be unable to block them. To overcome this, linkbases could send *POSSIBLE.DISPATCH* messages, which could then be converted into actual *DISPATCH* messages by the filter if it deems them appropriate. The end result would be as before, however it is necessary to add new message types to accomplish this and the filter would need to run at all times in order for any of the links to arrive at the dispatcher. This runs against the open modularity approach of Microcosm, but would be necessary to replicate the filtering which is so easily achieved using the original topology.

The direct communications model is described in far greater detail in the work of Robert Wilkins (Wilkins, 1994).

3.5.3 The Hybrid Approach

The hybrid approach was initially adopted to avoid having to modify all the existing filters to the direct communication model. The direct communication system operates alongside the original filter manager topology and filters can use either system as a means of communication. This is illustrated in Figure 3.5.

By using the combined topology, filters are still able to sit in the chain and affect messages which pass through them, but they also have the ability to directly communicate with other processes when the chain mechanism is not needed. For example, linkbases might return buttons directly to viewers to bypass the end of the filter chain, but links would be sent using the traditional route to allow other processes to intercept and possibly filter them. This would assume that buttons are not to be filtered by filters further down the chain.

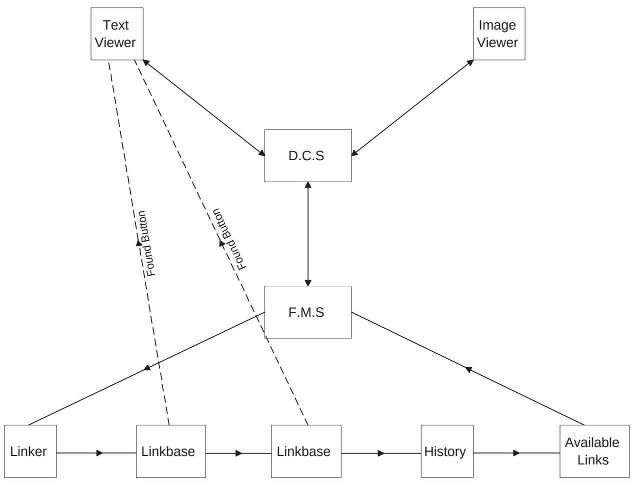


Figure 3.5: The hybrid filter chain topology.

By using a hybrid system, only those filters wishing to take advantage of direct communication needed to be modified, with the remaining system behaving as normal. This differed from the direct communication and the active filter manager discussed next, which both required all of the existing modules to be modified in order to work with the new topology.

3.5.4 The Active Filter Manager

The active filter manager was developed by Hill (Hill, 1994) as a means to reduce the number of messages passed through the system yet maintain a filter chain which permits new modules to intercept messages.

As with the direct communication topology, filters register with the filter manager and state which messages they would like to receive. The filter manager creates a separate chain for each message type, containing only those filters that have registered for that message. Filters can, and usually will, appear in more than one chain. Figure 3.6 illustrates the active filter chain topology.

When a message arrives at the filter manager it is passed through the appropriate chain. In his thesis (Hill, 1994), Hill shows how this can significantly reduce the number of messages passing through the system. Unlike the direct communication model however, a chain still exists. So to use a previous example, if the link creation tool sends a *CREATE.LINK* message, it will be passed to each registered

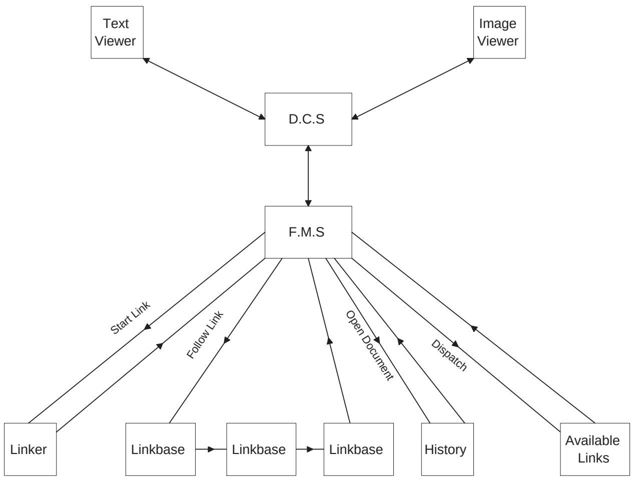


Figure 3.6: The active filter manager topology.

linkbase in turn rather than simultaneously as is the case with the direct communication model. The first linkbase can therefore store the link and discard the message ensuring that the same link doesn't get stored in multiple linkbases.

3.6 Open Functionality: Navigation Tools in Microcosm

One of the reasons for the Microcosm filter chain architecture was to allow the modular integration of navigational tools. This provides open functionality in that the user can configure the system to provide the facilities that they want without the need for recompilation. Microcosm formed a testbed that could be used to experiment with a variety of navigational tools and strategies. A number of navigational tools were implemented as filters for the Microcosm system. Each tool was constructed to collect one type of navigational information and provide this to the user through its own interface. I will briefly discuss some of the key tools provided. A more detailed description of navigation tools in Microcosm can be found in (Wilkins, 1994).

3.6.1 Direct Access Tools

A front-end interface is provided to the underlying Document Management System (DMS), which enables users to browse all of the documents in the dataset. The ability to add a hierarchical structure to the organisation of the documents allows the authors to create contents lists.

Furthermore, authors can create text documents to act as indexes into the data, with links providing quick access from an item in the index to its appearance in the content. The reverse can also be used to create glossaries, where generic links are created from a glossary term to its description in a glossary text file.

3.6.2 The History Tool

The Microcosm history tool keeps track of the opening and closing of documents by the system. The user can view a list that shows all of the documents that have been opened during the current session. Documents that are currently open are shown in bold, differentiating them from documents that have been closed.

Double clicking on one of the documents in the list will cause it to be re-opened. The list will update automatically to reflect this.

3.6.3 The Available Links Tool

The available links tool provides the user with a selection of links when their actions have resulted in a number of alternative destinations. The links might have been generated by linkbases or possibly by automatic link creation filters such as the computed linker.

The list enables users to go through all of the alternative destinations if they choose, rather than forcing them to backtrack and carry out the previous action which resulted in the list of links.

3.6.4 The Mimic Tool

The mimic tool of Microcosm was designed to help authors to construct guided tours, or trails, through the material. The author can generate a mimic tour either by example, or by explicitly choosing the documents and the order in which they are shown.

A tour consists of a series of documents that can be automatically triggered on a timer as in a slide show, or that can be stepped through by the user using a simple console. Mimic tours are treated as documents by the system, so they could be the destination of a link if desired.

3.6.5 Local Maps

Microcosm local maps provide the user with a simple view showing an icon representing the current document in the centre and icons for all documents that can be reached from this document by links arranged in a circle around it. By clicking on any of the document icons on the map, the document will be launched in an appropriate viewer and the focus of the map will shift to the new document. To reduce the complexities of the map only documents one link away from the current document are displayed.

3.6.6 The Advisor Agent

The Advisor agent was developed by Robert Wilkins as a means to integrate the information from a number of different navigational tools into one coherent piece of advice (Wilkins, 1994). Each of the navigation tools in Microcosm concentrates on collecting one type of navigational information and presents this information, or a summary of it, to the user in its own interface.

The aim of the advisor agent was to pool this information and provide it to the user through one unified interface. It achieved this by sending messages to the various navigational tools, requesting a rating for a particular document based on how important the tool considered it. For example the history filter would respond with negative ratings for documents that the user had already seen. The linkbase would give a positive rating for documents that had large numbers of links that started or ended with the specified document.

All of these ratings were collated by the advisor agent and presented to the user as a list of documents that had the highest rated document at the top. The user could weight the advice given by the different navigation tools, perhaps increasing the weight of the advice from the history filter if they felt they were unlikely to need to read a document twice.

3.7 Problems with the Microcosm Architecture

By virtue of its implementation in the Windows operating system, Microcosm's interface consists of a number of disparate windows in which information is presented to the users. The architecture makes it difficult to present a coherent interface to the user, as there is no communication between interfaces and no underlying strategy. A look and feel has been attempted through use of icons and dialogue styles but this is not enforced by the system.

The Advisor agent discussed previously, does serve to highlight another important issue. For a hypermedia system to be truly open, the information it holds needs to be accessible. The Advisor agent got around the problem by enabling the navigation tools to send their information directly to the Advisor agent but this is not a modular solution as other modules would not be able to take advantage of this information. The information was also normalised, to a number between 0 and 100, which potentially reduces the expressiveness by discarding information.

3.8 Conclusions

The Microcosm system was developed as an architecture to research issues and problems with open hypermedia. The Microcosm architecture tackled a number of aspects of openness, including open architectures, open linking services and open functionality. One of the problems encountered with the Microcosm system was that the interface was fairly ad hoc, as each module in the system was responsible for its own interface.

Chapters four and five of this thesis will examine screen management issues and describe a new architecture which helps to integrate the diverse interfaces of the Microcosm system into better managed, more user friendly interfaces. Although approached in terms of the Microcosm system, the principles could be easily applied to many windowing systems where individual modules provide their own interfaces within a larger system.

Chapters six through eight extend this interface abstraction one stage further by separating the underlying hypermedia information from the mechanisms used to display it. Most hypermedia systems remain closed when it comes to accessing the hypermedia information contained within the system. By allowing open access to the underlying information, the way is left open for novel interfaces to be provided as front ends to open hypermedia systems.

Chapter 4

Screen Management

4.1 A Brief History of User Interfaces

Before delving into the issues of screen management and in particular the management of multi-window environments, a brief history of Graphical User Interfaces (GUIs) will be covered.

Ironically, one of the first people to recognise the need for closely coupled human and computer interaction was the father of hypermedia Vannevar Bush. In his seminal paper 'As We May Think'(Bush, 1945), he says

> If the user wishes to consult a certain book, he taps its code on the keyboard, and the title page of the book promptly appears before him, projected onto one of his viewing positions.

In the 50's Licklider extended the ideas, referring to a 'man-computer symbiosis' (Licklider, 1960). His ideas suggested that the current way in which computers were used was restricting the human's ability to participate in the process. While computers were useful for dealing with problems that could be thought out and planned in advance, they were not able to cope with problems that are better suited to a trial and error approach where a user can correct the program as it comes across problems. This was a definite move away from the idea of batch processing where the program was produced, the data was fed into the computer, and then some time later a set of results would arrive.

In the early sixties Ivan Sutherland, while working at MIT Lincoln Laboratory produced a system called Sketchpad(Sutherland, 1963). The Sketchpad system pioneered ideas such as graphical representation of internal hierarchical structures, the use of a light pen for picture construction and the separation of the coordinate system in which a picture is defined from the coordinate system on which it is displayed. These principles and many others helped form the foundations of modern GUIs. Sutherland was also behind the development of new input and display technologies such as the forerunner of full immersion virtual reality (Sutherland, 1968).

One of the main applications that utilised the rapidly growing technology of interactive computing was that of word processing. A key figure in exploiting this potential was another hypertext visionary Douglas Engelbart. His Augment system allowed users to interactively construct and view documents using a keyboard and his new device, a mouse(Engelbart, 1963).

Perhaps the leap in technology that is most associated with Graphical User Interfaces is the work which came out of Xerox's Palo Alto Research Center(PARC) in the early seventies. Xerox developed the first personal workstations, which came complete with a display, keyboard and mouse and resembled fairly closely the personal workstations that we use today. Alongside this research, Alan Kay was developing his idea for the 'Dynabook'(Kay & Goldberg, 1977), a device which would be recognised today as a hand held computer.

All of this research eventually culminated in the release of the Apple Macintosh in 1984, which was the first implementation of the interface style developed by Xerox to be a commercial success. This interface style has been utilised by many of the systems we see running today, including Microsoft Windows in all its incarnations and the wide variety of X servers available on UNIX systems.

4.2 Multiple Window Environments

When people talk about GUIs, they are almost certainly referring to an interface where multiple windows can exist on the screen at the same time. Each window is constructed from a number of basic tools. These can include, but are not limited to, scrollbars, menus, buttons and text areas.

Each window will display certain information, referred to as the content. This content might be a piece of media, a text file or a graphic, or it might be a dialogue for interacting with the user, containing various buttons and lists. In nearly all windowing systems, the manipulation of the content of the window is handled by the application. A window manager controls the manipulation of the window itself. The window can be manipulated by moving it, changing its size or perhaps minimising it to an iconic representation.

The ability to manipulate the window gives rise to a number of potential problems however. Firstly, the window manager will not usually have any control or knowledge of the content of the window. The window manager may allow the user to move the window half way off the screen even though it obscures the content of the window. Similarly, if a window is resized, the content may no longer fit within the window and, if the application cannot rescale the content, will be cropped resulting in a partially obscured image or perhaps unreadable text.

A second problem is that each window will be dealt with on its own merits and any connection or interaction with other windows will be ignored. The movement of one window might obscure the contents of another. A window containing an image and a second window with text describing the image might be on either side of a screen with no visible connection between the two.

A final problem, and one that will be discussed in detail in the next section is that there may be different requirements of the interface under different circumstances. This can be referred to as the scalability of the interface.

4.3 The Need for Scalable Interfaces

Interfaces that are adaptable to users and environment are by no means restricted to computer software. A good example of a scalable interface is the interface to driving a car. Taking just two of the basic components, we have a steering wheel and a gear stick. Anyone who learns to drive in Britain will be familiar with sitting at the steering wheel and changing gear with their left hand. If however the person goes to the United States and hires a car, the interface will be different, the reason being that the in the USA cars drive on the right and the driver is seated closest to the centre of the road. The interface is different in that the gear stick is now located to the right of the driver. Everything else however, pedal order, gear selection positions, remains unaltered as the interface only needs to change to adapt to the new position of the driver.

If a car driver is unhappy with having to change gears, they can always get an automatic. Here again, the interface is modified to help the user. The gear stick becomes a simpler selector with perhaps with only park, neutral, and drive. The clutch pedal is now redundant and removed. The rest of the interface remains the same as it was before.

In the car example above, the interface is adapted to changes in environment and user ability. This change, though immediately noticeable to the user, does not require the underlying system to be changed. The principle of scalable interfaces becomes very applicable when dealing with multiple window environments.

4.3.1 Different Interfaces for Different User Abilities

When an interface is created, one of the key factors that has to be taken into account is the ability of the target user. A video player for example, is a complex piece of hardware containing video and audio boards for extracting a coded analogue signal from a magnetic tape and outputting a signal that can be understood by a television set. The user however does not need to know how all this works. They are presented with an interface, that when distilled to its basic components involves placing a tape in the slot provided and using a control which offers play, stop, forwards and backwards. This is enough to be able to harness the power of all of the complicated hardware underneath. Admittedly, the interface presented to the user will invariably contain dozens of other functions most of which the user will neither understand nor have any use for, but the basic interface is designed to be usable by practically anyone.

Interfaces to computer software are no different. They are often designed for the lowest common denominator. The problem with this is that when an expert user comes across such an interface it can get in the way of them carrying out the task they want to carry out. The classic example would probably be the Windows operating system. Ask any UNIX hacker what they hate most about Windows and chances are they will complain about the inability to 'get in there' and just type the commands. The abstraction of the interface is removing the ability of someone who knows what they are doing to get the job done more quickly.

To take a real world example, in the majority of current web browsers, when a link is clicked on by the reader by default the currently viewed document is replaced by the destination document of the link. This is essentially designed for ordinary users who are happier with only one document open at a time and don't want to have to continually juggle multiple windows on their desktop. Some users however might like to keep documents open, perhaps for comparison with the document at the destination of the link they have followed. In Microsoft Internet Explorer, a common browser, this can be achieved by right clicking with the mouse presenting a menu from which the user can then choose to open the document in a new window. This functionality takes more steps to achieve as it is not the default functionality. Were the browser a scalable interface the more experienced user who might choose to open a new window by default would be able to configure it always to open the document in a new window. This scalability does not appear to be available in any of the major web browsers. The interface behaviour has been decided at design time and cannot be altered.

Most interfaces are designed with a specific user in mind. Providing an interface that can adapt to a users abilities either dynamically or by user configuration is going to add to the cost of producing the interface. By separating out the interface more from the application, the possibility exists to supply a modified interface to users of differing abilities rather than having to supply whole new applications.

4.3.2 Different Interfaces for Different Environments

When designing and creating a hypermedia application, the author will have to take into account the environment in which the application is going to end up running. For example, the curator of a museum may decide to provide an information kiosk as part of an exhibit, which allows users to find out additional information about the items on display. The hardware for such a kiosk could well be a touch screen interface, but for this example we will assume a normal PC, which provides the users with a simple interface to browse the information. A typical interface might allow the user to choose from a limited number of choices which results in a short series of pieces of media on the subject. The user moves through the sequence by clicking on a button with *Next* on it. A simple system like this would allow novice users to quickly learn how to use the system and remove the added complication of peripherals such as a keyboard. This is important in an environment where a user new to the system will have to learn the interface and get the information they want in a very short period of time.

Having viewed the exhibit and interacted with the kiosk, the visitor may wish to take this information with them when they return home. The museum curator may decide to have a CD-ROM available for purchase in the museum shop, which contains the hypermedia application and perhaps additional information about the exhibit. At home however, the visitor has their own PC that has a keyboard and mouse and all the additional functionality that comes with it. In addition, when at home the user will also have more time to learn a more advanced interface. If the simple interface is hardwired to the kiosk application, a completely separate hypermedia application will need to be created for the CD. If however the interface is separated from the application, the same underlying hypermedia application can be used in both environments. An example of the scalability might be that if a keyboard is available the user might type a text search, whereas in the kiosk environment the user has to select from a series of options.[1]

If the content and functionality of the application can be clearly separated from the interface, only one hypermedia application need be created. The kiosk can then have a different version of the interface to the CD ROM application without greatly duplicating effort.

4.3.3 Different Interfaces for Different Hardware

In the example above, the museum may have opted for a touch screen system for their kiosk. This presents an additional problem in that even if the interface designer wishes to only have one way of accessing the content, the interface will need to be different depending on the hardware.

Another example of this is the hypermedia system used in their factory environment by Pirelli cabelling (Crowder *et al.*, 1993). The hypermedia system is used

[1]This is a greatly simplified case of course and it is possible that the material presented in the two cases might need to be different, however the underlying principle is still valid.

for fault finding and maintenance. The content is created and maintained using standard PC's in an office environment and can be accessed in this way. In addition to this however the shop floor operators also need access to the same information while they are at the cabelling machines on the shop floor. The approach taken was to use portable computers with a simple pen interface. The content being accessed is identical to that accessed using the office system, but the hardware requirements necessitate modifications in the interface. When using the portable computer, documents occupy the whole screen as shuffling around windows is both more difficult using the light pen and the operator has a much smaller screen to work with. Also, it quickly became apparent that sound is unusable as part of the interface for use on a shop floor where heavy machinery is operating.

By providing an interface that is abstracted from the content and scalable to the users requirements, these hardware issues were overcome. More details of the Pirelli system are given in section 5.9.2.

4.4 The Problem of Separating Content from Interface

Perhaps the greatest problem facing interface designers is separating the content from the interface. What we are talking about here is data abstraction. The object of this abstraction is to hide the low-level implementation from the user and present the information to the user in a more easily understandable form. To use a previous example, when the user presses the play button on their video recorder, they are not expected to know that this starts a motor inside the machine, which pulls the tape between the playback heads, generating the picture for the television to display. This process has been abstracted away to the user metaphor of press this button and the contents of the video will appear on the screen.

One advantage of data abstraction is that when a new technology comes along where the picture is stored on a different media for example with Digital Versatile Disc (DVD), the data abstraction is still valid and the interface for the user can remain the same. Press the play button and the picture appears on the screen. The fact that the underlying mechanism has changed is hidden from the user.

Sometimes however it is not easy to find a suitable data abstraction that keeps the content and presentation clearly delineated. Often, attempts to do so simply lead to confusing and muddled metaphors that are less useful than more accurate depictions of the data. A classic example would be the trashcan on the Macintosh operating system (MacOS). In an attempt to over simplify the interface, the designers of MacOS decided that to eject a floppy disk the user should drag an icon representing the disk to the trashcan icon on the desktop, a metaphor which they would be familiar with for deleting files. This proved to be far from intuitive. In practice, a simple eject floppy menu item might have been more useful to the user. A fuller discussion of this aberration can be read in (Erickson, 1990). Rather than solving a problem, the abstraction had only served to make the interface more confusing.

4.5 General Approaches to Interfaces

There have been a number of general approaches to interface design. A few such techniques are discussed in the sections below, with particular reference to their approach to data abstraction.

4.5.1 Window Managers

Window managers in their simplest form provide a data abstraction, separating out the window behaviour from the behaviour of the application underneath. Typically this will include functions such as moving and sizing of the windows but will not include control over the content of the window. The window manager will often provide facilities for positioning a number of windows such as tiling, or ensuring that all the windows are on the screen. Without knowledge of the content however, or the ability to provide constraints, the content can often be obscured or clipped by the window manager. This is due to the decision taken by the designers of window managers that the domain covered by the window manager is that of inter-application screen management and the general relationships between the windows. The content and application specific nature of the window is left to the individual application.

4.5.2 User Interface Management Systems (UIMSs)

User Interface Management Systems (UIMSs) emphasise the abstraction of the syntactic level (dialogue) from the semantic level (application), using notations for the dialogues, for example state transition diagrams or formal grammars.

Two major difficulties have been identified with UIMSs however (Took, 1990).

- It is often difficult to model complex dialogues where the user is interacting simultaneously with a number of different dialogue boxes. By providing formal languages with which to model the dialogue, the interaction becomes far more prescriptive for the end user.
- If the dialogue is abstracted from the underlying objects being manipulated problems can occur with semantic feedback. It is often difficult to maintain consistency between the state of the dialogue and the state of the underlying data.

The abstraction of UIMSs is concerned with the dialogue between the user and the applications. This formalisation fails to cover the interaction between applications or specific abstraction of the underlying information.

4.5.3 Toolkits

Toolkits provide a set of basic building blocks such as menus, scroll bars and dialogue boxes. The designer can construct their interface from these basic components or sometimes modify them to create new components. An example of a toolkit, is the swing toolkit that forms part of Java 1.2.

Toolkits also suffer from a number of problems (Took, 1990).

- It is often difficult if not impossible to create completely new classes of interface objects. More normally, existing objects are modified to meet the new requirements. This places the onus on the toolkit to provide a decent set of basic components to work with.
- In order to synchronise the presentation of the interface it is often necessary to provide global input and presentation objects which pass interactions down to the lower level components.

- Sometimes toolkits do not provide enough abstraction between the interface objects themselves and the underlying information they are controlling.

The data abstraction of toolkits tends to concentrate on the basic functionality building blocks of applications. Interaction between applications or the user and the application are left to the individual functionality of the applications. Where the toolkits can fail is where the attempted commonality forces the designer to use tools that are not quite appropriate for the task, such as the many differing uses of the trashcan icon on the MacOS desktop.

4.6 Specific Systems

Aside from the general approaches to screen management discussed above, a number of systems have been devised which seek to tackle some of the more specific issues of interface design and screen management. Some such systems are briefly discussed in the sections below.

4.6.1 Aquanet and VIKI

The interface to the Aquanet system (Marshall & Shipman III, 1993) was a direct rendition of the underlying structure of the information in the system. The users directly manipulated instances of objects in a shared space. The system allowed multiple references to the same object to exist within the space.

Aquanet objects include the concept of composite objects, which represent a group of objects. This helps reduce the complexity of the space, with users being able to investigate the contents of composite objects if they wish to. The representation of the space is relatively informal, with no automatic grid or alignment mechanisms. The interface can often end up looking like a true desktop with objects aggregated in piles or spread out in user driven layouts.

Whereas many spatial interfaces take over the task of organising the information for the user based on a wide variety of layout techniques, Aquanet simply provides the users with a set of tools which allow them to organise their information in a manner which they find intuitive to use.

VIKI is another spatial hypertext system that aims to support the emergent structure of the information it contains (Shipman III & Marshall, 1995). The user manipulates the graphical objects that are organised in hierarchical collections or workspaces. The system provides tools for the user which help identify implicit structure in the information and can suggest groups of information that might suitably be converted into an explicit collection. The tools of VIKI work on document analysis, extracting structure and knowledge from the document and presenting this graphically to the user in the interface.

4.6.2 Pad++

Pad++ (Bederson *et al.*, 1994) the successor to Pad (Ken Perlin and David Fox, 1993), provides an interface to structured information which uses zooming as a principle method of acquiring more information. The term Zooming User Interface (ZUI) has been coined to describe such a system. The user can move around a large planar information space where information is laid out spatially. The work builds on the multiscale interface research of Furnas et al (Furnas & Bederson, 1995). The user can zoom in on interesting information, leading to more detailed information being revealed. For example, if the user is viewing an annotated diagram, as they zoom in on a label on the diagram it might change to represent a more detailed description of the feature. Continued zooming might lead to whole documents on the subject. The principle applied here is that the screen can only show a certain amount of meaningful information at any given time. As objects become larger through zooming, the screen resource available for providing information about the object increases. This allows more information about the object to be displayed. This is illustrated in Figure 4.1.

The system provides a novel approach to screen management, and has been applied to a number of different application areas ranging from web browsing with PadPrints(Hightower *et al.*, 1998), to hypertext artwork fiction such as Grey Matters (Wardrip-Fruin *et al.*, 1997).

4.6.3 Style Sheets

When talking about the interface to the World Wide Web, it is important to remember that so much of the presentation is left to the browser to decide on. The

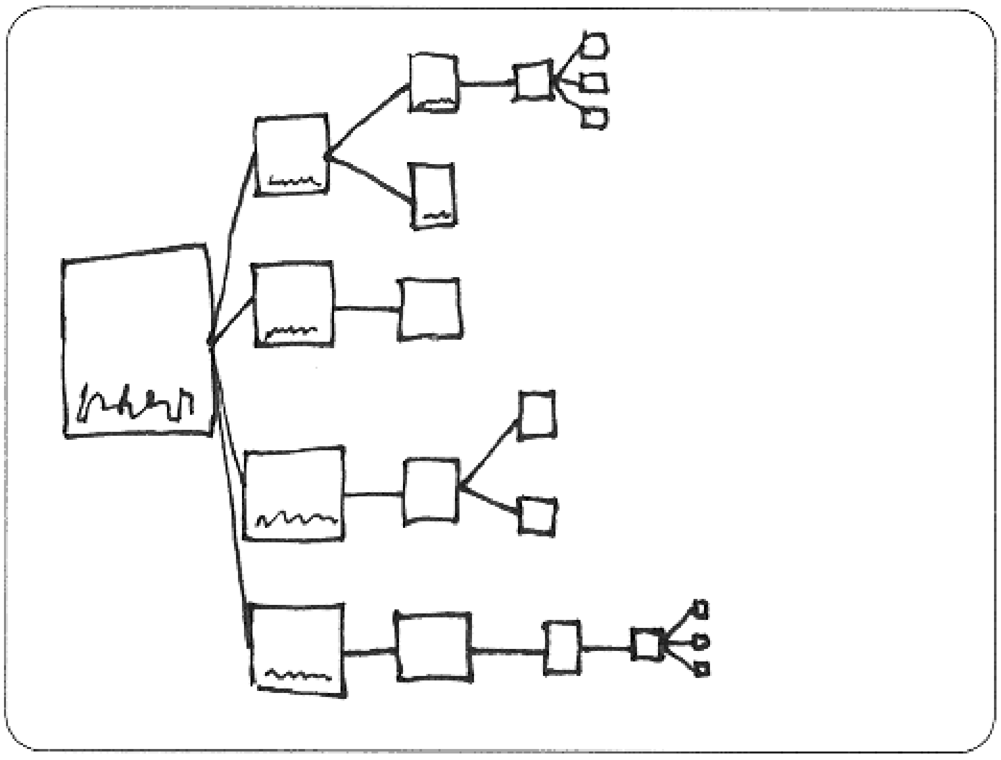


Figure 4.1: An illustration of the Pad++ zooming browser based on a screenshot in (Bederson *et al.*, 1994).

original version of HTML, HTML 1.0, had a very limited capability for specifying page elements. Essentially, the language described the structure of documents allowing the author to delimit paragraphs, quotes, lists etc. but leaving the rendering of the document to the browser. Where the author is only concerned with the content of the document that was fine but some authors wanted more control, and the desire that the document should look the same to all users (Andrew B. King, 1999). As early as 1993, the idea of "style sheets" was discussed as a way to formalise the presentation of HTML (Alan Taylor, 1999).

Hakon Lie sent a proposal to W3C, the World Wide Web Consortium, that described "Cascading HTML style sheets". The concept of cascading style sheets means that the browser uses a hierarchy of style sheets that are prioritised. This gives users the flexibility to override the style imposed by the author of the document. Cascading stylesheets have evolved into the specifications CSS1 which deals with the style of the document and CSS2 which is concerned with the layout. Both are supported by the latest versions of Microsoft Internet Explorer and Netscape Navigator, the leading browsers on the market.

Style sheets use common desktop publishing terminology, with formatting instructions to deal with margins, fonts, spacing etc. Unlike languages such as postscript and PDF however, stylesheets should be viewed as influencing the look of a document rather than specifying it, as style sheets can be overridden and different browsers will not always interpret the style sheet in an identical manner.

The separation of the content from the style is what gives style sheets their power. By changing one style sheet the author can change the look of a single page or perhaps an entire web site. This provides a simple clean mechanism for imposing a default style across a whole range of documents, ideal for imposing a company image on a web site with the minimum of effort.

Another benefit of this ‘openness’ is that individual users can use style sheets to affect how they view all documents. This is invaluable where a user has special requirements. For example a partially sighted user can use a style sheet to increase the font size of all the documents they view in order to make the text more readable. Style sheets have also been used to adapt pages for use by specialised speech plug ins, providing more usable interfaces for blind users.

One of the problems that has been identified with stylesheets is the issue of conflict resolution where multiple style sheets affect a single document (Jukka Korpela, 1999). The browser examines style sheets based on a precedence system. Without examining this in detail, the order of action is as follows.

1. author’s important declarations.
2. reader’s important declarations.
3. author’s normal declarations.
4. author’s implicit declarations expressed in HTML.
5. reader’s normal declarations.
6. user agent defaults.

Because both the reader and the author are aware of this precedence, a tug-of-war can ensue with both trying to ensure their preferences are used. An author can flag everything as important to make sure that readers can’t override what they consider to be the correct style. In this case the style sheet looses much of its openness as the author has essentially hard-wired the style. Faced with this, the reader might simply choose to switch off the style sheets and then any possible benefit is lost. Alternatively readers might flag their own style sheet as important, thus overriding any normal declarations made by the author. Even if authors and readers use the system as it was intended, it is not clear that the sum of all the style will necessarily lead to the best overall style. Clearly if the author specifies that the background should be blue and the reader declares that the text should be blue, the resultant document will not be easy to read.

4.6.4 Interface Agents

An interface agent is a process that operates within the interface, assisting the user with their task. One of the key criteria of an interface agent is that it can act autonomously rather than simply reacting to the user. The agent can affect the objects in the interface independently of the user. To inform its decisions the agent monitors the users inputs to the system over a period of time and modifies its behaviour accordingly. A context sensitive help system is one example of an interface agent. A more complex example is the Letizia system (Lieberman, 1997), which assists users in the task of web browsing. In recording the URLs browsed by users, a profile of user interest is compiled. The agent uses this information to search the web to find information containing similar content for the user. This information is presented to the user in a separate section of the web browser.

4.6.5 Elastic Windows

The elastic windows approach of Kandogan and Shneiderman (Kandogan & Shneiderman, 1997) is based on three principles:

hierarchical window organisation This allows users to group all of the windows related to a single role or activity into one area. By using nested groups, sub tasks can be gathered together. The border colours of the windows is used to differentiate between the groupings.
The use of hierarchical grouping allows the folding of whole hierarchies down to individual icons providing a very scalable approach to screen management. Users also have the ability to make any single window in the interface full screen. This might be an individual task window, or a hierarchy representing a particular role.

space-filling tiled layout Giving the system its name, the windows will stretch elastically to fill the available space. The non-overlapping approach was adopted to avoid wasted space and also potentially disturbing overlaps, which can obscure content.

multi-window operations In Elastic Windows, a window operation can be applied to a whole hierarchy, with the action propagating down to the lower level windows recursively. This allows whole hierarchies to be resized or closed with a single user action.

4.6.6 Synchronized Multimedia Integration Language (SMIL)

The Synchronized Multimedia Integration Language (SMIL), pronounced 'smile', was developed by the World Wide Web consortium with the aim of allowing a broader audience to author multimedia presentations using the Web (Hoschka, 1998).

SMIL is a text based language, built on the Extensible Mark-up Language (XML). This gives it the advantages of being easy to write, only a text editor is required, and it is also easy to make it compatible with HTML, which shares a similar base.

The SMIL language separates the content of the presentation from the layout by including two separate sections in a SMIL document. The content of the presentation is structured using commands which describe how the content is laid out temporally, i.e. media objects can be presented in sequence or in parallel and synchronisation can be achieved using begin and end timing information.

The layout of the presentation is covered in a separate area of the SMIL document, where regions can be specified which can then be used to position the media objects on the screen. Region information can include the top and left position of the region, height, width and z-order information, allowing regions to overlap each other. The media objects of the presentation can have anchors and hyperlinks associated with them allowing users to browse associated material, pausing the presentation if appropriate.

One of the more interesting aspects of the language is the ability to provide system dependant versions of documents to cater both for differences in hardware capability and also possible disabled users. This relies on the presence of alternative media objects that can be chosen according to the active system state. For example, if a machine is suffering from network bandwidth problems it might choose to display a still image and audio commentary rather than try to stream digital video across the network. The same facility can be used to supply subtitles to audio and video for users with hearing difficulties. In this case, the captioning would be requested by the user rather than being based on system monitoring.

4.6.7 The Apple Event Object Model

The Apple Event Object Model (AEOM) consists of a number of standardised sets (or suites) of messages and abstract data objects. They are designed to cover both the inter-application exchange and manipulation of data, and the external control of applications.

There are two defined suites which applications are encouraged to adhere to, known as the required suite and the core suite. The required suite is a set of four basic messages that every Macintosh application is required to support. These are: Open Application, Open Document, Print Document and Quit Application. The core suite must be supported by every AEOM compliant application. The messages allow chiefly for the retrieval and modification of data within applications, plus closing documents, and the making of selections.

Other suites are defined for specific data types, including text, pictures, tables, Quicktime and sound. Applications are of course allowed to define their own suites, which can then be utilised by other applications.

Most suites define a set of objects to cover the data type that they handle. An application's data is arranged as a hierarchy of these objects. For example: at the top of the hierarchy is the 'application' object. This will have a set of 'window' elements. A window within an application can be specified by name, or by order: window 1 will be the front most, window 2 the second from the front, etc. A window has a number of properties, including name, position, bounding rectangle etc., and selection.

An external application can therefore retrieve the position and size of a window by calling the relevant apple event. This would allow limited screen management functionality by allowing external processes to affect the positioning of other windows on the screen.

Like most protocol's apple events is only as good as the applications written to use. If applications choose not to support it then many of the benefits of the system are lost. With respect to the overall management of screen resources, this would only be possible if all of the on screen applications were compliant with the protocol.

4.7 User Interface Tools

The systems described above all attempt to deal with the whole problem of screen management. There have also been a number of research projects that have developed tools that can be utilised by such screen management systems. A number of these projects are briefly covered below.

The work of Harrison and Vicente suggests that the use of transparency in user interfaces can make better use of the screen space available(Harrison & Vicente, 1996). In an application where a user works on a main view but has various tools available, the use of transparency for the tool windows made them less distracting and enabled more of the main view to be seen.

Mereu and Kazman showed in their research that 3D interfaces used by visually impaired users can be enhanced by using audio(Mereu & Kazman, 1996). Although the visually impaired users took longer to carry out the tasks they were able to do so with the same degree of accuracy as a sighted user. This was achieved through a simple modification to the interface and did not require the underlying application to be modified.

The work of George Furnas concerns the use of fish-eye views and space-scale diagrams to help present the maximum amount of information in the minimum available screen area (Furnas & Bederson, 1995). This approach has been used in many information layout systems such as the Document Lens (Rao & Card, 1994) and the Perspective Wall (Mackinlay *et al.*, 1991), both of which will be discussed in more detail in Chapter 6. The underlying principle of such systems is to allow the focus elements of the information to occupy more screen space than the information that is of less relevance.

All of the tools above may well be useful tools for interface designers to use however because of the closed nature of interfaces, it would be necessary to re-implement the tools within the chosen interface. Since these tools are in essence generic and applicable to a wide range of interfaces, modularity of interface would be highly desirable enabling interface designers to take these tools and plug them directly into their interface without needing to modify the underlying application.

4.8 Open Hypermedia Interface Issues

Hypermedia systems may often be composed of a number of different interfaces with a commonality of purpose. Documents in the system must be presented to the user and the variety of media types available may necessitate a number of different document viewers. Navigation within the system may require the user to interact with the system using an interface that allows them to type in searches. Alternatively the user might wish to browse the hypermedia structures using hierarchical lists or perhaps a graphical browser. The results of queries and searches will need to be presented to the user in a way that allows them to make a choice based on the results. Other navigational information such as lists of previously viewed documents might be provided for the user and this again will require an interface.

In a closed hypermedia system all these interfaces can be combined to provide a single interface for the user. The best example of this is a web browser. The browser displays the documents, can be used to provide forms for the user to enter queries, and indeed displays the results to the user with links giving access to the documents. History lists can be accessed from menus along with bookmarks and various other navigational tools. Because all of the functionality is fixed, the interface can be easily integrated.

With open hypermedia systems, their very nature prevents the interface being designed as a single unit. Where the openness of a hypermedia system extends to the ability to add new media types and new functionality, unless the interface is rebuilt with every addition, the interface needs to take on a new form.

In Microcosm for example, each media type has its own viewer associated with it. This might be built specifically for Microcosm, or could just be a third party application. Because of this, each document is viewed in a separate window. The ability to plug in new navigational tools also makes it difficult to present a single interface. The history information is presented by the history tool in its own dialogue box. The Microcosm system encourages a common look and feel to the interfaces, with dialogue boxes using similar styles, fonts and icons. Ultimately though, the systems interface is a collection of windows which are not managed as single application but instead left to the operating systems window manager to organise. The interface is closed in that new modules are responsible for sorting themselves out and no support is given for creating a cohesive interface for the application as a whole.

4.9 Conclusions

Screen management issues have been around since computers first placed text onto monitors. Initially the problem was simply getting as much of the text onto the screen at a given time and allowing the user to move around the text as easily as possible.

The windowing systems of GUI interfaces added a new dimension to this problem. The information from different programs running in the operating system could appear in their own windows, which could be moved around and indeed overlapped by the user to give the appearance of much more information being present on the screen than there actually was. By adopting a ‘desktop’ metaphor, an attempt was made to ground the user in a familiar environment and explain why the information presented to them looked so cluttered and confused.

A number of different approaches were taken to alleviate these problems and to cater for the fact that different users on different systems had different requirements. Open hypermedia systems are often modular in their nature and screen management problems can be identified within the application itself.

The next chapter introduces a novel approach to overcoming some of these problems and suggests an approach to open interfaces that enables the designer of the interface to build tools that can exist separately within the system and do not prohibit the addition of new modules.

Chapter 5

The Design and Implementation of SHEP

5.1 Screen Management and Microcosm

The Microcosm system was designed and implemented as a modular architecture. This provides flexibility to add and remove utilities from the system on the fly, and provides an extensible framework for further research and development. The modularity also extends to the document viewers, with separate viewers being used to view documents of different types.

Many of the components of the Microcosm framework have visible interfaces, be they windows displaying documents or dialogue boxes allowing the users to change the options on a module. Each component is in charge of managing its own interface under the umbrella of the Windows operating system. The Windows management built into the operating system treats each window as a separate application since the modules of Microcosm form no hierarchy in the normal parent/child sense.

Some interfaces within the Microcosm system do carry out their own screen management. A number of viewers provide the user with the option to save the size and position of the window displaying the current document. When the document is viewed again, the previous size and position is recalled and the window displayed as it was before. This simple system provides an author with the ability to set the default window size and position for the documents of an application. Once a user views a document, they will then have the ability to save the window at a different size and position. A similar principle extends to some dialogue interfaces that the user can configure to their own preferences.

5.1.1 Problems with the Basic Microcosm System

There are a number of significant disadvantages to the approach described above however. These can best be illustrated using some examples, in some cases from actual applications implemented in the Microcosm system.

Firstly, there is no logical connection between the various windows on the screen even though they may form part of the same application. For example, a user selects a piece of text in a viewer and asks the system to find some links. This might result in a new link appearing in a window on the other side of the screen. When the events occur in quick succession the user may intuitively connect the action of asking for links with the arrival of the new links in the other window. If however the link doesn't arrive for some time, the user may not be sure whether the link has arrived as a result of their action or perhaps as a result of some other event unseen by the user.

Secondly, Microcosm is tied in to the windowing metaphor of the operating system, where a user can overlap windows, move them partially off of the screen and minimise and maximise them. This is often too complicated for less computer literate users and they may well want a more organised approach. By having modules manage their own interfaces, any new metaphor or approach must be implemented in all the modules which have interface components.

Furthermore, there are no facilities within the system to synchronise the display of documents. When images are displayed as part of an application, the author may wish to display a text commentary that describes the image, or perhaps gives the copyright clearance for the image. The author can set the initial positions of the image and text documents so that they complement each other, but once the user moves the image window, the association is lost. The association is merely an artefact of their initial position rather than a semantic connection that is maintained by the screen management system.

It may also be useful to synchronise the contents of two document viewers. The user might be viewing a facsimile image of a document in one window, and have a transcription of the document in a second window. As they scroll down the transcription, it would be useful if the image was scrolled to display the relevant portion of the facsimile. Without the ability to externally control the viewers, this is not possible.

What is needed is an interface that can be customised to suit the users abilities, yet doesn't require modification to the underlying application with each new screen management tool added. Being an open hypermedia system, Microcosm is ideally suited to this, as the architecture is designed to be flexible. Additions are therefore needed that extend the Microcosm framework to include an open approach to screen information in the same way that it exists for linking.

5.2 Window State

In a windowing system, each window can be said to have, at any given time, a state. This state represents a number of properties, or attributes, of the window. These attributes should provide all the information necessary to exactly recreate the window as it currently appears. As such, the information should include both external state information and internal state information.

5.2.1 External State

Two such properties might be the height and width of the window. Since the window can be moved without changing its height and width, a position must also be included which defines where the window is in relation to the desktop. This position is usually given relative to the top left hand corner of the desktop, (0,0) in the coordinate space. These state attributes are illustrated in Figure 5.1

Window size and position can be considered the external state of the window. In most window management systems, this is the only information used. When considering a 2D layout such as the desktop, the important question is 'How much screen is the window occupying?' What it does with the area of screen is less important.

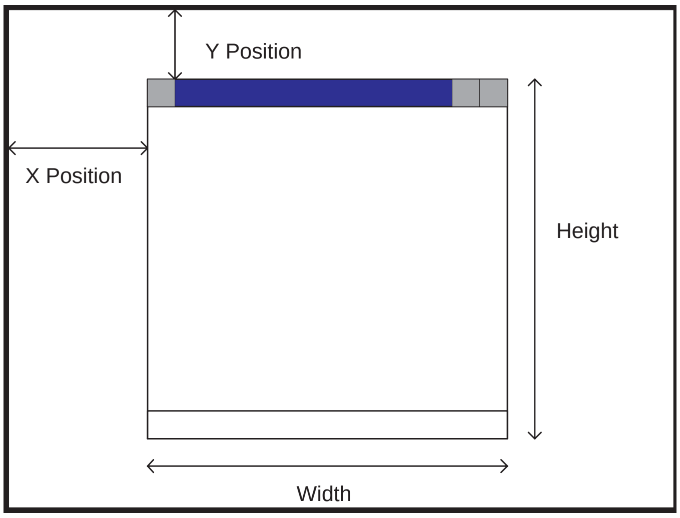
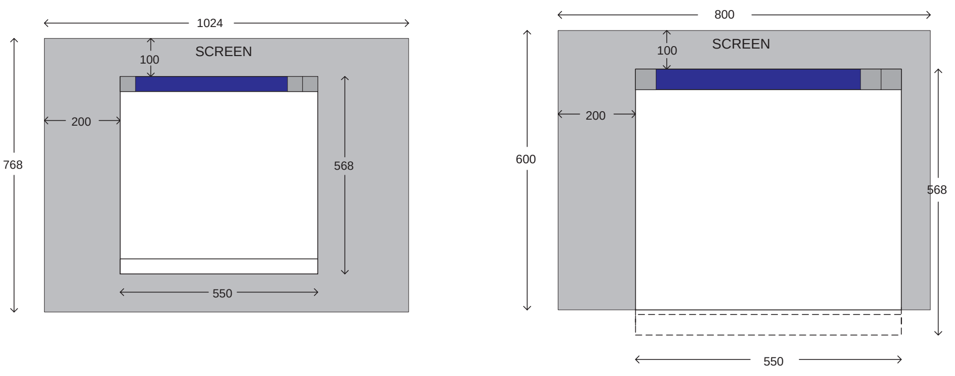


Figure 5.1: The external state of a window component.



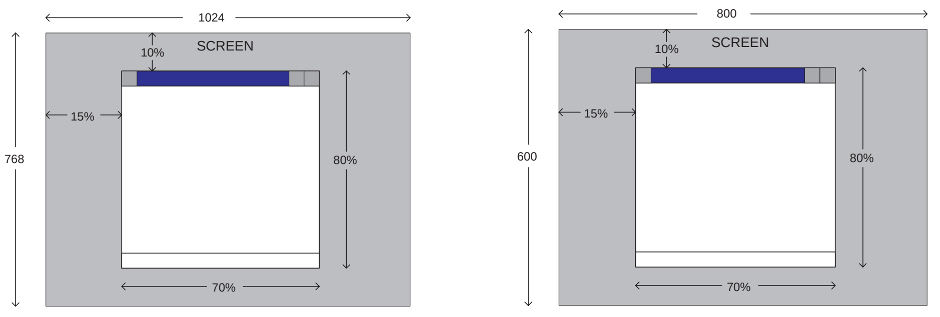
(a) The viewer before the screen resolution is changed.

(b) The viewer after the screen resolution is changed.

Figure 5.2: Using actual size and position information.

The size and position of a window need not be expressed in absolute co-ordinates and measurements however. Indeed, where the size of the desktop can change, using absolute measurements can cause problems. Consider the case where the window manager stores information about the windows external state in order to recreate the window exactly, next time it is used. If the user modifies the size of the desktop, for example by changing the screen resolution, the window that is recreated shares the same state as the previous window but appears markedly different, as shown in Figure 5.2. Because the height has been maintained the window no longer fits on the screen and part of the window has been cropped.

To overcome this, rather than storing absolute measurements, relative measurements could be used. In the example illustrated in Figure 5.3, the height and width are expressed as a percentage of the screen. The position is also expressed



(a) The viewer before the screen resolution is changed.

(b) The viewer after the screen resolution is changed.

Figure 5.3: Using relative size and position information.

relative to the overall size of the screen. When the window is recreated at the lower screen resolution it appears the same although in practice the absolute dimensions of the window will have shrunk.

Although in some cases this would seem like a better solution, the use of relative measurements makes an important assumption about the contents of the window; that they are scalable. Whether this is true depends on a different set of attributes of the window, its internal state.

5.2.2 Internal State

The internal state refers to attributes of the content of the window. If a window is constructed from a number of panes (or areas), this might include their relative positions. In the case of image viewers a magnification ratio, or zoom factor, might be an attribute of the internal state. Unlike external state, the internal state is much more subjective and a fixed set of attributes cannot be easily derived.

One function of the internal state would be to allow a window to be recreated exactly as it was before. If this were taken as a goal, then for document viewers, properties such as the position within the document would be needed. Depending on the media type of the document, the position might be represented in different ways. For a text document, an offset into the file might be recorded which locates the top visible line of the document. For an image that is larger than the window displaying it, the visible area might be stored. In the case of video, a time offset or frame offset would be appropriate. There are other examples of attributes where the value is specific to the particular type of window and it will be important to differentiate between them.

5.2.3 State Attributes

A number of different types of state attributes can be identified in Microcosm. It is perhaps useful to define categories that the attributes can be grouped into. It should be noted that some attributes fall under a number of categories.

General Attributes: These attributes can be identified in the state of all windows. Most external state attributes fall into this category such as size and position.

Representation Specific: Some values of some attributes will need to be handled differently depending on the type of window. For example, the offset attribute that indicates which part of the current document is being viewed will differ depending on the document type. In the case of text it might be a character offset whereas in the case of an audio file it might be a time reference. The attribute value can only be understood in the context of the type of document it is applied to.

Media Specific: Some attributes will only be applicable to certain types of document. A volume attribute would have little meaning in the context of a text document, and likewise font would be of no importance if video were being displayed. These media specific attributes are only used in state information when applicable.

The question of where state ends and content begins is an interesting one. If the goal is to be able to recreate a window exactly, then the content or at least a pointer to the content of the window could also be included in the state of a window where applicable. In practice, the state of viewers in Microcosm is stored in the registry in reference to the particular document and this information is used in retrieving the state. In this sense, the state and document are already linked.

5.3 Screen Managers

Windowing systems concern themselves solely with window management. In order to achieve this they only need the ability to manipulate the external state of the windows. This enables them to tile windows or maximise and minimise windows, but is not concerned with the contents of the windows.

The Microsoft Windows operating system is a good example of a window management system. Windows can be moved around and controlled by users independently but do not interact, other than to obscure each other providing a sense of front and back, a perceptual 3D cue on an otherwise 2D desktop. Facilities are provided which tile all the open windows, or list them on a task bar, but these features are embedded in the operating system and the information that these tools utilise is not exposed outside of the operating system.

5.4 Making the State Open to Examination

To enable external screen management processes to affect the state of windows, interface components will have to publish their current state. Furthermore, the responsibility falls on the interface component to publish any changes in their state when they occur.

This will provide open access to the state information of windows, either by viewing a list of window states or by direct querying of interface components to establish their current state. But, if solely relying on interfaces to publish the information, the state would only be flowing in one direction. To allow the external processes to affect the state of the windows, the interface components must also be in a position to receive new state information and modify their presentation accordingly.

The easiest way to achieve this is to provide a message handler within each of the interface components that can receive messages concerning state. If this message interface is a rigidly defined format then all interface components can be accessed in the same manner. The same message interface could be used to query the interface component for its current state.

The ability to send and receive state messages provides the open access to the state of the interface components that is needed.

5.5 The Intention Action model

We now have a two-way communication between the interface components and the screen handlers, but an issue of overall control still needs to be resolved. Ultimately we wish the screen handlers to be in overall control of the interface components. To achieve this, we must ensure that the interface components do not act on any state changes until the screen handlers have had the opportunity to approve or indeed modify the new state.

To take an example, a new document viewer is launched to display a document. It has an initial state that it publishes. A screen handler has saved a previous window state for this document, which it retrieves and sends to the interface component. The window is then moved to reflect the new state. The resultant appearance to the user is of the window starting up in one position and then jumping to a different position as a result of the screen handler's action.

In order to give the screen handlers total control, an intention action model can be adopted. Using the previous example, the new document viewer is launched to display the document. Rather than acting on its initial state it first informs the screen handlers that it intends to move to the position specified in its initial state. The screen handlers can then choose to modify the state or replace it with an entirely new state. This state is returned to the interface component and the window is modified to reflect the new state. Finally, the interface component publishes its new state.

The subtle distinction is that when the viewer appears on the screen it appears in the correct position and the user only observes the final state. The screen handlers are free to modify the state during the intention message loop and only once this has been returned does the interface component act on the state information. Examples of screen handlers described in later section will give more detailed descriptions of this intention action model.

5.6 The SHEP Architecture and Implementation

SHEP stands for Screen Handler Enabling Process. It was also the name of a sheepdog on the BBC program Blue Peter and the sheepdog analogy is a fitting

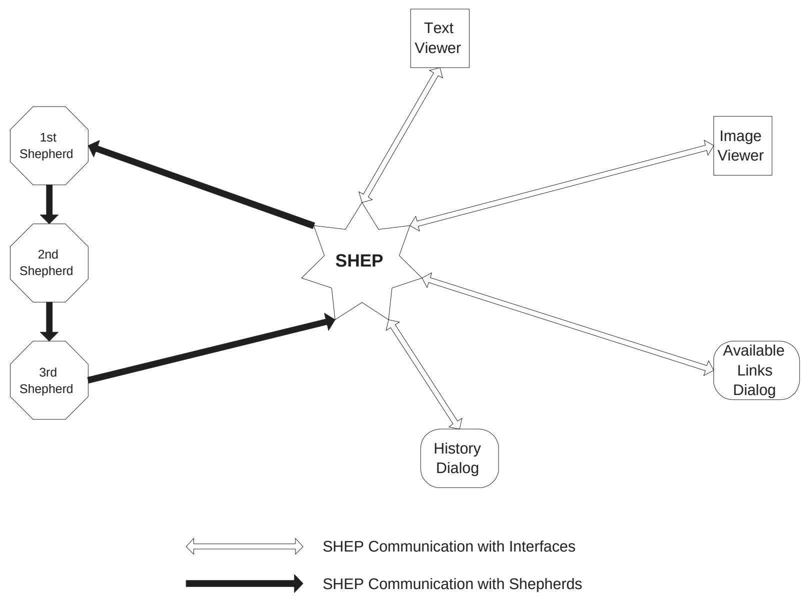


Figure 5.4: The SHEP architecture.

way of describing the underlying architecture.

The three key components in the SHEP architecture are:

- The interface components (sheep).
- The screen handling processes (shepherds).
- The communications process, or SHEP.

Figure 5.4 illustrates how the SHEP module connects the various interface components and the screen handlers.

The interface components in the SHEP architecture can be considered as sheep. They can be given explicit screen management instructions, for instance go to the top left hand corner of the screen, and they will carry those instructions out to the best of their ability. Whenever they are ordered to do something by a user, for example a user tries to drag the window across the screen, the interface component will communicate with SHEP to find out if it is allowed to move or whether SHEP has overriding instructions for it. The interface component cannot carry out any screen management actions without checking with SHEP, the virtual sheepdog. Each of the interface components has a direct two-way link to the central communications hub as is shown in Figure 5.4.

On the other side of SHEP we have the screen handlers, or shepherds. It is their job to organise the sheep. They can't talk to the sheep directly however, so they give their orders to SHEP, which passes them on. SHEP also passes any requests from the interface components to the screen handlers, which can then decide on any action that is required. As can be seen in Figure 5.4, the screen handlers are arranged in a chain primarily to avoid conflict. The chain also has side effects which can be exploited to provide specialist screen management functionality as will be discussed in later sections. The mechanics of the shepherd chain is perhaps best explained by an example.

The user clicks on a window title bar and tries to move the window to the right. When the user lets go of the mouse button the interface component tells SHEP that it intends to move to the new position. SHEP sends this to the first screen handler in the chain. This screen handler is only concerned that the window doesn't change in size. Since it doesn't, the screen handler decides to let the window move to the new position and lets the request go. The request then goes to the next screen handler. This screen handler is concerned that the window would be partially off the screen so modifies the request by resizing the window so that when moved to the new position the whole of the window is still on the screen. The modified message is then sent back via SHEP. Finally, the interface component carries out the modifications suggested by the screen handlers.

If however the shepherds had been in the opposite order, the first modification to the request would involve resizing the window in order to keep it on the screen. The other screen handler, which is concerned with resizing, would then receive the request and decide that it would rather not allow the interface to be resized. The result might be changing the final position of the window to retain the original size or alternatively just restoring the original size resulting in the interface component ending up off of the screen.

As can be seen from the simplistic example above, the use of a chain prevents conflicts in screen handling processes resulting in deadlock by allowing handlers later in the chain to take precedence over the screen handlers before them. The side effect of this is that different orderings of the same screen handlers can result in different perceived behaviour in the interface components. Furthermore, if it is not at the end of the chain a screen handler cannot be guaranteed of meeting its

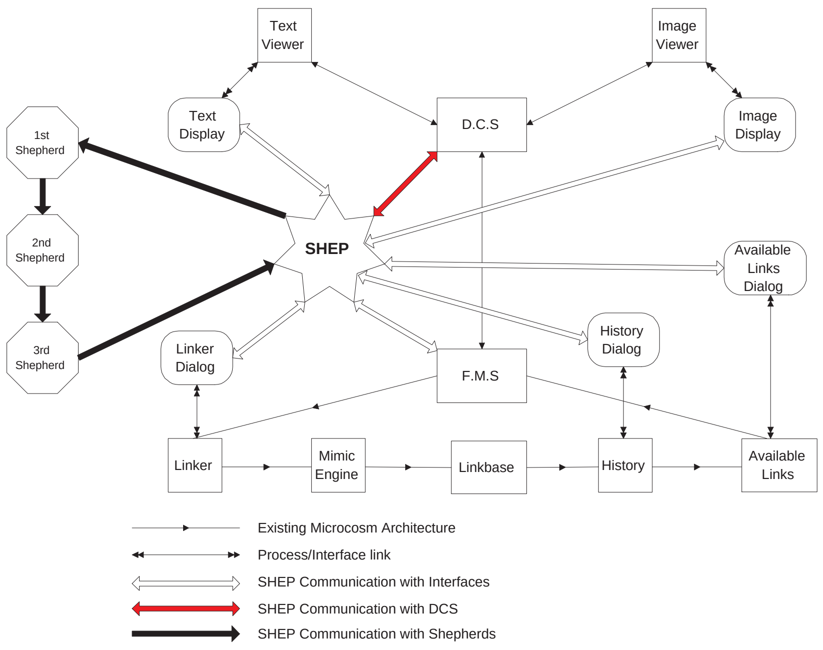


Figure 5.5: The Microcosm architecture including the SHEP framework.

own personal objectives of screen management. In practice the chain is usually constructed to ensure an overall screen management strategy.

5.6.1 The SHEP and Microcosm Architectures Combined

If the SHEP architecture displayed in Figure 5.4 is compared to that of the Microcosm architecture as presented earlier in Figure 3.1, the similarities are clear.

In both cases a central hub communicates with different processes. In the case of Microcosm it is the DCS / Filter Manager System combination, in the case of SHEP it is the central SHEP component. On one side of the hub sit a number of processes which communicate with the hub directly. These are the viewers in the case of Microcosm and the interface components in the case of SHEP. On the other side of the hub sit a number of processes which are arranged in a chain. For Microcosm this is the filter chain, for SHEP it is composed of screen handlers linked together.

Figure 5.5 shows how the SHEP architecture overlays on to the Microcosm architecture. Each of the Microcosm modules which has an interface component connects directly to SHEP. To make this clearer, the interface components have been shown as external components of the individual modules. The shepherds only connect to the central SHEP component and do not interact with any other part of the Microcosm system other than through their processing and replying to messages sent to them.

5.6.2 State and the SHEP Protocol

An integral part of the SHEP system is an extensible definition of state. This forms part of the SHEP Protocol and aims to provide a common language for sheep and shepherds. Most interface components will provide the core state information of size and position. Some interface components will also supply more specific information, for example an interface which generates audio information may choose to include volume as part of its state. Because the state information is not a fixed structure, the majority of shepherds may not understand or choose to deal with volume information, and ignore that part of the state. A specifically written audio shepherd however can take advantage of this information.

The protocol defines a number of standard messages that can be sent around the system. The messages are identified by their SHEPAction tag, which is examined by the shepherds in order for them to assess whether they can process the message. A few of the messages that are passed around the SHEP system are described below.

NEW.SHEEP This message is automatically sent by SHEP round the shepherd chain to indicate that a new sheep has registered.

STATE.CHANGING A sheep sends this message to indicate it intends to change its state to the state contained in the message. This is the 'Intention' message.

STATE.CHANGED A sheep sends this message to inform the shepherds of the state it has just changed to. This is the 'Action' message.

STATE.SAVE This message is sent by a sheep to indicate that it would like its state to be stored for future use. If there is a shepherd in the chain to process this message the state will be stored.

STATE.REQUEST A shepherd can send this message to a particular sheep to find out what its current state is. This can be useful for a shepherd that is carrying out global screen management as opposed to just local screen management on an individual window.

5.6.3 Interface Components: The Sheep

The sheep of the SHEP architecture include any component of the system that has an interface. For the purpose of SHEP, interfaces need not be restricted to simply windows on the screen, but might include audio or even haptic devices. This section illustrates how an interface component might interact with the SHEP architecture.

When the interface component starts up the first thing it does is register with SHEP. This sets up a line of communication for the component to send and receive state information. The sheep registers its name and an information message. This information message describes the component and is made available via SHEP to the shepherds. The message might indicate that the interface cannot be resized, as would be the case with some dialogs, or perhaps that the interface cannot be minimised. This information can be utilised by the shepherds when carrying out their screen management.

Once it has registered, the interface component asks SHEP for its initial state. SHEP can then ask around any registered shepherds and pass the state back to the component. The interface can then be rendered on the screen. If there are no shepherds registered, the interface component will use its default settings. Once registered, the interface component sends messages to SHEP whenever its state is about to change, and whenever its state has changed. These two messages are quite distinct and will be handled differently by the shepherds.

When a sheep notifies SHEP that it is about to change state, the state contained in the message is where the sheep intends to end up. The action STATE.CHANGING is included in the message to indicate this intention. This state can be modified though, and the sheep will move to wherever the returned state indicates. Alternatively, the returned message may inform the interface not to change states at all.

Once a sheep has changed its state, it sends a second message that informs SHEP of its new state. This is purely an information message and the sheep will not act on any state modifications made by shepherds to the message. To indicate this, the action STATE.CHANGED is contained within the message. This two-stage intention/action message notification is designed to reduce the possibility of deadlock or infinite loops.

Shepherds, via SHEP, can also send sheep messages. A sheep might receive a request for its state, which it can return as the reply to the message. This might be important for shepherds that are interested in global screen management as opposed to window management on a window by window basis. A sheep can also be told to move to a particular position on the screen. In order to give all of the shepherds in the shepherd chain a chance to approve the move, the sheep will respond by sending a STATE.CHANGING message. Once the state has been passed to all of the shepherds, the sheep will act on the modified state and send its STATE.CHANGED information message.

5.6.4 Screen Handlers: The Shepherds

A shepherd will be created with one or more specific screen management roles in mind. By having a chain of shepherds, in most cases it removes the need for an individual shepherd to be aware of the actions of the other shepherds in the chain. As I will explain below, there are situations where the ordering of shepherds becomes important and modifying the order of shepherds may modify the resulting screen management effect.

The first thing a shepherd must do is register with SHEP. This process allocates a unique identifier for the shepherd and initialises a two-way communication between the shepherd and SHEP. As part of the registering process, a shepherd assigns itself a priority that equates to where in the shepherd chain it would like to be placed. The priority is a number between -100 and 100. A priority of -100 will place the shepherd at the front of the shepherd chain and a priority of 100 will place the shepherd at the end of the shepherd chain. If a shepherd is not worried where it appears in the chain then the priority should be set to 0. It should be noted however that even if a shepherd registers with a priority of 100 it is not guaranteed to end up at the end of the chain since any shepherd registering with the same priority afterwards would usurp the original shepherd from its position.

It could be argued for notifying a shepherd that it is no longer at the end of the chain, however this might encourage the sly programmer to un-register and then re-register the shepherd to regain its forfeited position. This would seem to be a tailor-made scenario for deadlock as shepherds start to leap frog each other in the shepherd chain ad infinitum. Examples of shepherds for which chain position is important are given in later sections.

Once registered, the shepherd sits and waits for messages from SHEP. When SHEP receives a message from a sheep it passes the message on to the first shepherd in the chain. The shepherd may choose to ignore the message, modify the message and even generate new messages, however the message should be forwarded once the shepherd is finished with it. There are of course exceptions to this where a message should only be processed once. Again, these are dealt with later on.

When SHEP receives a message back from a shepherd it is passed on to the next shepherd in the chain. When the last shepherd has returned the message it is passed back to the sheep which originated the message. The sheep can then act on the information it has been provided by the shepherds.

5.7 Examples of Shepherds

The sections below describe a number of different shepherds that were created to operate with the SHEP system as implemented within the Microcosm architecture. Although representing relatively simple screen management functions they hopefully serve to illustrate the SHEP mechanisms and provide an insight into the possibilities provided by the SHEP architecture.

5.7.1 Saving State Information

Since documents contained in a hypermedia system are all different shapes and sizes it is extremely useful for the viewers to be able to remember the state they were in the last time the document was viewed. Some users won't mind viewing a text document in a small window where as others prefer to have the viewer as large as possible. These personal preferences must be stored if users are to avoid resizing the windows every time they are displayed. The same is true of navigational interfaces such as the history window, or the document selection dialogue.

State information in Microcosm is saved in the Microcosm registry. By using a hierarchical approach, similar to that of the Windows registry, the settings can be stored under a separate branch for each user, allowing personal profiles to be constructed.

Under the original implementation of Microcosm it was the responsibility of the individual interfaces to store their information in the registry. This required every module that wished to save information to understand the workings of the registry and to make decisions as to where to place the information.

Using the SHEP architecture, the interface component simply sends a message to SHEP containing its state and requests that it be saved. SHEP passes this message along the chain of shepherds, where a shepherd charged solely with interacting with the registry to load and save settings will carry out the request. When the interface component loads up a new document it sends a message to SHEP asking for the saved state of the document. This time the shepherd will retrieve the information from the registry and pass it back to the interface component via SHEP. This new method has a number of advantages over the original architecture.

- The interface components no longer need to be aware of the workings of the registry. Should the registry be modified in a way that affects its interface, only one shepherd need be changed rather than every interface component.
- By using the shepherds to load and save the information, improved screen management techniques can be added in a modular way without affecting the interface components, for example using relative rather than absolute size and position. This is described in more detail in the section below.
- There now exists a central control point as to where to write the settings in the registry. Before, the settings were always written to the users branches in the registry, which is appropriate when building personal profiles. Often however, the author of an application would wish to define default initial settings for the positions of documents. They may decide they want all text documents to appear on the left side of the screen with images appearing on the right. This would often involve the author saving all of the settings in their own personal profile and then using a registry editor to copy all of their personal settings into the application area of the registry. Once copied, they would become the default settings for users who have not built up a personal profile. By using a shepherd to control the writing to the registry, an author need only change a single setting to let the shepherd know that

the settings should be stored directly in the application branch. Without the use of SHEP, each interface component would have to be modified to handle this situation.

There is one disadvantage to the system however. If the save size and position shepherd is not in the shepherd chain, the size and position information is not stored. The interface components will only be aware of this when they try and retrieve the information only to be returned an empty state message. In order to accommodate this situation, when requesting for a state to be loaded, the interface component supplies a default state which it will use unless it is modified as a result of loading information from the registry.

5.7.2 Relative Size and Position

One particular problem encountered by authors of Microcosm applications was being able to achieve a prescribed layout for documents on the screen. For example, an author decided that all text files should occupy the left hand side of the screen and all images when they are viewed should occupy the right hand side of the screen. This could be achieved by setting the size and position of each text document to the left and image document to the right. Unfortunately, the Microcosm viewers save absolute screen size and position, since this is the information they operate with. Therefore, when a user views the application on a screen with a higher resolution, the text appears in the top left hand corner and the images somewhere in the middle at the top.

Using the SHEP protocol, a shepherd was written that converts from absolute screen co-ordinates to relative co-ordinates and vica versa. The relative co-ordinates are based on a virtual screen size of 32 000 by 32 000. This could have been implemented as a library, which all of the interface components called, but this would require modification of all of the interface components. If it were a library, the save/load shepherd could call it, or alternatively it could be incorporated into the save/load shepherd. In order to create greater flexibility, and to help test the chain mechanism, it was implemented as a separate shepherd.

Unlike most other shepherds, the relative size and position shepherd registers twice. It registers once at the start of the shepherd chain and once at the back of the shepherd chain. Because each time it registers it receives a unique identifier it is a simple process to keep track of the two separate registrations.

When the shepherd receives a message at the front of the chain, it reads all size and position information and generates relative size and position state information based on the current screen resolution. This new information is added to the message in different tags to ensure that the relative values are not inadvertently processed as absolute values. At the back of the chain it looks for relative size and position information and if it finds any it converts it to absolute size and position, again based on the current screen resolution.

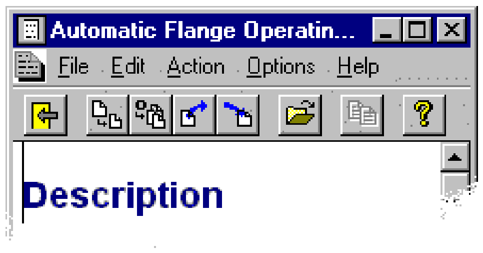
This has the overall effect that all shepherds between the two registered relative size and position shepherds can deal with relative rather than absolute size and position. In the case of the save load state shepherd, it is not interested in the contents of the state message but simply stores and retrieves it.

Taking the previous example, when the author stores the state of the windows it is stored as a relative size and position, which would equate to half the width of the relative screen. When the window is viewed by a reader and the state restored from the registry, the relative size and position is converted to an absolute size and position which would still appear as half the screen, irrespective of the current screen resolution.

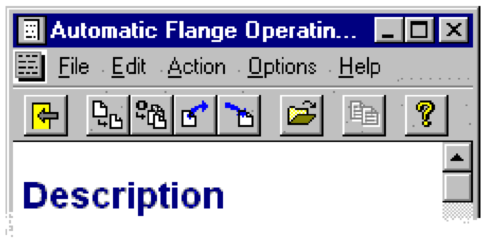
5.7.3 Restricting Window Positions

For naîve users, reducing the complexity of window management can be a real benefit. A shepherd was created which attempted to do this by restricting the way in which users can manipulate windows. It sought to address three common problems of window management.

- Users minimising windows and then being unable to find them again, a problem exacerbated by the Windows98 task manager's ability to hide itself.
- Users maximising windows and then not being able to see or interact with any other windows.
- Users moving windows partially off of the desktop obscuring their contents.



(a) The viewer displaying the locked icon.



(b) The viewer displaying the unlocked icon.

Figure 5.6: The Microcosm viewer locking mechanism.

Whenever a user attempts to modify the size and position of a window, the window sends a message to SHEP containing the state that it wishes to change to. The shepherd examines this new state, and if it finds that the window would be off of the screen, it modifies the state to keep the window on the screen, by altering the windows position. If the new state includes a minimised or maximised screen position, this can be blocked by the shepherd preventing the window from ending up in either of these states. The functionality can be customised for each user, so more experienced users are able to modify their settings to allow greater flexibility in their window management.

5.8 Controlling the Number of Windows Using SHEP

As well as controlling the position and appearance of windows once they appear on the desktop, a screen management system would want to have some control over the number of windows that are displayed at any one time. One possible way of achieving this would be to use the system described above to ensure that if a new window appears which exceeds the desired limit for windows on the screen an unused window is minimised. This does not provide a complete solution however as users could rapidly accumulate unwanted minimised windows, which will quickly use up the resources of the machine.

In Microcosm there exists a distinction between viewers, used to view documents, and navigational interfaces, for example the history list. As a way of limiting the number of viewers which a user can have open at any given time, a locking mechanism was devised. This appears as a small icon on the menu bar of the viewer shown in Figure 5.6. The icon is a representation of a page, with the corner folded over if the user wants to ‘lock’ the document and prevent it being replaced. If a document is not locked and the user opens a new document, the new document is opened in the existing viewer rather than spawning a new viewer. The user can toggle the locked state of a document by clicking on the icon. The default lock state can be specified in advance.

One problem with this system was that the locking mechanism only applied to documents of the same type. Because different viewers were required for media of different types, if the user opened an image, but had a text document unlocked, a new viewer would be spawned for the image and the text document remained. If a new text document had been opened, the previous text document would have been replaced. This therefore partially broke the metaphor of document locking, and meant that the user would be able to have at least one document of any media type on the screen implying that the user could not be restricted to one document at a time for instance.

In order to overcome this, the SHEP architecture was extended to encompass the document dispatching components of the Microcosm system. In the original Microcosm system, when the user requests a document to be launched, the dispatcher examines its table of opened documents. If it finds an open document of the same type, which is unlocked, it sends a message to the viewer to open the new document. The previous document is replaced.

The Microcosm dispatcher was modified to register with SHEP as a sheep. As with the viewers, the dispatcher publishes its state, which rather than having size and position information, contained information on the current viewers and documents that were open. Whenever it was about to launch a new document it would first send its dispatch message to the shepherds via SHEP. A shepherd could then modify the message to stop the document being launched, force the document to be opened in an existing viewer, or possibly close an existing viewer to make way for the new one. The dispatcher would receive the returned message and execute the instructions within it.

By providing a hook into the dispatching mechanism, other screen management activities could be carried out such as preventing the user from closing all the documents by trapping close document messages and discarding them if it is the last remaining document on the screen. This facility can be important where the sole navigation method around the dataset is intended to be link following.

5.9 SHEP in Use

The SHEP framework was implemented as part of an active research tool and has been utilised by a number of different projects. The sections below briefly describe two such projects and the way in which SHEP was applied to provide open screen management.

5.9.1 The Historian's Workbench

The historian's workbench consisted of a number of historical applications constructed using the Microcosm system. Applications could consist of a variety of images including paintings, cartoons, newspaper articles and maps, as well as textual material obtained from primary sources or specifically authored for the package. Where appropriate video and audio materials were also included. One of the aims of the project was to provide a more user-friendly interface than the basic Microcosm system provided.

A number of specific interface issues needed to be addressed due to the nature of the application. The first approach to these problems was to create a new interface using Visual Basic, which the application could be viewed with. This proved to be both inflexible and problematic and a more elegant solution was obtained using the SHEP framework. The following sections discuss a number of interface problems that arose with the historian's workbench, and the solutions that were achieved using SHEP.

Varying Levels of User Ability

The application was designed to be used by a wide variety of prospective people. These ranged from school children, who are becoming increasingly more computer literate, to academics that might not be familiar with the windows environment at all.

The context in which the application will be used also affects the interface. If the package is to be used in a University environment where students will be spending a number of hours using the package, it is more acceptable for them to have to learn how to use the interface. Indeed, they will also expect more functionality, such as the ability to compare images or have multiple documents open at the same time.

If the same package were to be used in a museum as an information kiosk providing additional information about an exhibit, the interface would probably need to be quite different[1]. Firstly, the users would spend a few minutes rather than many hours with the package so the interface would have to be very intuitive and straightforward to use. It is more than likely that the package would be used in combination with a touch screen display so the interface would have to avoid the need for any complicated clicking or dragging which would best be achieved using a mouse. Finally, because of the varying levels of abilities of the users the system would have to cater for the lowest common denominator and a simple page-turner style interface would be the obvious choice.

A number of shepherds were created which could be switched on or off depending on user preference. The shepherds each focussed on one aspect of screen management and in combination could be used to make the interface more straightforward to use. The functionality provided by the shepherds was :

- The ability to prevent the user from minimising or maximising windows. This can prove confusing for naîve users.
- Restricting the windows to the screen area by preventing the user from moving any part of the window off the screen.
- Restricting the user to only having one document on the screen at a time. If a new document is opened the previous document is closed.
- Ensuring that there is always a document on the screen. Initially, this operated by preventing the user from closing a document if it was the only document left on the screen. Eventually, an alternative, less confusing, approach was adopted whereby if the user closed the only remaining document a contents page document was opened for them allowing them to follow new

[1]The different context may well necessitate different styles of documents as a museum kiosk would be more likely to present bullet points and card file style text than textual essays, however an open hypermedia system such as Microcosm would not prohibit the use of a variety of document styles within the same application and the content of the package is not the focus of this chapter.

threads through the application. In a kiosk environment this also provides an easy way to reset the machine for the next user.

- Preventing documents from overlapping. When receiving information from a viewer that its state was about to change, the shepherd examines all the other interface components to see if they will be overlapped when the viewer moves to its new position. If they will, they are moved out of the way of the viewer, reducing the window in size if necessary. The shepherd was careful to ensure that by moving the component out of the way it wouldn't overlap other windows, causing a chain reaction that might well lead to an infinite loop.

Copyright Notice Requirements

Many of the images used as part of the content for the application were owned by publishers. To use the images, copyright clearance was required and the terms of the clearance varied from publisher to publisher. Frequently however, one of the terms was that the copyright notice had to be displayed whenever the image is visible on the screen. One approach would be to add the copyright notice to the bottom of all of the images. At first glance this appeared a straightforward if laborious solution, but problems quickly arose if the image was resized or if the user zoomed in to view details of the image. It was easy for the copyright portion of the image to be obscured or scrolled out of view, which technically broke the terms of the copyright agreement.

To overcome this, a separate text file was created which held the copyright information. This could then be viewed in a separate viewer to the image. A special shepherd was used to control the copyright windows and treat the image viewer and text viewer as single entities. The shepherd used the following rules to control the copyright features.

- Whenever the image viewer was moved the text viewer holding the copyright was also moved to keep it next to the image viewer. (The copyright window was always positioned beneath the image viewer.)
- The user was prevented from moving the image viewer into a position where the text viewer holding the copyright information was pushed off of the screen.

- If the image viewer was closed the text viewer was also closed.
- Likewise, if the text viewer was closed the image viewer was closed.
- The text viewer was kept to the front to avoid it being obscured by other windows while the image was still visible.
- If the image viewer was minimised the text viewer was also minimised, and likewise both were restored to normal size at the same time.

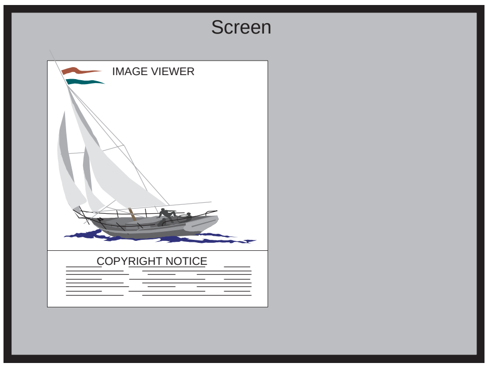
These rules obeyed the copyright terms to the letter, even though in practice, keeping the text window permanently to the front reduced the available screen resource for other windows. Figure 5.7 illustrates the affect these rules have on the interface to the application. The copyright window is always positioned just below the image viewer window as shown in Figure 5.7(a). Figure 5.7(b) shows the affect of preventing the user from moving the copyright window off of the screen, the bottom left hand corner being the furthest the user can move the window. Even when the image window is manipulated to zoom in or scroll around the window, the copyright text remains unaffected as shown in Figure 5.7(c). This would not be the case had the copyright information been added to the bottom of the image. Finally, Figure 5.7(d) illustrates the inability of the user to obscure the copyright information with other windows.

It would have been possible to achieve the same results by producing a modified version of the image viewer which had a separate pane attached to the bottom of it that could hold the copyright information. This would solve the problem, but would require producing a new version of the viewer. In addition, the copyright problem also applies to sound and video files, which are viewed (or heard) using different applications.

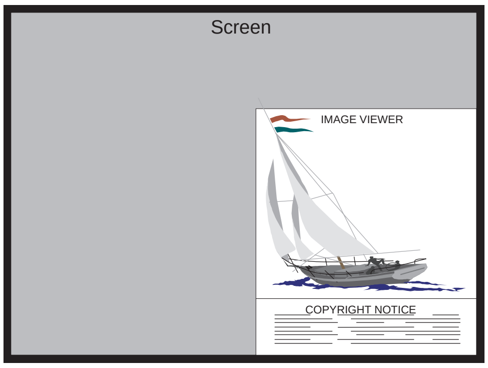
A final approach to the copyright problem would have been to display all the copyright information in a permanent task bar window along the bottom of the screen. Although this meets most of the requirements laid down in the copyright terms, there would be a dissociation between the copyright information and the piece of media that it refers to.

Synchronising Text and Image Representations of Documents

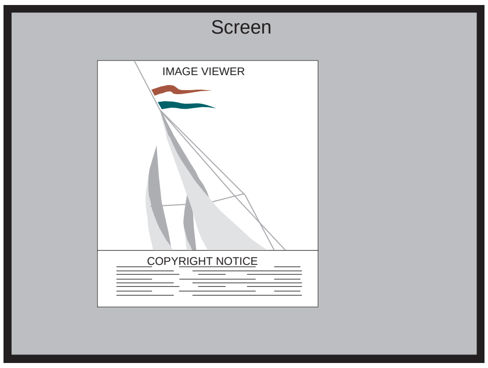
In order to provide primary sources for historians to work with, the applications will often contain images of documents, possibly hand-written. These images



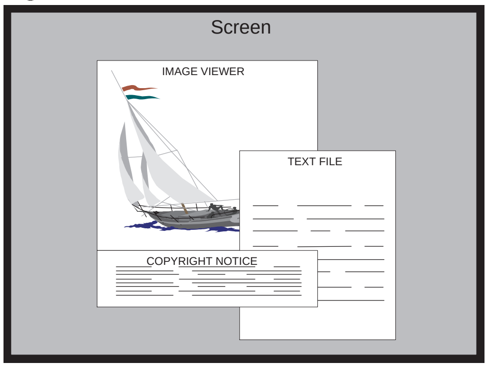
(a) The positioned copyright notice.



(b) The user cannot move the copyright notice off of the screen.



(c) The image can be scrolled and rescaled without affecting the copyright notice.



(d) The copyright must be visible even if only part of the image is visible.

Figure 5.7: The copyright screen management under SHEP.

are not always legible to everyone using the system and textual transcriptions are frequently provided. Figure 5.8 shows a typical screen where the image of the document is displayed on the left and the accompanying transcription is presented to the user in a text viewer on the right.

One of the goals of the historian's workbench was to maintain the semantic relationship between these two disparate viewers, presenting a unified interface to the user. Both the image and the transcription can be scrolled independently of each other, and the objective was to keep the two documents synchronised.

One approach would have been to create a new viewer that displays both documents. This however would be a hardwired approach involving large amounts of work to provide the full functionality of the current image and text viewers. Instead, using SHEP, a shepherd was written that monitored the internal state of the two viewers and kept them synchronised. A simple approach has the two viewers supplying state information on their scrollbars. If the image viewer is scrolled and informs the shepherd of the new scrollbar position i.e. 50% of the

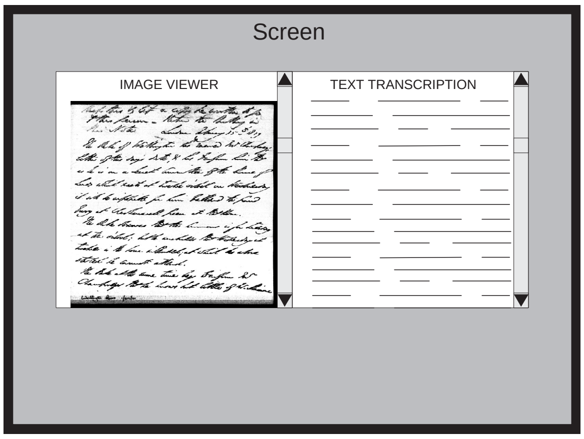


Figure 5.8: The synchronisation of document images and textual transcriptions.

way through the document, then the state of the text viewer can be altered to match this.

As well as providing synchronisation for images and text, the same technique, and possibly even shepherd, could be used to synchronise between two text documents. One might be a document in French, and the other the English translation. If the text viewer is able to give offset information in terms of paragraphs (which would presumably be preserved in the translation process) then the synchronisation can be more fine-grained than using the scrollbar position. Similar techniques could also be used to synchronise audio or video files to a scrolling commentary in a text file.

5.9.2 FIRM : Factory Information Resource Management

The FIRM project aimed to use open hypermedia principles to develop new techniques for integrated information management in the manufacturing environment, in particular to ease maintenance and fault-diagnosis problems. This involved the creation of a large factory-wide database of multimedia information, potentially available to other sites in the organisation through the use of appropriate communication technology.

As part of the project, a case study hypermedia application was constructed in Microcosm from the maintenance and operator set-up manuals for a large cable making machine at Pirelli cables in Aberdare. The application included a wide range of electronic documents including text manuals, engineering drawings and data tables. The documents were linked together using a number of open hypermedia approaches.

One of the reasons for using SHEP was that the system needed to be targeted at a number of different groups of users, such as :-

Shop-floor operators: These skilled operators use the system to aid them in setting up the machine, basic maintenance such as cleaning and lubricating the machine and loading the machine with the packaging components.

Maintenance personnel: These are highly skilled personnel who use the system for preventative and predictive maintenance, corrective maintenance (faultfinding) and resetting the machine once faults have been cleared.

Sales and Marketing: Outside of the factory environment the sales and marketing teams also need access to the information held within the multimedia information system. Their focus is different from that of the operators and they require quick access to the information at a less detailed level than the shop floor personnel.

The variety of users presents two key interface problems.

Firstly, marketing and maintenance personnel may wish to access the machine in an office environment where the use of keyboard and mouse is most likely. The shop-floor operators on the other hand will most likely be accessing the information as they are working on the machinery. The combination of a dirty working environment and the need for portability lends itself to the use of hand held pen based computers, or even voice operated hands free systems. These two different types of platform would necessitate two very different types of interface.

Secondly, the sales and marketing personnel are likely to be reasonably computer literate and have fewer problems working with a window based operating system. The shop floor operators are less likely to be familiar with such operating systems and rather than force them to use a complicated windowing system using a hand-held pen based computer which would probably require additional training, it seems more sensible to provide as straightforward an interface as possible.

With these differing requirements either a number of different interfaces could be created, or SHEP can be used to provide a variety of possible screen management

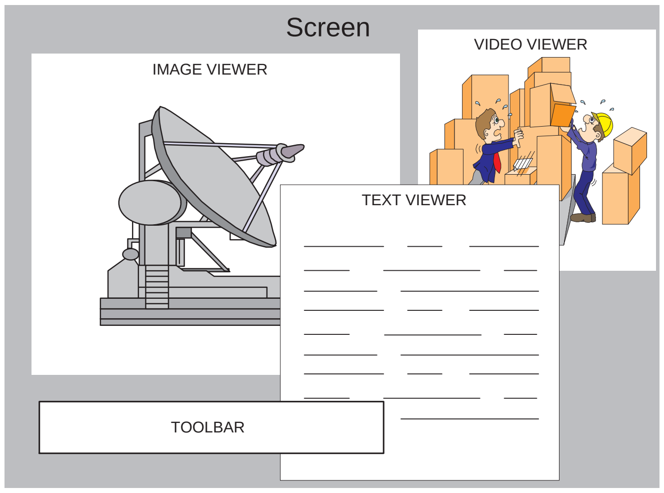


Figure 5.9: The original cluttered screen layout for FIRM.

scenarios which are appropriate to the users of the system.

Figure 5.9 illustrates a typical FIRM screen in Microcosm, with only the standard screen management being provided by the Windows system. This layout may well suit the technical authors who are modifying the system as they can shuffle documents around on the desktop and have a mouse and keyboard to interact with the information.

This interface would not suit the shop floor operator, who might be using a simple hand held machine with a pen interface or tracker ball. The windows would be too fiddly to move around the desktop and the smaller screen resolution compared to a large monitor in the office would reduce the screen resource available to the windowing system.

Instead, SHEP was used to create a fixed window layout as illustrated in Figure 5.10. Here, the toolbar always resides at the bottom of the screen keeping the functionality easily accessible. Documents are displayed in two windows, images on the left and text on the right. The windows cannot be moved or resized, reducing the screen management burden on the user. Although only two documents can be viewed at a time, this is more than adequate for the shop floor operator to carry out their maintenance tasks. By fixing the windows, the users tasks are reduced to clicking on the buttons on the tool bar or links in the documents. The interface is therefore simplified, removing more complicated mouse oriented tasks.

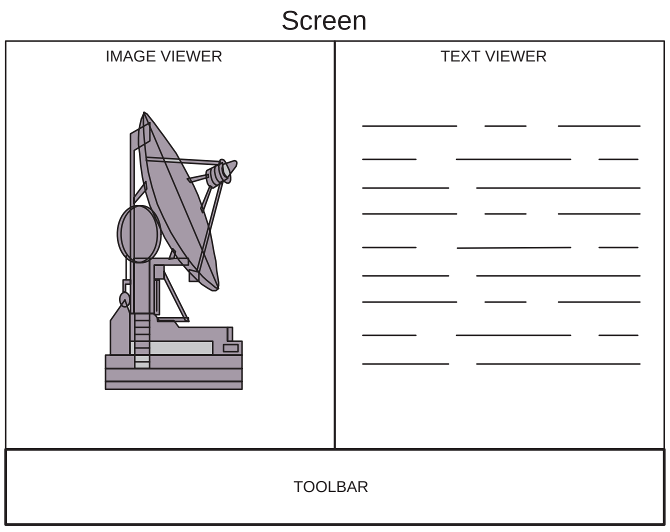


Figure 5.10: The SHEP organised screen layout for FIRM.

5.10 Problems with the SHEP Architecture

One of the problems with the SHEP architecture is the sheer volume of messages that can be generated by the system. The interface components will offer state information whenever the user makes the smallest change to the window. This could include scrollbar position changes as well as changes to the external state of the window. Ideally, if there are no shepherds registered which will utilise particular state information, the interface components should be informed that they don't need to send that particular piece of state information every time it changes. This would reduce the number of updates taking place within the system.

One way of achieving this would be to have the shepherds register which state tags they are interested in. SHEP could then collate these into a list that is available to all of the interface components. This would allow them to only send updates to the system when a state tag on the list has changed. Since state information can be stored from sessions to session, when the state is sent for storage it should include all state information even if there are currently no shepherds that make use of it.

Also, the presence of a central communications hub that every message must pass through presents a potential bottleneck in the system when message traffic is high. To reduce this, the SHEP library could act more as a broker than as a router. For instance, when a shepherd sends a message asking for the state of a sheep, the sheep is given a direct communication channel to reply to the shepherd through. This avoids the need for returned messages to pass through SHEP.

One last possible improvement would be in the arrangement of the shepherd chain. In the current system, each message is passed along the entire set of shepherds in the chain whether they are able to process them or not. If the shepherds were to register which messages they are able to process, SHEP could construct a separate chain for each message (possibly on the fly). This is equivalent to the Active Filter Manager topology described previously in Section 3.5.4. The message would then only be passed around those shepherds who can handle it. This would reduce the total number of messages passed through the system and speed up the response time to messages sent by the interface components.

5.11 Conclusions and Future Work

The problems of screen management are well documented. The advent of windowing systems both helped, in that by layering windows the illusion of more information being present than was actual currently visible could be achieved, but also hindered in that users had to develop new skills to manipulate these windows on the screen.

One of the problems open hypermedia systems have is that because of their modular nature they often have a number of interfaces present at the same time, displaying a variety of information. This presents a variety of problems for the user in that they may have to look in a number of places in order to find the information they are looking for, and sometimes there is no clear connection between two pieces of information which are related. For example, a copyright notice for an image may appear in a different window to the image itself.

To help overcome some of these issues, an open framework called SHEP was created which enabled the modular construction of tools to help control the screen and assist both the user and interface designer in their tasks. The framework was constructed as part of the Microcosm open hypermedia system and a number of modules in the system were adapted to take advantage of the framework. Finally, two example applications were looked at, and the benefits provided to the applications by the SHEP framework were considered. A number of problems can be identified with the framework and these were also examined.

The practical applicability of the SHEP architecture has been demonstrated by its integration into the commercial product Microcosm Pro by Multicosm limited. Once integrated with the commercial system it was used to construct an interface for a wearable computer produced by Xybernaut Inc, a US company based near Washington DC. The wearable computer uses the Microcosm system as its operating system, displaying the interface on a heads up display for the user to view. Interaction with the wearable machine is through voice communication via a microphone attached to the users headset. A shepherd was written that took the output of the voice analysis and converted the results into interface movements where applicable. The user can instruct the system to minimise or maximise windows and SHEP carries out the commands with the currently selected interface component.

The flexibility of the SHEP architecture enabled this functionality to be added through the implementation of a single shepherd that connected directly to the modular system. Existing shepherds were also configured to tailor the Microcosm interface to the 640x480 screen that was available in the head up display. This customisation only required the modification of a few entries in the Microcosm registry.

The use of the SHEP architecture within a commercial framework serves as a testament to its usefulness, as the flexibility of the system has already been demonstrated in the construction of the Xybernaut wearable computer interface to Microcosm Pro. In addition, it's use with FIRM project further demonstrates its practical application and user trials of the system constructed using SHEP have been published (Crowder *et al.*, 1997).

Providing increased control over the interface does help alleviate some of the problems of screen management, but sometimes the problem is more deeply rooted. Rather that seeking to place two interfaces together because the information they contain is related, a better approach might be to manage the information at a lower level. By combining the information rather than interfaces we can move towards true visualisations of the information. The remainder of this thesis will investigate this approach.

Chapter 6

Visualisation Systems

6.1 Introduction

One of the underpinning goals of hypermedia is to allow authors to construct cognitive pathways through information. Unless these pathways can be presented to the readers of the hypermedia, their function is lost. Traditionally in systems such as the World Wide Web, the pathways are shown by simply highlighting the beginning of a link, suggesting to the user that they may wish to look here for some relevant information. This is a basic form of visualisation, only a short step away from the inclusion of a full reference, but arguably it does not help the user to construct a mental map of the information.

This chapter investigates the roles of visualisations in hypermedia and examines both the mechanical process and the psychology involved in the creation of visualisations of open hypermedia. First, a brief overview of some of the technology used to create visualisations is given. This is followed by a brief look at some of the software tools with which to construct visualisations. Next, the concept of visualisation metaphors is discussed, with a number of different metaphors considered. Finally, an array of visualisation applications are reviewed, focussing on their relative strengths and weaknesses.

6.2 Visualisation Technology

Visualisations are all about the communication of information. This communication is bi-directional, with information passing from the machine to the user and requests and control information being passed from the user to the machine. Traditionally the machine to user mechanism has been the VDU screen, with information presented graphically. The user machine mechanism more often than not involves a keyboard and or a mouse. To enhance the computer human reaction hardware researchers have been looking beyond these simple communication tools and exploring the diversity of human senses and abilities.

The field of software visualisation has gone hand in hand with the development of hardware. From the Cinerama developed by Fred Waller in the 1930's, to Douglas Englebart's (Engelbart, 1963) modest suggestion that computers could be used to display information on screens and Ivan Sutherland (Sutherland, 1965) pioneering the use of head mounted displays for visualising information, software and hardware have driven forward the field of visualisation together. An excellent account can be found in Howard Rheingold's book on Virtual Reality (Rheingold, 1991). In it, Rheingold defines Virtual Reality as

> an augmentation tool and not an automation tool.

This is an important distinction. Virtual Reality is not an end in itself, and forms only part of the whole system. It is very easy to channel all the effort of a project into creating a wonderful 3D virtual world, and then spend what little time is left trying to work out how to use it.

Currently, desktop PC's have advanced to a stage where texture mapped 3D worlds can be manipulated using a variety of input technologies and Virtual Reality is available to everybody through rapidly improving 3D graphics hardware. Communication of information from machine to man does not end with hardware and graphics technologies though, and they form only a part of the overall picture.

6.3 Visualisation Tools

This section aims to give a quick overview of some of the software visualisation tools that are currently available. On their own, they provide frameworks which authors can use to create visualisations by adding information and by applying metaphors.

6.3.1 The Virtual Reality Modelling Language (VRML)

The Virtual Reality Modelling Language (VRML) was devised and implemented by Mark Pesce and Toni Parisi (Pesce *et al.*, 1994) in 1993, as a 3D interface for the World Wide Web. Pesce describes VRML as

> A language for describing multi-user interactive simulations - virtual worlds networked via the global Internet and hyperlinked within the World Wide Web.

The language is based on the Silicon Graphics Open Inventor format and at the time of writing has reached version 2.0.

The VRML file, or 'world', contains a complete description of a three-dimensional scene, listing all of the objects that appear within that world. It is the notion of a world that gives rise to the .wrl file extension used for VRML files.

Like the Open Inventor format on which it is based, VRML has a tag to signify the format being used and then a list of objects. Each object is self-contained, with nesting being possible to allow inheritance of properties from parent to child.

Objects within VRML are called Nodes. Nodes can contain anything from 3D geometry to JPEG images. Arranged hierarchically they form scene graphs. As well as geometrical shapes such as spheres and cubes, VRML also supports light sources and cameras. The properties of objects are called fields.

Two other important nodes in VRML are the WWWAnchor and the WWWInline.

WWWanchor : As the name suggests, creates an anchor on an object similar to anchors in HTML. By clicking on the anchor in the scene a WWW link can be followed to a document. The document might be a web page or indeed another VRML file. Along with the anchor, can be sent the x, y and z coordinates of where the user clicked within the anchor. This can be used to provide different results depending where on the object the user clicked. This behaviour is not unlike that of image maps in HTML.

WWWinline : This node is equivalent to the IMG tag for inline images in HTML. The inline component can range from a VRML object to an entire VRML file. An included file has to be in the VRML format.

It is these nodes that serve to raise VRML above the plethora of 3D modelling languages and into the domain of hypermedia systems.

6.3.2 Distributed Interactive Virtual Environment (DIVE)

DIVE (Carlsson & Hagsand, 1993), or Distributed Interactive Virtual Environment is a platform for developing virtual environments, user interfaces and applications. The system was designed with both distributed operation and collaborative multi-user applications in mind.

Based on a peer-to-peer network, DIVE makes use of the distributed nodes of the system to store the state of the objects in the system. By replicating the objects at a number of nodes, the consistency of the environment can be maintained if a node is lost from the network without warning.

Users are represented within the system with simple avatar shapes that can be enhanced by the mapping of images or even digital video onto their surfaces. Because DIVE is often used for collaborative tasks where communication between users is essential, the mapping of actual images of the users greatly increases the expressive capabilities of the system.

A number of research projects have utilised the DIVE platform and these include Q-PIT and VR-VIBE, which will both be discussed in detail later in this chapter.

6.3.3 RenderWare

RenderWare is included here as a representative for the increasingly large numbers of 3D graphics libraries that are on the market. Developed by Criterion software (Criterion Software, 1998) it provides a powerful API for the construction of virtual worlds. As with many of its counterparts, the library enables users to create virtual cameras, control lighting and most importantly to create 3D objects. The system provides, at a fairly low level, the tools for a programmer to construct a visualisation, without any emphasis on particular metaphors. As was mentioned earlier, this is just one of a whole host of similar libraries in a growing field, fired in no small part the proliferation of 3D games and increasing developments in PC graphics hardware.

6.4 Visualisation Metaphors

Metaphors serve to root the user within a familiar framework, which allows them to take advantage of real world knowledge in order to more successfully browse and navigate within an interface.

An example of a metaphor might be folders within a computer filing system. By using a concept which the users understand such as a filing cabinet full of folders containing documents, the users can apply their real world knowledge to the computer environment. When users put documents in the filing system they place them within folders. To view documents they must first extract it from the folder.

How far to take the metaphor is a question which interface designers are always faced with. Using the example above, the interface designer might choose to restrict users from having more than one folder open at a time, mimicking the fact that it is impractical to have multiple drawers of a filing cabinet open at a time. This would however impose unnecessary restrictions on the user, where there needn't be. Taking the example further, often no limit is placed on the user as to how much information can be stored in the folders. In the real world, when a folder or drawer begins to become overloaded the user might be forced to break the information down into subcategories and create new folders to hold these. In the computer world, users are less aware of the overloading of folders so perhaps are not encouraged to distribute their documents more evenly. Maybe the graphical folders should bulge more the fuller they get. In many cases these choices form a trade off between additional functionality and the comfort of a familiar environment.

> Metaphors, by definition, must provide imperfect mappings to their target domains. If a text-editor truly appeared and functioned as a typewriter in every detail, it would be a typewriter. The inevitable mismatches of the metaphor and its target are a source of new complexities for users.(Carroll *et al.*, 1988)

Members of the Multimedia Communication Research department of Bell laboratories have been exploring the use of metaphors for communication (Ensor, 1998). Their work includes a skywriting metaphor where a users message is written in a graphical sky by a virtual bi-plane. Over time the message disperses. Another example involves writing a message in the sand (Seligmann *et al.*, 1997). The message remains until a wave comes in and washes it away. Anyone walking along that particular stretch of the virtual beach will be able to read the message until it is washed away. A final example builds on the work of Hill (Hill & Hollan, 1992) and the representation of read wear on electronic documents. Called live web stationary, the representation of a web page is modified by users during the course of browsing. As more users view the document, so the document representation gathers smudges and coffee stains, showing the wear and tear of being handled by multiple users. Visual indications are given of which links have been followed most.

The next sections look at a number of different metaphors that are used in visualisation systems.

6.4.1 Desktop Metaphors

Perhaps the most commonly experienced metaphor is that of the desktop. First produced at Xerox's Palo Alto Research Centre on the Star desktop computer (Smith, 1982) it has quickly been adopted as an interface standard, and has seen much development principally by Apple and Microsoft.

The concept is simple. The screen area is representative of a desktop. Items can be placed on the desktop including files that represent pieces of paper. Perhaps most importantly, as on a real desktop files can overlap obscuring each other. Although a 2D metaphor this form of depth cueing provides a pseudo 3D environment. Using simple perspective techniques can increase this form of depth perception.

Kandogan and Schneiderman (Kandogan & Shneiderman, 1997) suggest that overlapping windows

> no longer provide efficient means to serve... for today's information-intensive applications.

They propose the use of *elastic windows* based on a space-filling Tiled layout. Their approach is to organise items on the desktop according to the users current tasks. The windows can be grouped hierarchically, providing the user with an overview of all of their tasks allowing them to move around the hierarchy as they switch task contexts. The standard window organisational tools are extended to allow users to apply operations to groups of windows. This re-enforces the model of the user working within a context space rather than simple and application space.

Another way to improve the usability of the desktop is to reduce screen clutter and streamline on screen menus. A novel approach suggested by Harrison and Vicente (Harrison & Vicente, 1996) is to use transparent menus. Although concentrating on providing legibility using transparent menus, the study looked at how the user switches focus within the interface from the background task to the options provided by the menu.

Other research on enhancing the desktop metaphor has focused on the design of desktop icons to provide the optimum transfer of information (Byrne, 1993) and the use of sound to enhance the interface (Brewster *et al.*, 1993).

6.4.2 The Book Metaphor

The book metaphor has been used a number of times, to ground the user in a familiar environment. The World Wide Web lends itself to a book metaphor, being constructed of individual pages often linked by the use of forward and back buttons representing the movement through the pages of the book. Contents pages are also prolific providing indexes into the pages. The use of bookmarks as page markers could not be a more direct analogy.

Card, Robertson and York took the book metaphor of the web one stage further when they developed the WebBook (Card *et al.*, 1996). The system takes a collection of web pages and produced a 3D representation of the pages as a book. The user flips through the pages of the book and sees an animation of the pages turning, allowing them to quickly scan for a particular page. Links to destinations within the book are highlighted in a different colour to those links whose destinations reside outside the book. On selecting a link within the book, the pages flip until the destination page is reached. This increases the cognitive knowledge of the user by visualising the transition between pages rather than the user just experiencing a hypertext jump to the new page. This added knowledge helps the user form an overview of the material, which would assist them in returning to material at a later date. For example, the user might remember that the page was near the front of the book. This of course relies on the page ordering within the books remaining consistent, which may not be the case if the books are created dynamically.

6.4.3 Library metaphors

The WebBook (Card *et al.*, 1996) sought to extend the web from single pages to aggregates of pages. Pages grouped together in a book can be manipulated as a single entity. By taking the metaphor one stage further a visualisation of a number of books can be created, adopting a library metaphor. Here, the WebBooks are arranged on shelves. The locations of the books could be arranged to represent the content of the books or perhaps they are just arranged on the shelves alphabetically. Users can browse the shelves to find the book they are interested in, or if when reading a book they follow a link to a different book, they can be taken to where the book is shelved. This allows them to return to the book directly at a later date if they wish.

This can be a very important form of cognitive map building. It is very easy to move around web pages by clicking on links, however it is often very difficult to remember the URL of a web page in order to return to it directly. All too often users find themselves having to retrace their route through pages via links in order to arrive at the desired page. By giving the pages a more memorable location such as a labelled shelf, users can use a more direct address navigation strategy rather than relying on the path following strategy which the web currently lends itself to.

6.4.4 Room based metaphors

The rooms metaphor (Card & Henderson Jr, 1987)(Henderson & Card, 1986) was developed at Xerox PARC and is an extension to the desktop metaphor. The user works on a number of workspaces, called rooms, and can move between rooms viewing the 'virtual' desktops contained there. It includes features such as the ability to share objects between rooms, allowing the user to manipulate the object from either of the shared rooms. It also allows for overviews of the workspaces and gives the user the ability to load and save their workspaces.

The underlying principle is the organisation of work by context, with the switching of rooms representing the changes of context. Users could have their report writing tasks in one room and their coding tasks in another for example. This has been widely adopted by UNIX window managers with many variations on a theme currently available.

The 3D rooms metaphor took the analogy one stage further by presenting the user with a virtual room. As well as the desk space, the user now had walls which information could be placed, creating a 2D display. The metaphor was enhanced with doors, which lead to the other rooms. Leaving by the back door would take you to the last room you visited. The user was provided with a floor plan of the rooms containing some indicators of what tasks were being carried out within the rooms. This built on the notion of using rooms for different contexts. A final addition was the use of ‘pockets’ in which users could carry pieces of information between rooms ensuring they were always available to them.

6.4.5 Spatial Metaphors

Spatial metaphors are those which seek to exploit the spatial positioning of objects in order to help the user gain a cognitive overview of the information. Some common spatial metaphors are discussed below and examples given of systems which use the metaphors.

Landscape Metaphors

The idea of a landscape metaphor is to place the objects representing the information onto a virtual landscape. Browsers of the information can then either walk around the landscape, as in Alphaworld or Habitat (discussed in later sections), or fly over the landscape gaining overviews of the information as is the case in QPIT. The key to the landscape metaphor is the generation of spatial information for the represented objects based on some form of semantics about the objects. This can result in clustering based on content, or simply laying out the information to retain hierarchical properties of the data.

The landscape metaphor often draws on work in town and city planning, the intent being to allow users to apply their ability to navigate around towns and cities to the navigation within the landscape visualisation.

A subcategory of the landscape metaphor is the Populated Information Terrain, or PIT. PIT's, as well as placing information on a 3D landscape, focus on the notion of multiple user interaction within the visualisation. Two such systems are AMAZE and Winona, both of which will be reviewed in later sections.

The Galaxy Metaphor

Using the galaxy metaphor, objects representing the information are placed in a 3D space, usually using position in the 3D space to represent semantic information about the object. Where objects are similar semantically they will cluster, appearing to the browser as a 'constellation' of objects hence the use of the term galaxy.

Unlike the landscape metaphor, the visualisation is fully three-dimensional. For this reason, the users more often than not fly around the visualisation. In order to view all of the information it is often necessary to allow the user to have six degrees of freedom on their movement, being able to rotate as well as move around the visualisation to gain the best view of the data. For this reason galaxy metaphors are often harder to interact with as users are asked to master more complicated interaction techniques which may seem unnatural to them.

Systems which have used the galaxy metaphor include VR-VIBE and Vineta.

Cyberspace

William Gibson coined the term Cyberspace in his seminal science fiction novel Neuromancer (Gibson, 1984). To use Gibson's own words

> Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation. By children being taught mathematical concepts... A graphic representation of data abstracted from the banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding.

The Cyberspace metaphor encompasses a number of different metaphors. It describes the spatial positioning of the galaxy metaphor with the cityscape of the landscape metaphors. Because of the wide range of it's usage I have chosen to mention it separately although there is clearly an overlap with other metaphors that have been discussed in this section.

6.4.6 Influences from Literature

It is perhaps worth mentioning the influence that literature, and in particular science fiction, has had on the field of visualisation and interface design. Sometimes an almost intuitive grasp of what is required, coupled with no necessity to adhere to current technology has lead to the description of interfaces which are now both inspirational and achievable. Take the idea of the World Wide Web as predicted by Arthur C. Clark in his seminal sci-fi novel 2001: A Space Odyssey (Clarke, 1968).

> There was plenty to occupy his time, even if he did nothing but sit and read. When he tired of official reports and memoranda and minutes he would plug his foolscap sized newspad into the ship's information circuit and scan the latest reports from Earth. One by one he would conjure up the world's major electronic papers; he knew the codes of the more important ones by heart, and had no need to consult the list on the back of his pad. Switching to the display unit's short-term memory, he would hold the front page while he quickly searched the headlines and noted the items that interested him. Each had it's own two digit reference; when he punched that, the postage-stamp-sized rectangle would expand until it neatly filled the screen, and he could read it in comfort. When he had finished he would flash back to the complete page and select a new subject for detailed examination.

Perhaps one of the best examples of life imitating art is that of Cyberspace. Through the system he describes in his novel Neuromancer (Gibson, 1984), William Gibson has inspired a large number of interface designers. The term Cyberspace has come to describe practically any 3D world presented to a user where information in the system is represented visually. In fact, the internet itself is often described as Cyberspace, alluding to the creation of a world where two people on the opposite side of the planet can meet and interact irrespective of the physical distance between them.

In his novel Snowcrash (Stephenson, 1993) Neal Stephenson describes a man-machine interface known as the Metaverse. Immersed in a virtual world, characters are represented by self created avatars and can interact with each other and objects within the Metaverse. The detailed descriptions of this place, emphasise the use of metaphors and also the physical representation of information as is illustrated by this short extract.

> The room is filled with a three-dimensional constellation of hypercards, hanging weightlessly in the air. It looks like a high speed photograph of a blizzard in progress. In some places, the hypercards are placed in precise geometric patterns, like atoms in a crystal. In other places, whole stacks of them are clumped together. Drifts of them have accumulated in the corners, as though Lagos tossed them away when he was finished. Hiro finds that his avatar can walk right through the hypercards without disturbing the arrangement. It is, in fact, the three-dimensional counterpart of a messy desktop, all the trash still remaining wherever Lagos left it. The cloud of hypercards extends to every corner of the 50-by-50 foot space, and from floor level all the way up to about eight feet, which is about as high as Lagos's avatar could reach.

6.4.7 Which Metaphor to Choose?

With all these different metaphors at our fingertips, which metaphor should we choose for a system? Perhaps the answer is to leave the choice to the user. Different tasks and different users will demand different metaphors to work within. This requires an open framework that allows us to visualise our hypermedia information in many different ways if we so choose. In his keynote speech at the Hypertext '87 conference Andreas Van Dam expressed this same opinion.(Van Dam, 1988)

> But we don't want to put things together in such a way that there is *one* point of view, because if we have learned one thing from interactive

> tools up to now it is that multiview is the way people work. You can not have it just one way. We need an update to Larry Tessler's "Don't mode me in." Jim Foley and I recently came up with "Don't metaphor me in." Don't give me a little card image and say that "That's all you've got, because that's what I thought you should want for your virtual shoe box." There have got to be multiple modalities and the designers have to be able to deal with that.
>
> Don't metaphor me in, don't give me only one way of looking at things.

6.5 Visualisation Applications

The sections below describe a number of visualisation applications for information. A brief synopsis of each application is given along with comments on the advantages and disadvantages of the visualisation approaches adopted.

Rather than using screenshots of the applications, illustrations are used which are designed to highlight the pertinent features of the visualisations, usually removing irrelevant information such as background desktop icons and operating system specifics. In each case, a reference is provided for the screenshot upon which the illustration was based.

6.5.1 SemNet

SemNet (Fairchild *et al.*, 1988) was developed to help visualise the complex relationships within large, arbitrary knowledge bases. It attempted to explore the use of 3D visualisation to improve user comprehension. The objects within the database and relationships within them mapped nicely to 3D graphs. The visualisation was essentially a technique for turning large knowledge bases into large directed graphs, exploiting users spatial awareness in assisting them to create cognitive maps of the information as shown in Figure 6.1.

Much of the experimentation with SemNet involved finding methods of reducing the complexity of the graphs generated by utilising techniques such as information hiding and novel graph layout approaches such as fish-eye views.

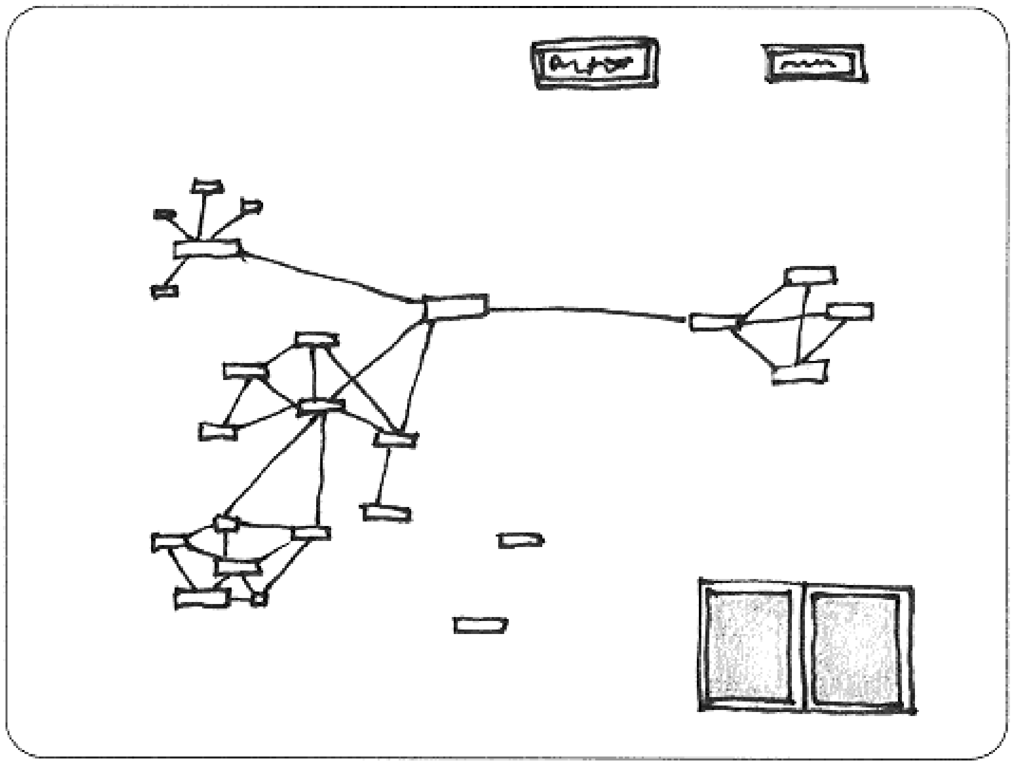


Figure 6.1: An illustration of a SemNet screen based on a screenshot in (Fairchild *et al.*, 1988).

6.5.2 GraphVisualizer3D

The GraphVisualizer3D was created by Ware et al. (Ware *et al.*, 1993) as a testbed for investigating approaches to presenting 3D graph information to users. The two main areas of investigation were information perception and graph layout strategies. The application area chosen was the visualisation of source code, where the program procedures and call trees map nicely into graphs constructed from nodes and arcs

Drawing heavily on the SemNet system, the GraphVisualizer3D allowed visualisations to be created of graphs constructed of nodes and arcs, where both the nodes and arcs could contain multiple attributes. Where SemNet put the emphasis on automatic graph layout, GraphVisualiser3D advocated a more manual layout process, arguing that automatic layouts can do no more than provide a first approximation, with the best layouts taking into account semantic information which can only be provided by authors.

The system took a Benediktine approach to visualisation, with nodes represented

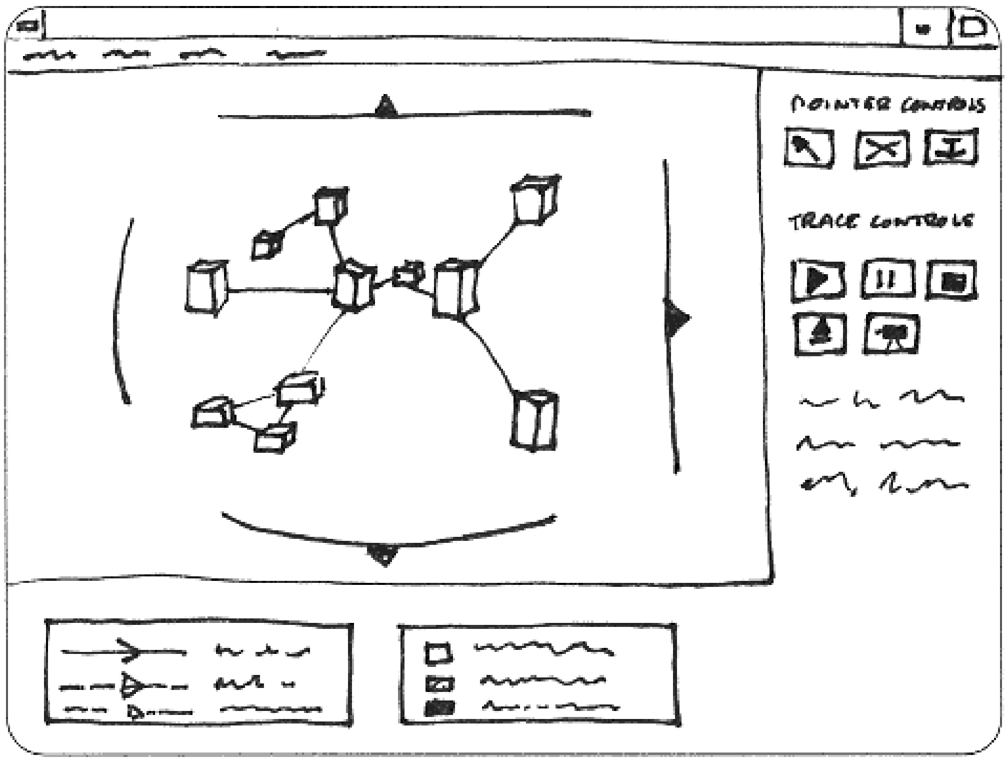


Figure 6.2: An illustration of the GraphVisualizer3D interface based on a screenshot in (Ware *et al.*, 1993).

as objects and arcs being represented by the lines connecting them. Attributes of the software objects being represented in the visualisation such as type, size and structure were mapped onto attributes of the virtual objects such as shape size and colour. Figure 6.2 illustrates a typical screen from the GraphVisualizer3D interface.

Unlike a number of the other systems mentioned here, user trials were carried out in order to assess what factors most affect the users ability to understand a graph from a 3D visualisation. Different methods of interaction were studied using the system, including both hardware and software approaches to improving the interface. The different approaches were assessed based on the users ability to determine whether two highlighted nodes in a complex graph were connected by no more than two arcs. The trial considered the time taken to make a judgement and the error rate of the judgement. The different conditions under which the trial was carried out included; 2D representation, 3D representation, stereo information, rotation and full head coupled display techniques. All of the conditions took place either on just a monitor, or by using fish_tank VR including stereoscopic glasses and hand and head tracking techniques.

The results concluded that the 3D solutions provided less errors and slightly faster assessments than conventional methods. Other techniques such as stereoscopic displays and head coupling did provide some improvement over the simpler techniques but will always be both more cumbersome and more expensive to provide.

6.5.3 The Information Visualizer

The Information Visualizer (Card *et al.*, 1991) was developed at Xerox's Palo Alto Research Center. It is based on a Rooms metaphor and includes a number of novel interface tools designed to support a new interface paradigm.

An array of tools were developed to provide animated visualisations of hierarchical information resources. Some of these are described below.

Cone-Trees

Cone trees (Robertson *et al.*, 1991) provide a novel interface for representing large data collections. Cone Trees are hierarchies presented uniformly in a three dimensional space. An example can be seen in Figure 6.3.

The top of the hierarchy forms the apex of the cone, with child nodes appearing below. As we move down the hierarchy, the number of nodes in the current level is increased, therefore the nodes are more spread out leading to the appearance of a cone. Each child node forms the apex of a cone comprising its children. The base diameter of the cones is reduced as you go down the tree to ensure there is sufficient space to display all the nodes at that level. By presenting the hierarchy in 3D the available screen space is used more effectively.

Another variation on the theme is Cam trees. In most respects they are identical to Cone trees except the tree expands horizontally rather than vertically. In the experimentation carried out at Xerox Park on Cone and Cam trees, it was found that smooth animation when moving around the visualisation was important in helping the viewer to maintain their cognitive model of the information.

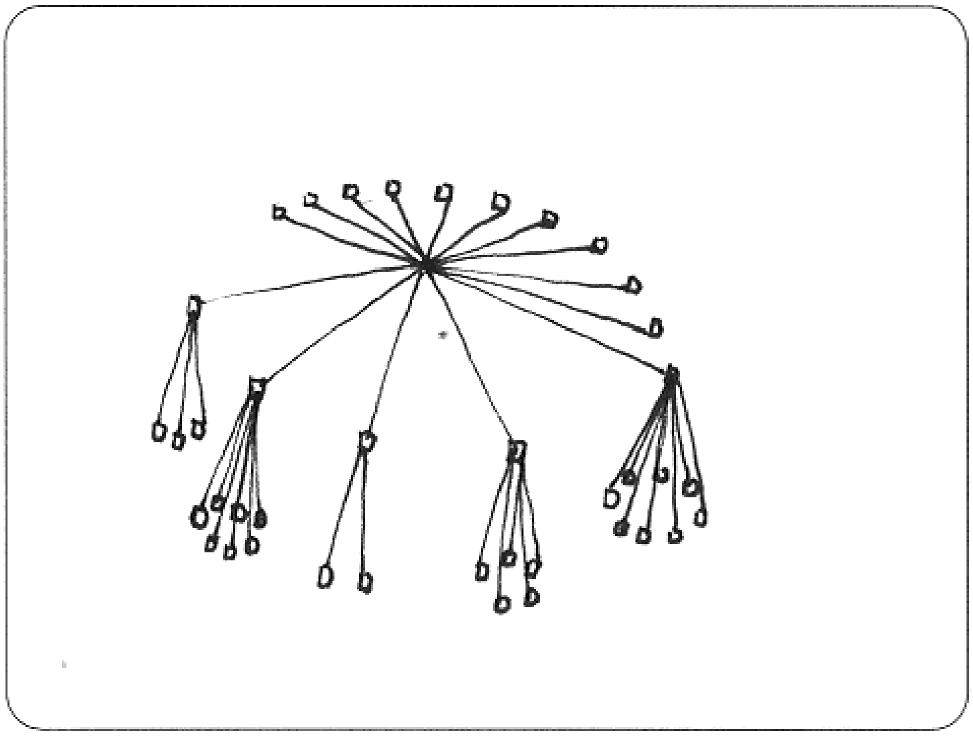


Figure 6.3: A typical cone tree visualisation based on a screenshot in (Robertson *et al.*, 1991).

The Perspective Wall

A major problem encountered when exploring large information spaces is the difficulty of examining detailed information while still maintaining a contextual overview. This is a particular problem with linear information, such as a spreadsheet, which when rendered has a much wider aspect ratio than the desktop. This would traditionally involve the user only being able to view a section of the information with the rest being effectively 'off the screen'. The Perspective Wall (Mackinlay *et al.*, 1991) was designed to address this issue.

Wide 2D layouts of information are mapped onto a 3D wall that then is divided into three panels. The central panel contains the detailed information that the user is focussing on. The two panels on either side are rendered in perspective, as if falling away at an angle from the screen. By rendering the section of the wall in perspective, the reader gets an overview of the information without being able to make out the detail. This is shown in Figure 6.4.

The reader can move left and right along the wall, bringing different parts of the information onto the central detailed part of the wall. The overall effect is similar

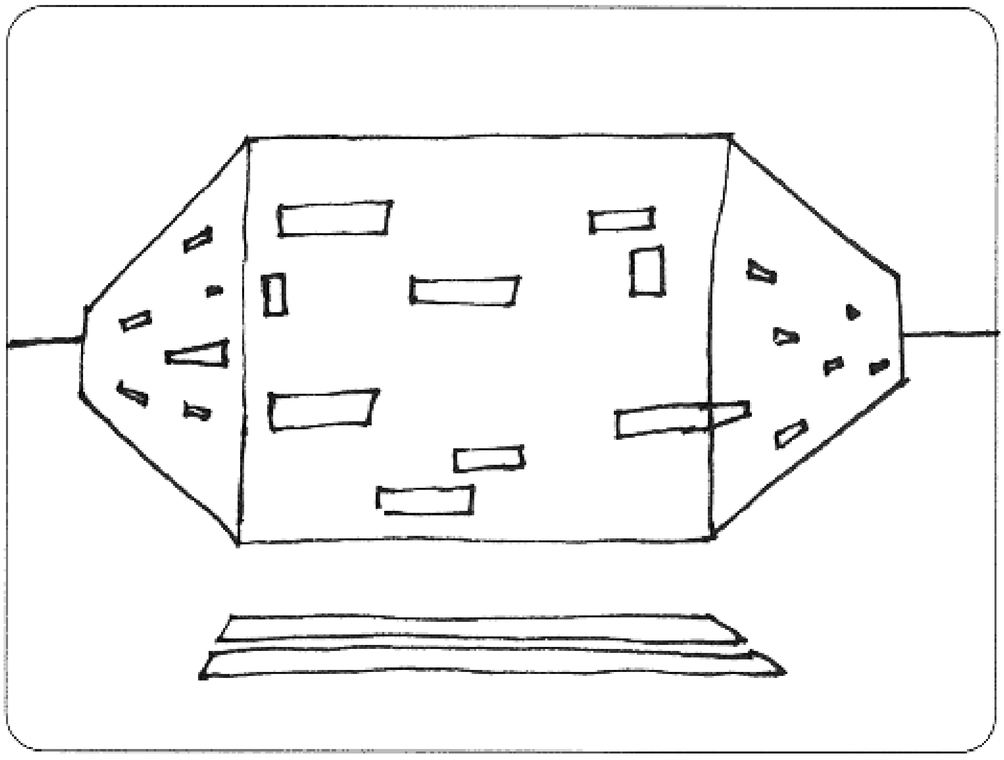


Figure 6.4: An illustration of a perspective wall based on a screenshot in (Robertson *et al.*, 1991).

to that of a fisheye view, with the reader's focus drawing information toward them, and peripheral information receding into the background.

The Table Lens

The table lens (Rao & Card, 1994) draws on the work of the perspective wall, allowing an overview of a table of information to be presented concurrently with a detail view of part of the information. Where it differs is that it operates in two dimensions rather than the single dimension of the perspective wall.

In order to overcome the restrictions of limited screen space, a variation on the fish-eye technique is used to increase the proportional amount of space occupied by the cells of the table within the focus of the view. The size and shape of the focus area can be changed, determining what part of the information the user sees in detail. Information out side of the focus area is compressed in the remaining space allowing an overview of the information to be viewed. The effect of this on a spreadsheet is illustrated in Figure 6.5.

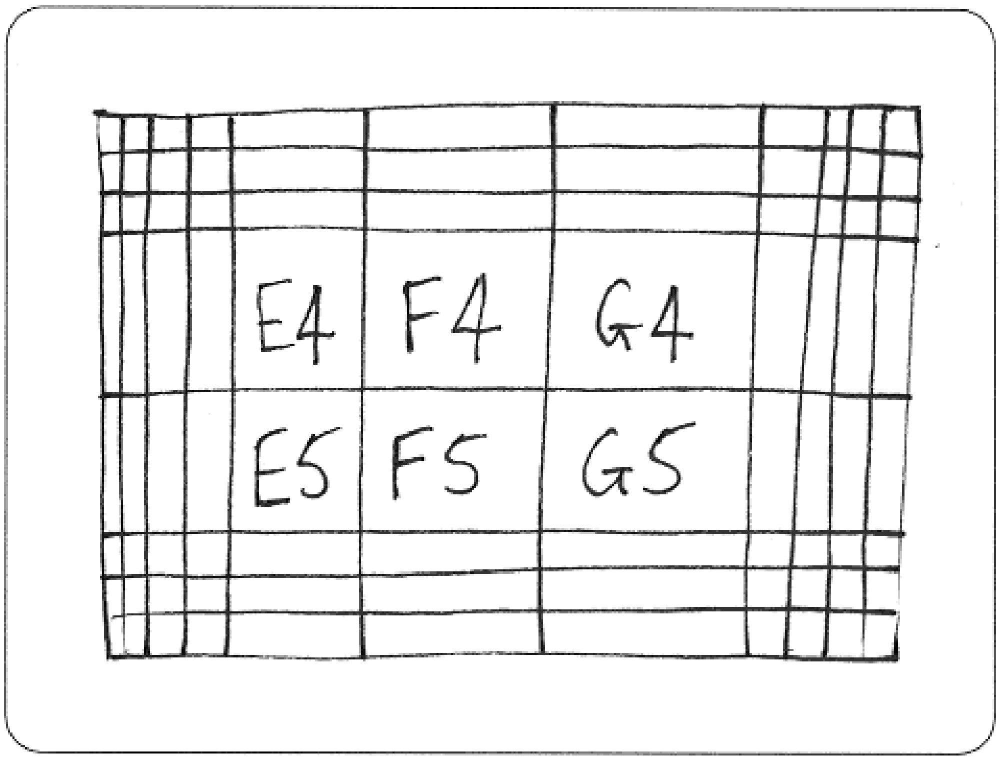


Figure 6.5: An example of the context+focus table view produced by the table lens, based on a screenshot in (Rao & Card, 1994).

Because the visualisation is designed for use with tables of information, maintaining the regularity of rows and columns was important and any distortions applied to the tables are only ever stretches in the horizontal or vertical directions, ensuring that the rows and columns always remain lined up. This has the added benefit that multiple focus areas can be allowed, providing the user with the ability to compare detailed information from two different parts of the table.

6.5.4 VR-VIBE

VR-VIBE is a 3D version of the original 2D VIBE system created at the University of Pittsburgh (Olsen *et al.*, 1993). VIBE provided the user with a two dimensional visualisation of a collection of documents.

VR-VIBE uses statistical techniques to visualise a document bibliography and allows the user to interact with and manipulate the space (Benford, 1995). VR-VIBE has been implemented using the DIVE system (Carlsson & Hagsand, 1993) described earlier in section 6.3.2. The VR-VIBE environment is constructed from a bibliography file and a corresponding set of queries.

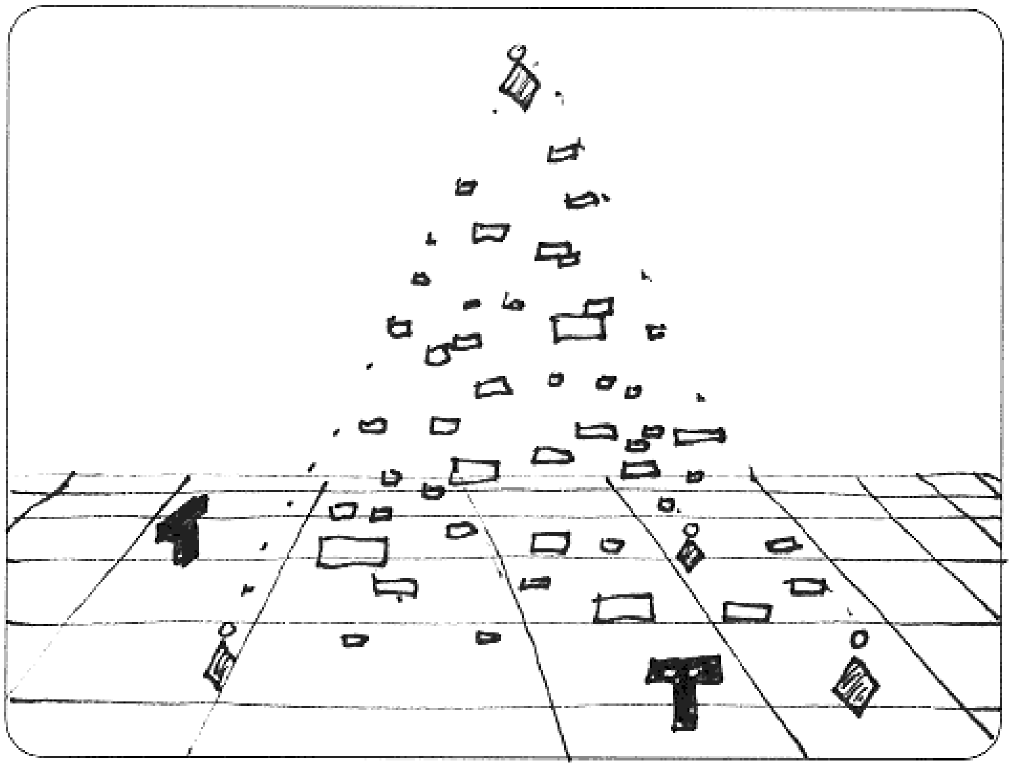


Figure 6.6: An example of a VR-VIBE visualisation based on a screenshot in (Benford, 1995).

The queries are sets of keywords that can be searched for within the content of the documents in order to generate scores that show the relative correlation between the query and the document. The queries are placed within the environment forming Points of Interest (POI's). The proximity of a document to the POI is directly related to the relative score of the document with the query. The relative attraction of the document to the queries is used to derive a position for the document within the virtual space. The VR-VIBE interface is shown in Figure 6.6.

If a document has an equally strong match to two queries then it is placed equidistant from them. One problem that arises however, is that if a document is equally weakly associated to the two queries it would also be placed equidistant from them. Because of this, 3D spatial location is not enough in itself to show the strength of the association to the queries.

In order to display the overall relevance, two approaches were taken.

If the queries are all placed on a two-dimensional plane, the third dimension can be used to indicate the overall relevance of the document. The higher the document is positioned above the plane, the more relevant it is.

If the queries are arranged in a three dimensional space, the spatial location of the document gives no clue as to its overall relevance. In this case, a combination of size and shade was used. The larger and brighter the document icon, the more relevant the document was overall. One drawback of this method is that large document icons can easily obscure the smaller less relevant documents. This makes the ability for a user to rotate the visualisation and view from different positions even more important.

Users can move around the visualisation with six degrees of freedom. When the user selects a document, by clicking on it, its title and author are displayed. Users can also mark documents for later reference, which causes the colour of the document to change. This provides an 'at a glance' history mechanism allowing the user to easily see which documents they have examined already. Users can also launch browsers to view the documents from within the system by double clicking on the document icons.

Relevance filtering is also employed to restrict the visualisation to those documents that satisfy certain threshold criteria. The queries can be dragged around the environment, and dynamically created and destroyed. Changing a POI causes the document space to be redrawn with the documents occupying their new position relative to the current POI's.

The system supports multiple users within the same document space, with users able to see representations of each other within the system. Co-operation between users is essential, since when a user modifies the document space by adding, removing or changing a query, the document space is re-arranged for all the users. Because the document representations are not fixed, when the spatial positioning changes it can be easy for users to lose track of documents they are interested in.

6.5.5 Lyberworld

Lyberworld (Hemmje, 1993) was developed to provide visualisations of queries carried out on an information system called Fulltext. The information space was

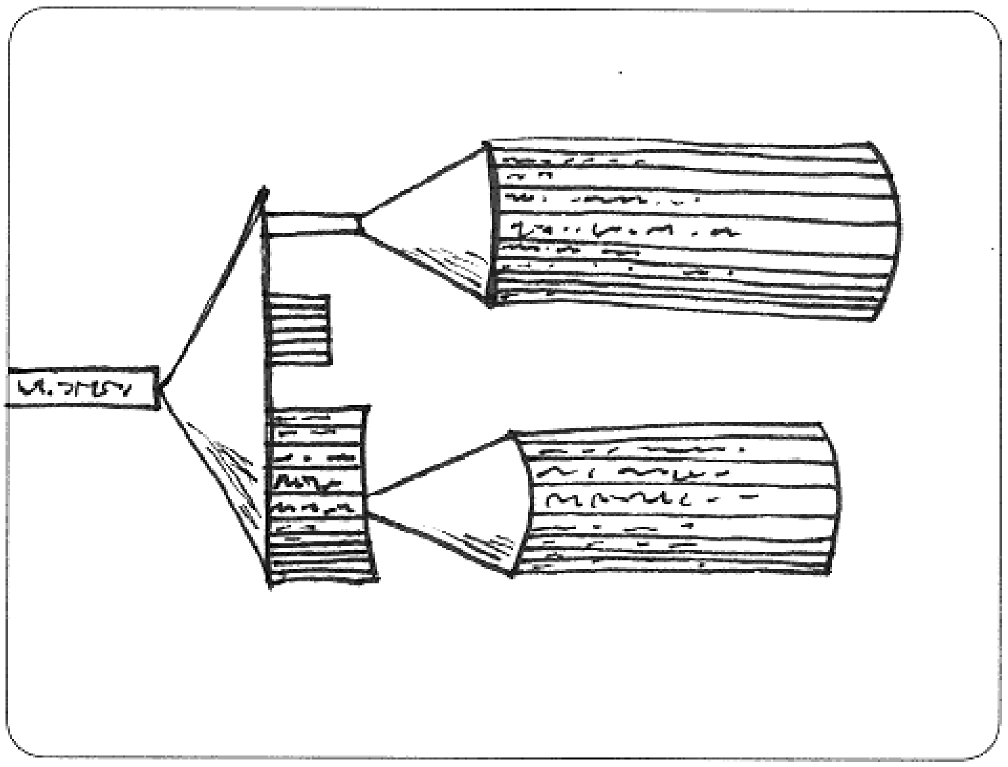


Figure 6.7: An illustration of Lyberworld navigation cones based on a screenshot in (Hemmje, 1993).

modelled as a network of documents and terms. The space was visualised using NavigationCones and RelevanceSpheres.

Navigation Cones

The NavigationCones were constructed to illustrate the extent to which the information space has been explored, presenting query paths through the information. They were loosely based on the Cone trees approach of Robertson (Robertson *et al.*, 1991) with the cones running horizontally rather than vertically. This allowed the nodes to be stretched which gave the opportunity to label the sides of the nodes with text labels to identify the objects. An example of navigation cones is shown in Figure 6.7.

One problem with cone trees is that the visualisation is most suited to data that is inherently hierarchical. To overcome this, Lyberworld maps network structures to hierarchical tree structures by introducing redundancy.

Relevance Spheres

The Relevance sphere was used to suggest documents that were similar to those matched by the users query. It operates on a similar principle to VR-VIBE but attempts to address the issue of documents with equally little relevance to the POI's being located in the same position as those with an equally strong relevance to the POI's. The terms of the query are placed on the outside of the sphere. The relevance of the document to the overall query equates to its proximity to the outside. A totally irrelevant document will be at the centre of the sphere. The document's relevance to the query terms is represented by its relative proximity to the terms. The user is free to rotate the sphere to look at it from any angle. This is designed to remove the overhead from the user of guessing the perspective of the 3D world from effectively a 2D rendition.

6.5.6 Q-PIT

Built using the DIVE system, the Q-PIT prototype provides visualisations of a database (Benford, 1995). The user generates a database schema that contains mappings from fields in the database to properties of objects within the visualisation. For example, people's names might be mapped onto the x-dimension, occupation on the y-dimension, location on the z-axis, age onto the spin speed of the object and gender onto the shape of the displayed objects.

Users can move around the visualisation examining the objects and their relation to each other. Q-PIT is a multi user system, and other users can be seen in the visualisation, represented by the standard DIVE 'T' shaped icon with eyes drawn onto the crosspiece of the T . This simple icon gives an indication of the users focus of attention, which can then be perceived by other users of the system. Figure 6.8 shows a QPIT screen in which a user can be seen amongst various data objects.

Once the users are familiar with the mapping system, they can begin to connect semantic associations with patterns they perceive within the visualisation. A particular cluster of documents might indicate a large group of people of the same occupation in the same location. The predominance of a particular colour might give a quick visual indicator of the gender make-up of the group of people represented in the database.

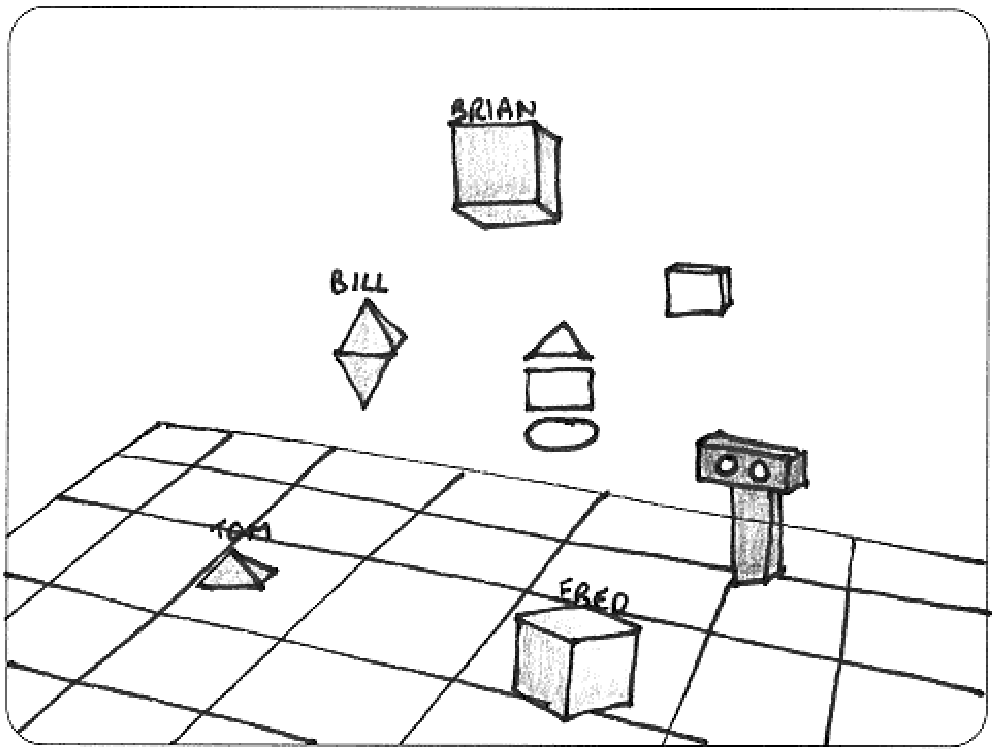


Figure 6.8: The QPIT visualisation interface based on a screenshot in (Benford, 1995).

A drawback of the Q-PIT system is that it relies on the users to provide all the mappings between object fields in the database and virtual object properties. Also, as in VR-VIBE, if one user modifies the mapping this must be communicated to all the other users in order to maintain the integrity of their cognitive associations between virtual world properties and real world information.

6.5.7 BEAD

BEAD is another multiuser virtual environment constructed using the DIVE toolkit (Chalmers & Chitson, 1992). It uses a technique known as simulated annealing to arrange the documents into a three-dimensional information terrain. By using a similarity metric based on word co-occurrence, documents are located on the terrain near similar documents. Initial versions of the system attempted to enforce the similarity metrics as closely as possible, resulting in a galaxy representation with documents clustering in three dimensions. Further research used a more 2D approach with the documents being located on a plane with only minor vertical shifting being used. The resultant visualisations resemble more of an information landscape. Figure 6.9 shows a birds-eye view of a BEAD landscape.

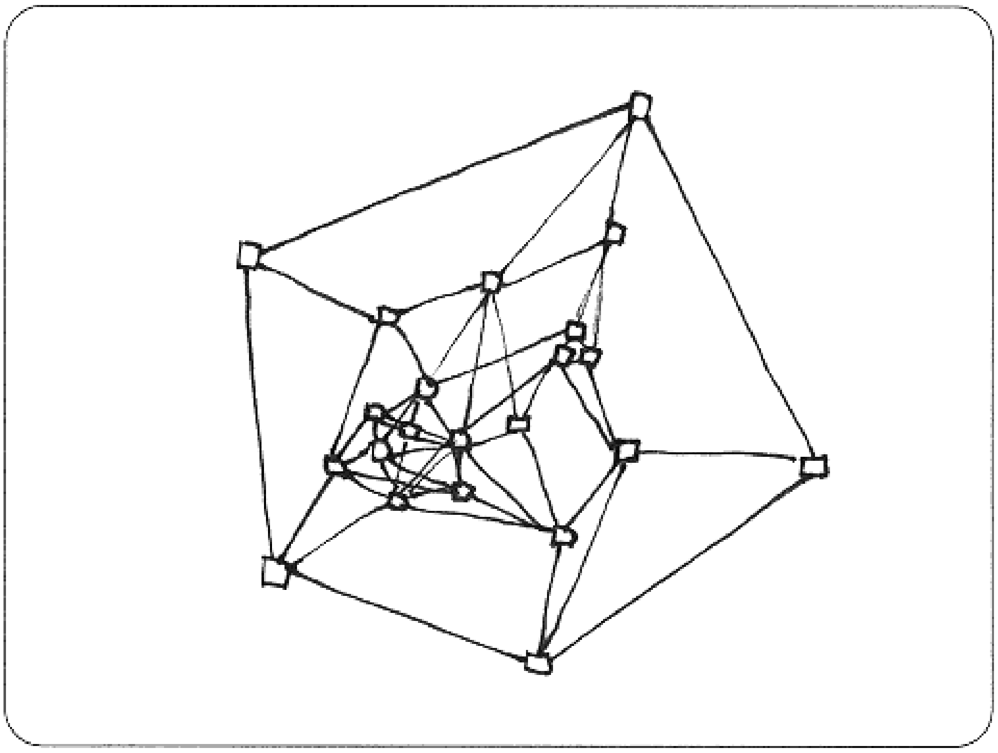


Figure 6.9: An illustration of a BEAD landscape (birds-eye view) based on a screeshot in (Chalmers & Chitson, 1992).

Multiple users can roam the landscape, represented using the standard DIVE avatars. Users can interact with the information and perform searches which leads to the objects matching the query being highlighted. This approach avoids some of the problems encountered when querying alters the spatial layout of the documents. The similarity metric also enables users to infer that documents close to the highlighted document might well be similar in content.

Being a multiuser visualisation, users can observe the actions of other users and be guided by their interactions with the information. Users are able to use overview information such as the density of documents to begin to form a cognitive map of the information. Similarly, documents located on the periphery of the visualisation are likely to have no real relation to the core contents of the information space.

6.5.8 Vineta

Vineta (Krohn, 1996) is a visualisation system designed for visualising, browsing and querying large databases of bibliographic information. Like VR-VIBE and Lyberworld, documents and terms are presented as graphical objects within a

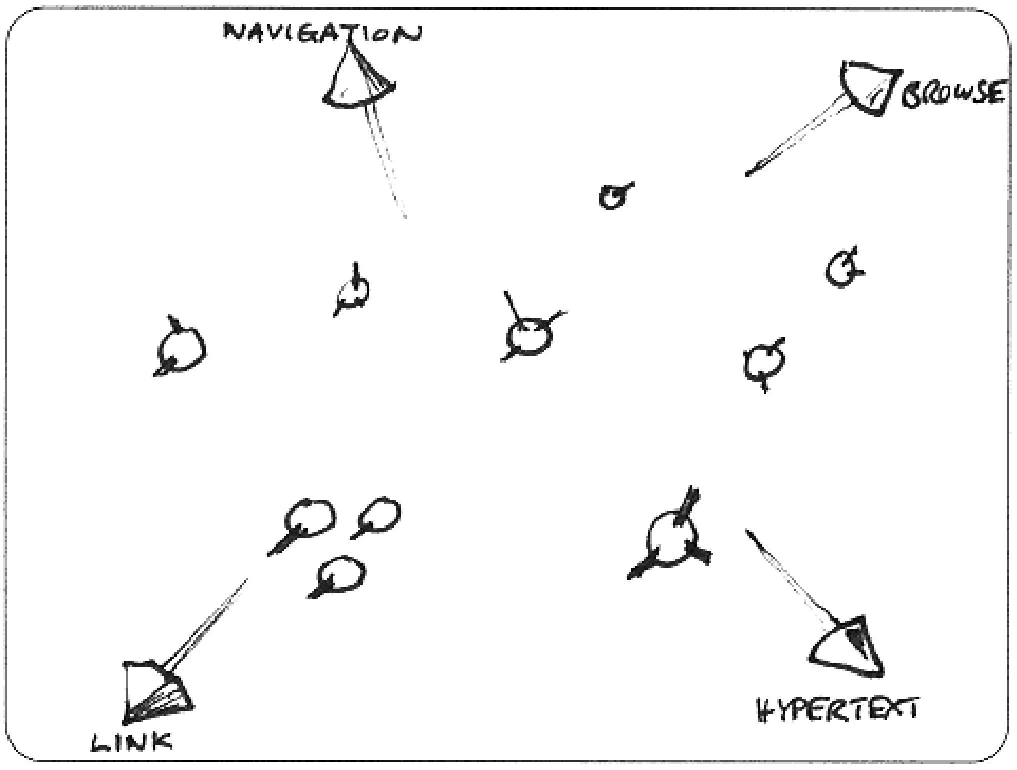


Figure 6.10: The Vineta galaxy visualisation based on a screenshot in (Krohn, 1996).

three-dimensional space. The position of objects within the space is representative of the semantic relevance between the documents, terms of the query and the user's interests. By using multivariate analysis and numerical linear algebra, Vineta suffers less from restrictions in the number of terms that can be mapped simultaneously compared to other systems. As in other systems, spatial proximity provides a direct correlation to semantic similarity.

Vineta uses two metaphors in generating visualisations. The first is a galaxy metaphor, where documents appear as clusters of stars, with terms presented as 'shooting starts'. The closer the stars are in the navigation space, the more similar the documents they represent are. Projections from the surface of the objects point at the various queries in the space. The length of the projections gives an indication of the relevance of that particular query. Figure 6.10 represents a galaxy visualisation presented by Vineta.

The second metaphor that can be used in the system is the landscape metaphor. Here, a flat landscape is presented to the user with documents represented as flowers. Flowers nearer to the user's viewpoint are most relevant to the current query. The direction and colour of the petals on the flowers is used to represent

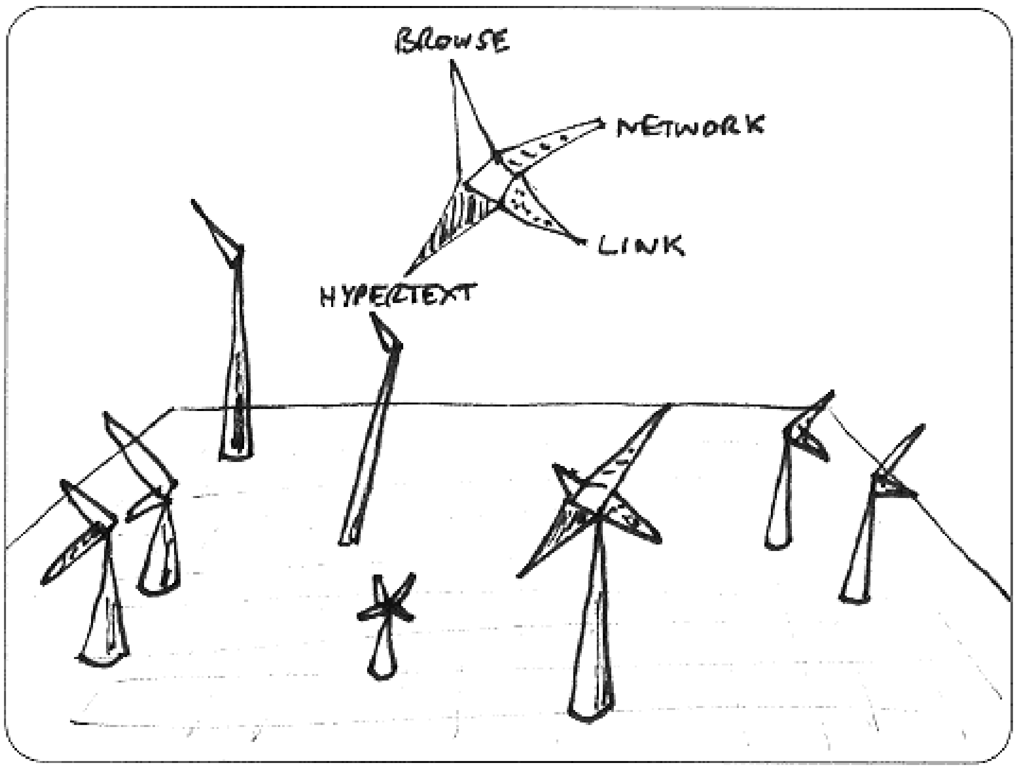


Figure 6.11: The Vineta landscape visualisation based on a screenshot in (Krohn, 1996).

the search terms and their relevance to each document. A flower covered Vineta landscape is illustrated in Figure 6.11

6.5.9 Habitat

Lucasfilm's Habitat was a large scale, multi user, graphical, virtual environment (Morningstar & Farmer, 1991). As an early form of shared cyberspace, the commercial project could support a population of thousands within its virtual world. It presented a real time animated view into an on-line simulated world in which the users could communicate with each other, play games and interact with the world in a wide variety of ways.

The principle lesson which its creators saw as coming out of the project is that cyberspace is defined more by the interactions which take place within it than by the technology with which it was implemented. In fact, its implementation platform was the Commodore 64 home computer, so the rendering of the world was cartoon like with the participants represented by simple avatars of the form of animated figures. Communication took the form of chat style conversations that

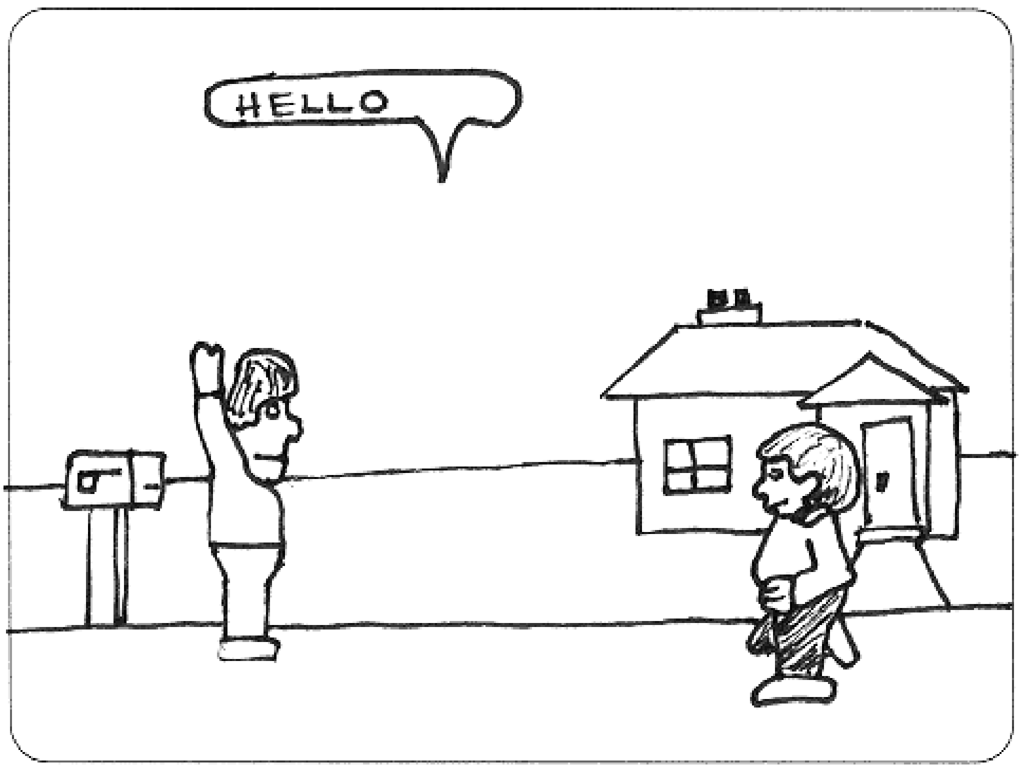


Figure 6.12: The cartoon look and feel of the Habitat world based on a screenshot in (Morningstar & Farmer, 1991).

appeared in speech bubbles above the character's heads. Figure 6.12 is a sketch of a Habitat screen.

This front-end, although primitive by today's standards, was quite capable of supporting the rich and varied activities that were happening within the visualisation. Despite the limited graphics, the experience was reported as being impressively immersive, and the system shows that rich interaction and useful content is usually more important than flashy graphics.

6.5.10 The Information City

The Information City (Dieberger, 1996) was designed as an extension to the Room's metaphor (Henderson & Card, 1986). Authors construct their hypertext from a number of basic building blocks, in this case quite literally.

A hypertext is represented by a single building in a landscape of hypertexts. Within the metaphor, a building with an open door is representative of a document with a strong relation to the users current context, and a half-closed door would represent a weaker relationship. The exterior appearance of a building is used to provide information about the document such as its age and complexity. By the user creating an interest profile, the city can be laid out to reflect the interests of the users, with documents on a particular topic being located within a district of interest. To overcome the problem of movement over large distances within the city, Deiberger opted to include a subway metaphor rather than a teleportation metaphor as this was thought to prove less disorienting to the user.

One of the core ideas of the Information City is that the information gains a location in the visualisation that doesn't change. If a user wishes to revisit the location they are able to use navigational strategies such as path following or direction cues in order to find the information again.

The explicit spatial model draws heavily on the work of Lynch(Lynch, 1960) in the field of city planning. By sticking rigidly to the metaphor, users should find the interface intuitive to use as it draws on their natural spatial awareness. Although providing the comfort of familiarity however, the rigidity of the metaphor imposes a number of harsh constraints on the system, and as in the real world the 'architects' are restricted in the buildings they can create. Also, the spatial distance between the information can become annoying to the user. One of the great advances of the information age was the ability to take information distributed geographically and allow it to be accessed easily from a single location. The Information City, may unwittingly provide a virtual reconstruction of this previous problem.

6.5.11 Alphaworld

Alphaworld (Worlds Inc., 1993) was designed and built as a virtual community where users can become citizens and move around and interact in a 3D virtual world. The world is loosely based on the Metaverse described by Neal Stephenson in his novel Snowcrash (Stephenson, 1993). Citizens can acquire plots of land and build their own buildings within the 3D environment. Figure 6.13 shows an art gallery constructed within Alphaworld. Users can view paintings on the walls as they walk around the gallery. By clicking on the pictures, large, more detailed versions can be viewed in a web browser.

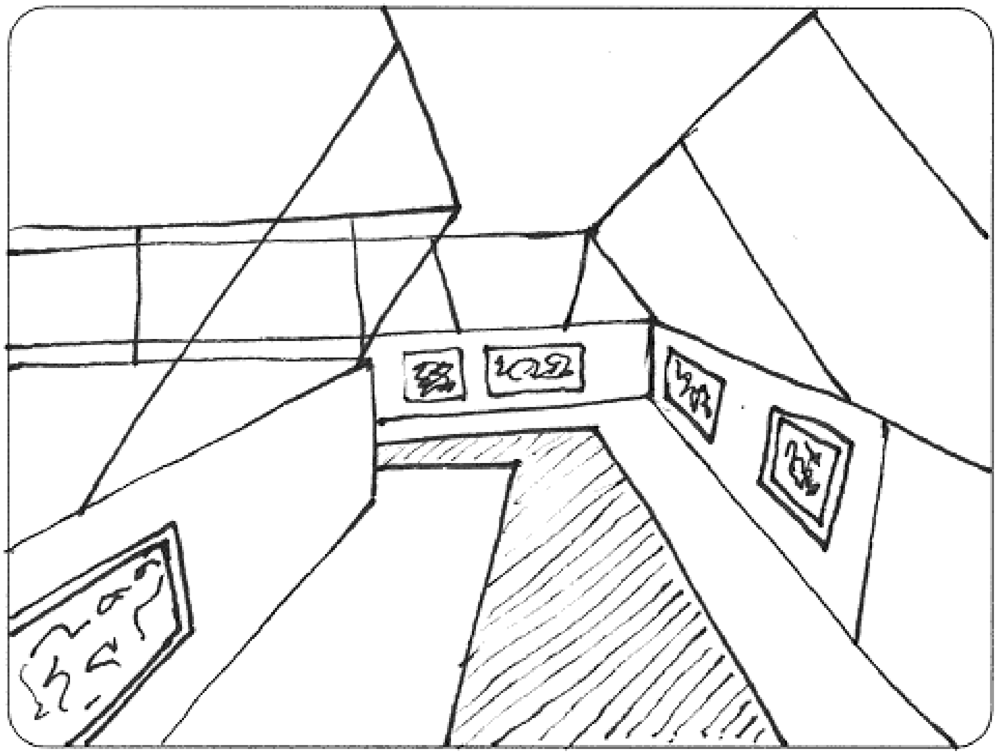


Figure 6.13: An art gallery constructed in Alphaworld based on a screenshot in (Worlds Inc., 1993).

The users are represented within Alphaworld by avatars, which they can create in the form of their own choosing. When users communicate through the chat interface, the text appears in front of the avatar, which is speaking in a pseudo-cartoon like fashion, not dissimilar to Habitat. Users are encouraged to create their own avatars and in so doing become visually unique beings within Alphaworld.

The system operates with a simple client server infrastructure with the server holding all of the models and textures required to construct the world. Initially the client will download these models and textures as and when it needs them, with local caching speeding up this process.

Navigation around Alphaworld follows simple town planning principles. The world is divided into zones with roads and footpaths connecting them as shown in Figure 6.14. In order to overcome problems of distance, features such as teleporters have been created which provide rapid movement between locations. This does provide a discontinuous jump that breaks the real world metaphor, but was considered a necessary evil.

Although it is possible to access web pages by interacting with objects within

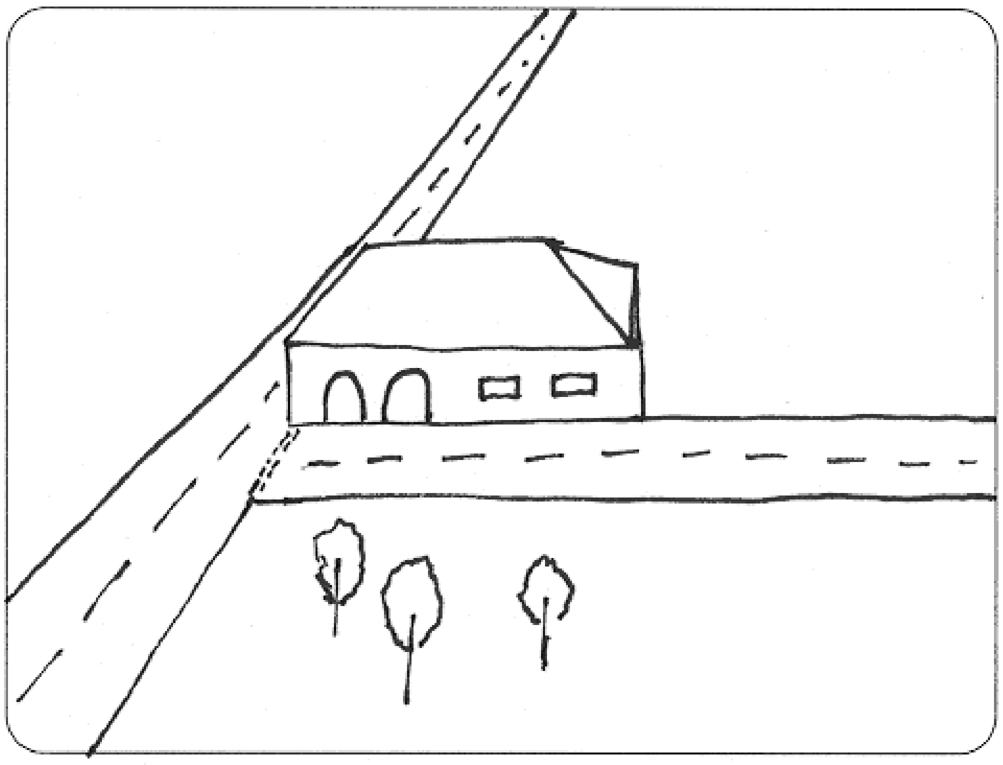


Figure 6.14: The Alphaworld 3D landscape from a flying perspective based on a screenshot in (Worlds Inc., 1993).

Alphaworld, it was the social aspects that provided the main thrust of the development. There have even been extensive documents on etiquette written which guide users in what is and isn't acceptable behaviour in the virtual world.

6.5.12 VIRGILIO

The main goal of the VIRGILIO project (Massari *et al.*, 1997) was stated as to

> Define novel, intuitively usable visual user interfaces, which significantly reduce the cognitive load of the user when working with multimedia database whose schema contains many semantic relationships.

Unique among the systems discussed here, VIRGILIO attempts to use dynamic metaphor generation to create a visualisation of a database that is both intuitive for the user to navigate and appropriate to the information contained in it. By combining user profile information and examining the structure of the database, the system generates VRML scenes which enable the user to interact with the results of their queries in a fully 3D environment.

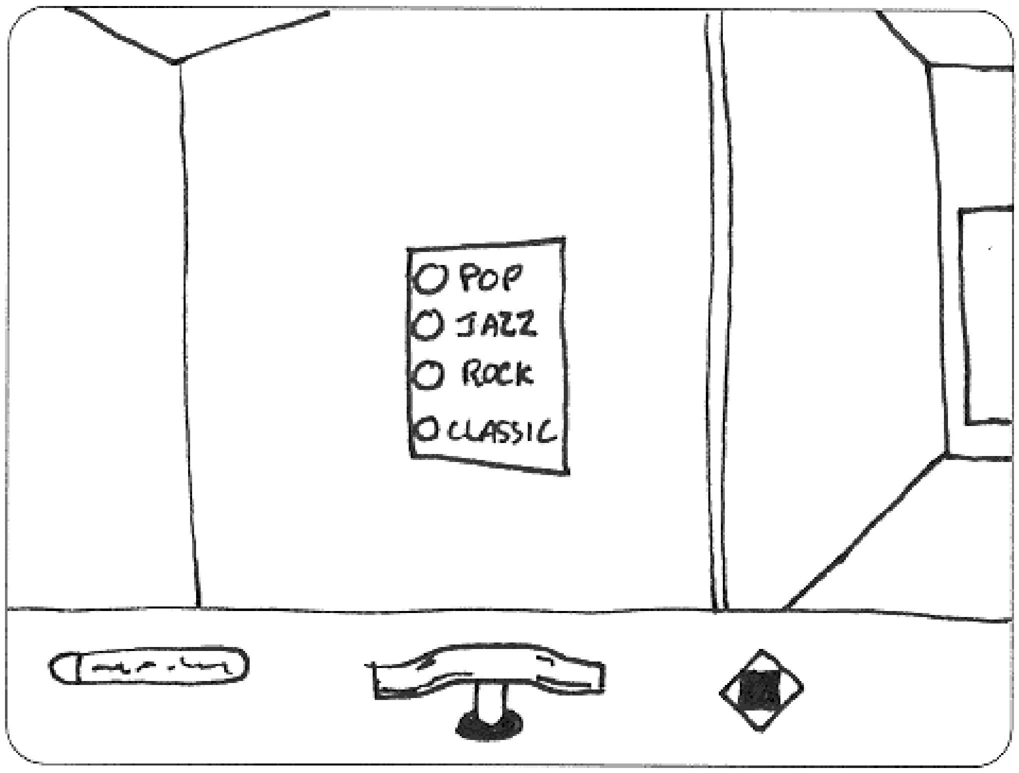


Figure 6.15: The music elevator in the VIRGILIO demo based on a screenshot in (Massari *et al.*, 1997).

The demonstrator described in the literature discusses querying a music database. The user finds themselves in the lobby of a building with a lift in front of them representing the database. Inside the lift, the buttons to select floors correspond to different types of music, as illustrated in Figure 6.15. The user might select 'pop' for example. On exiting the lift a corridor is presented which has a door for each artist in the database. As can be seen in Figure 6.16, the labels above the doors name the artist whose information is stored inside the virtual room. When the user enters a room, they are presented with objects that represent the information on the artist such as posters, songbooks, and records. By interacting with the objects they can listen to audio files, view images of the artists or read the song lyrics from the books.

The fact that people can naturally navigate around a building is the cornerstone of the metaphor. However, it is unclear that this interface is any easier to use than selecting artists from a list and then viewing a list of the material available on that artist. The added geography of the interface will certainly slow down the speed with which the user can access the information they are interested in.

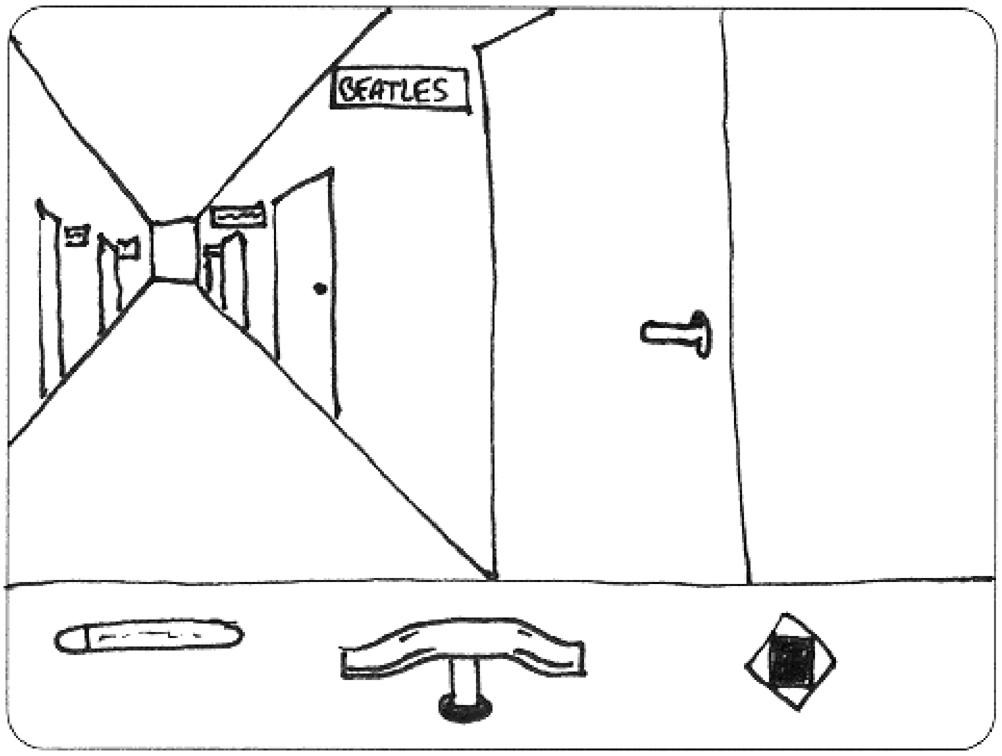


Figure 6.16: The corridor of musicians forming part of a VIRGILIO visualisation based on a screenshot in (Massari *et al.*, 1997).

If they decide they want to then find some information on ZZ Top for instance they will find they have to leave the room, change floors using the lift, and then walk down a very long corridor before arriving at the information they want. Here, the metaphor interferes with the users ability to carry out a task rather than simplifying it. This of course, may be a reflection on the clear and simple database they chose for the illustration, but is symptomatic of many visualisations that we encounter.

6.5.13 Cybermap

Cybermap (Gloor, 1991) is an automatically generated overview map specifically designed to assist with navigation through hyperdocuments. Using information retrieval techniques, nodes within the hyperdocument were clustered into hyper-drawers. The user can call up a map at any time that shows them their current location and any accessible location from where they currently are. The map display uses a form of fish-eye filtering (Furnas, 1994) to help present the information as clearly as possible.

The system makes no use of any structural linking within the hyperdocument, relying solely on the pre-built index generated by the information retrieval process. In addition, the system uses information about the user that has been collected, both actively, from a user profile, and passively, by monitoring the users actions. The profile is principally used to filter out documents that the user is least likely to be interested in, thereby reducing the number of nodes represented on the map.

Although the map is generated dynamically for the user, the index creation is an off-line process that must be repeated whenever new nodes are added to the system. This allows authors to customise the indexing process to take advantage of expert knowledge, and also reduces the time taken to calculate the maps on the fly.

6.5.14 AMAZE

The AMAZE system (Benford, 1995) uses 3D graphics to provide a novel interface to the construction of queries to a database and the representation of results. The user interacts with a 3D representation of the schema and builds the query on it. The results are presented to the user as an organised series of 3D objects.

To construct the query, the user moves around a three-dimensional representation of the schema. When they find an entity class they wish to perform a query on, they click on it and define their query. They can carry this process out on a number of different entity classes if they wish. Each set of added constraints is shown as a dangling cube from its parent entity class. This gives the user a visual indication of the query.

The results are visualised by grouping resulting instances of the same entity class into sets and colour coding them, with each resulting instance being presented as a cube within the result space. A typical query schema is shown in Figure 6.17.

6.5.15 3D File System Navigator (FSN)

Developed by Silicon Graphics (Silicon Graphics, 1995), the File System Navigator (FSN, pronounced fusion) provides a cyberspace rendering of a standard filing system. The directories in the filing system appear as pedestals on a landscape, rendered in 3D. The pedestals are arranged to reflect the hierarchy of directories

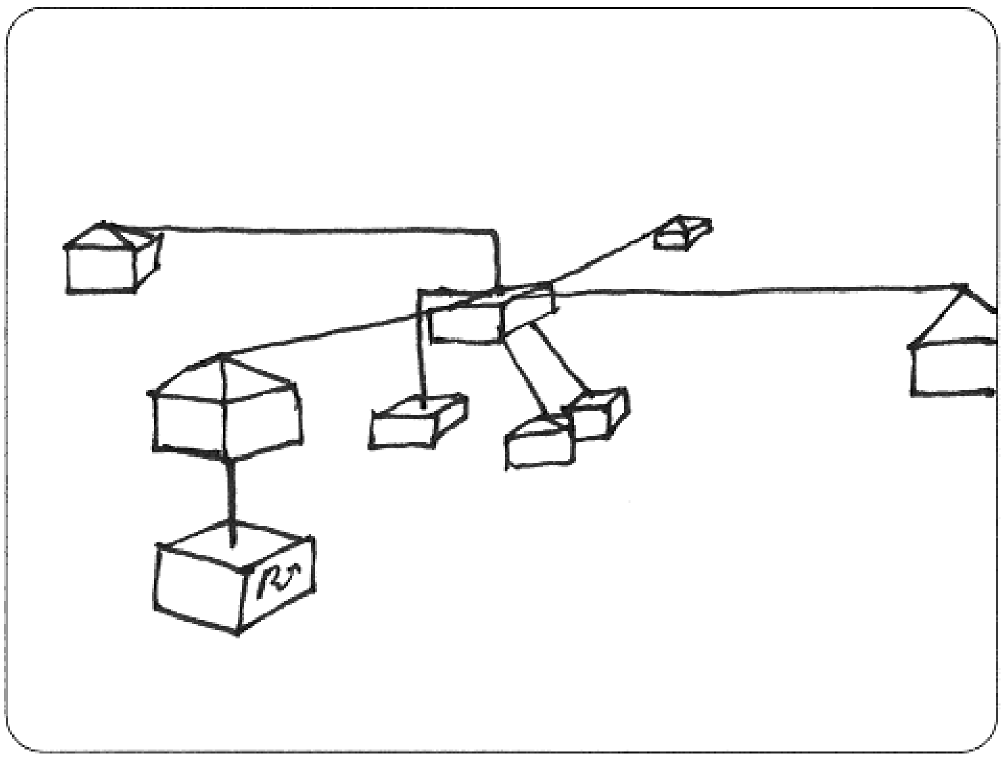


Figure 6.17: A query visualisation in AMAZE basde on a screenshot in (Benford, 1995).

in the filing system, with efforts being made to avoid overlapping of the pedestals that might cause problems of occlusion. The height of the pedestal reflects the number of files in the directory, which are represented by boxes on top of the pedestals. The system also makes use of colour to indicate the age of the files. Figure 6.18 is a sketch of a FSN screen.

User can move around the file structure by flying around the landscape to gain different views. To access the files users simply click on the box they are interested in.

Although used to provide a visualisation of a filing system, the general information landscape could be used to represent other graph and tree structures. The visualisation remains fairly static, mainly due to the limited file information it displays (size, age). This could perhaps have been extended to include information on which files are currently opened, if visualising a multi-user file space. The limited visualisation does have the advantage of a consistent mapping however, which the users can quickly become familiar with.

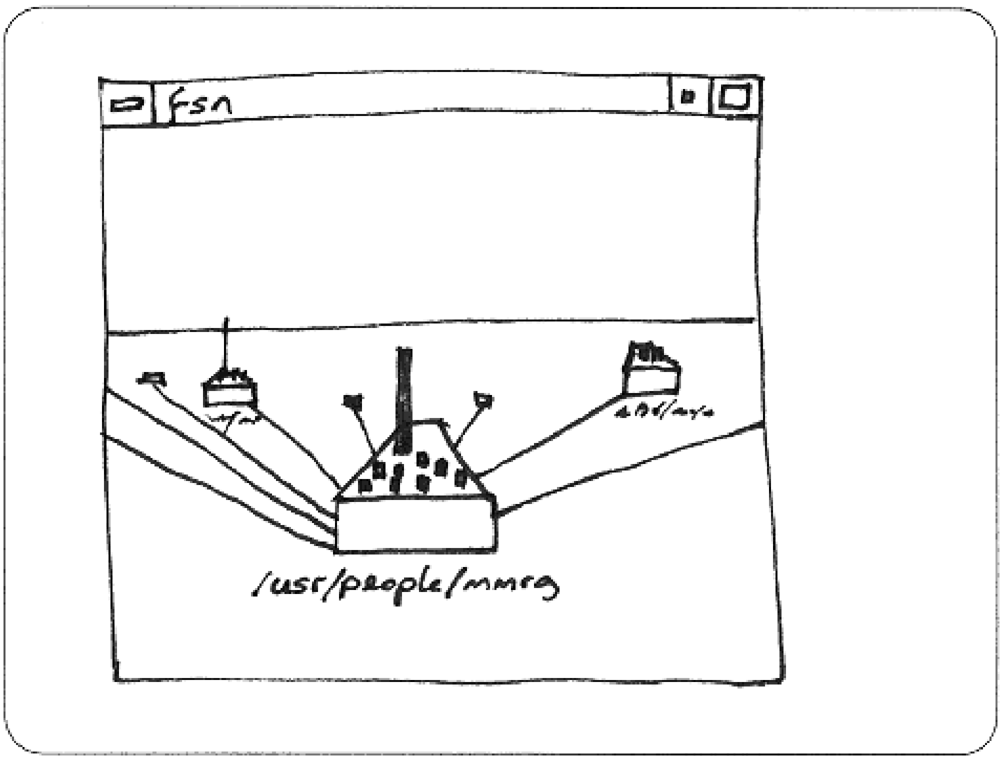


Figure 6.18: The File System Navigator interface based on a screenshot found at (Silicon Graphics, 1995).

6.5.16 Narcissus and Hyperspace

The Narcissus system (Hendley *et al.*, 1995) is designed to visualise large sets of computer based information with a view to showing the user the semantic structures within. Objects are placed in the visualisation as spheres, to represent the information being examined. This might be a software engineering system or a collection of web pages. All the objects in the system are given a behaviour that determines their movement in the space. The rules determining the behaviour are common to all the objects and can be summarised as

- All objects exert a repelling force on other objects.
- Active relationships between objects lead to attractive forces being exerted between related objects.

The result of these behaviours is that the objects in the system are self organising. They will naturally move toward similar objects and away from objects with no relationship. After a number of iterations a reasonably steady state can be achieved with clusters of documents appearing, illustrating semantic structure

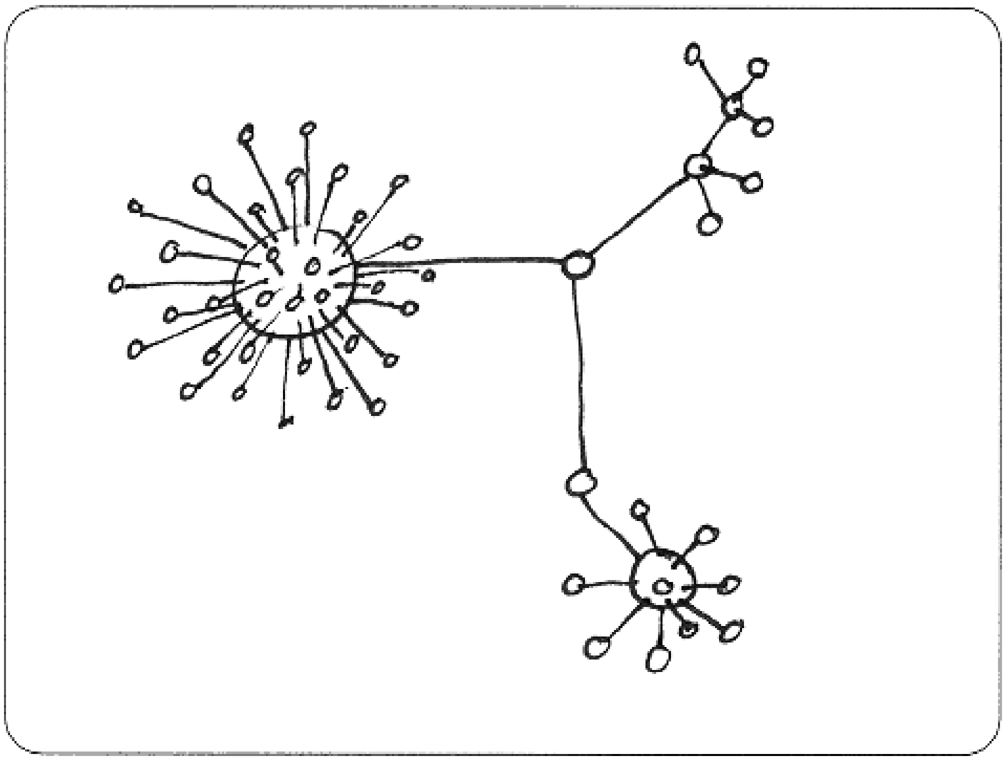


Figure 6.19: An example of the Narcissus visualisation based on a screenshot in (Hendley *et al.*, 1995).

within the document set. Figure 6.19 shows a set of objects having reached their final state and the emergent structure this has produced.

Because the forces in the system result in accelerations it is possible for documents to pulse or even orbit each other. It is claimed that this motion can form a valuable part of the visualisation. However, this continual motion can place a computational overhead on the system, so alternative approaches were used such as forces resulting in velocities rather than accelerations. This is far more likely to result in a steady state.

Within the space, users are free to move around and manipulate the objects. Users can also exert some control over aspects of behaviour of the objects. In order to provide more information on the objects to the users, techniques were used such as labelling the objects or mapping icons onto the surfaces of them. Among other techniques used was agglomeration to reduce the structure of the visualisation by merging similar objects into ‘super’-objects. If the user examines the super-object closely the more detailed structure is revealed.

One of the problems cited about such self organising systems is that even small changes to the system such as the addition of a new object can result in quite dramatic changes. The similarity of objects makes tracking a particular object during the re-organisation phase quite difficult. The importance of animating the re-organisation smoothly becomes even greater because of this.

Hyperspace

Hyperspace (Wood *et al.*, 1995) was developed from the Narcissus system. It is used to visualise web documents, with the objects in the visualisation being web pages joined by links. The links provide the attractive force between objects, acting as 'springs' during the re-organisation phase. The system is highly dynamic, with each new web page browsed, the document is parsed and all documents reachable from the page are added to the system. This causes re-organisation of the overall structure to accommodate the new information. In the visualisation, the size of the sphere is representative of the number of links from the page. An alternative approach would be to make the size of the sphere proportional to the number of hits on the page, providing the viewer with an indication of the importance of the document with respect to other web users. Key, well read, documents would then appear as large objects in the visualisation.

6.5.17 SHriMP Views

SHriMP (Simple HieraRchIcal Multi-Perspective) Views (Storey & Müller, 1995) are designed to enable graph layout adjustment while preserving various properties of the graph. The properties which it is suggested should be maintained in order to help preserve user's mental maps are listed as :

Orthogonal ordering. Maintaining the horizontal and vertical ordering of points.

Clustering. Ensuring that close nodes in the original graph remain close in the distorted graph.

Topology. Keeping the distorted graph homeomorphic to the original graph.

It is however, impossible to distort a graph and retain all of the properties listed above, therefore compromises must be made and different strategies reflect which of the properties are retained.

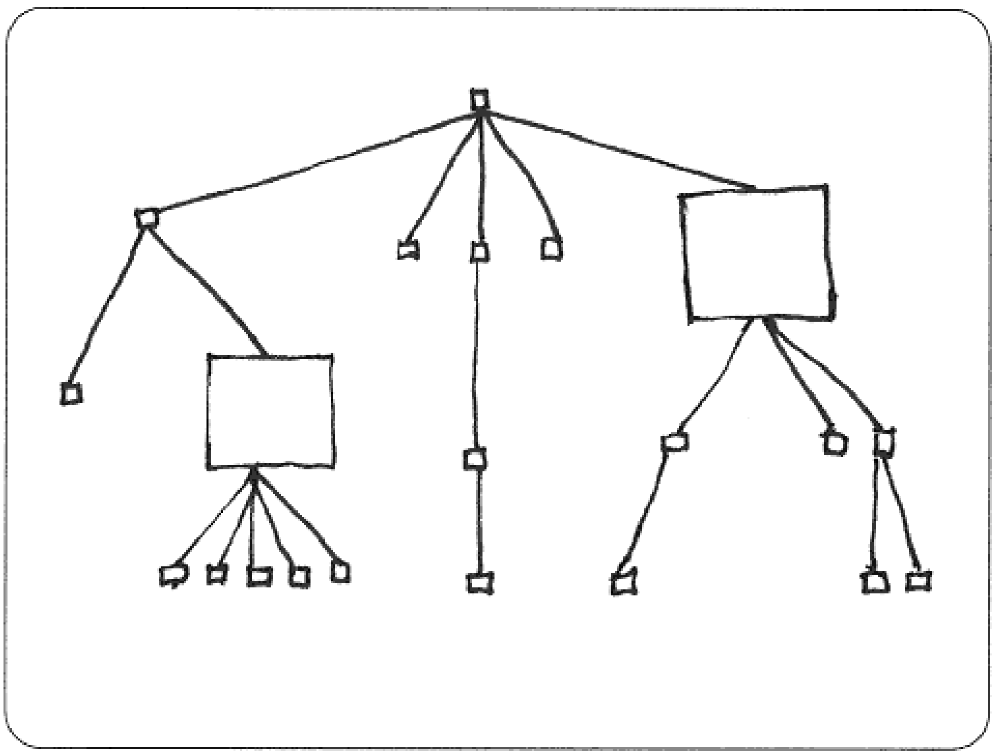


Figure 6.20: An illustration of the SHriMP graph layout approach based on a screenshot in (Storey & Müller, 1995).

The basic SHriMP view works by taking a graph and enlarging the focus node so it occupies more screen space. The surrounding nodes are moved away from the current node preserving their relationship to it. This would place the nodes off of the screen, so they are reduced in size in order to remain within the screen area. The overall effect is that the focal node has grown in size, squeezing its siblings into the remaining space available. In the distorted graph, no nodes are left overlapping if they weren't overlapping originally. The algorithm draws heavily on fish eye approaches to screen layout (Furnas, 1994). A distorted graph with multiple focus points is shown in Figure 6.20.

Depending on the nature of the graph it can be desirable to ensure that different properties are retained. For example, with simple grid layouts parallel and orthogonal relationships are often most important. For layouts such as underground maps, the proximity relationships may be more useful to preserve.

The algorithm can be extended for multiple focal points on the same graph in a similar way to the multiple focus fish-eye techniques. Unlike most of the other visualisations mentioned here, the view is 2D, designed to be easily incorporated on the desktop. The algorithm has been used in an application for visualising software call graphs and dependencies.

6.5.18 StarWalker

The StarWalker virtual environment was developed at Brunel University by Chaomei Chen (Chen *et al.*, 1999). A multi-user virtual environment, created using VRML, it emphasises spatial models, semantic structures and social navigation metaphors as a means to interact with document collections.

The VRML environment is built on top of Blaxxun's Online Community client-server architecture which provides the underlying multi-user communication. Users enter the virtual environment through various types of viewport.

Within the world, users are represented by avatars and the documents represented by spheres. Links connecting the documents are represented by cylinders. The length of a cylinder is used to represent the semantic distance between the documents and the diameter represents the similarity of the two connected documents. The colour of the spheres was randomly chosen to help distinguish them, however it is not clear from the literature whether the colours are maintained for a given document from session to session (Chen, 1999).

By allowing the user to move through the document set, the problems of focus versus context are reduced as users can 'back off' from the document set to gain and overview of the data, or move in close to gain a more detailed view of a particular portion of the dataset. As the user moves closer, more information can be revealed to them. For instance, as the user approaches a cluster object in the virtual world, the full structure within the cluster will be revealed.

6.6 Problems with Current Visualisation Systems

Visualisation systems tend to be inherently closed systems. This manifests itself in a number of ways.

- The visualisation operates on one form of data only, more often than not the target domain directly affects the construction of the visualisation software.

- The visualisation is based around a single metaphor with a single interface accessible to the user. The rigidity of the metaphor is often accounted for by prior knowledge of the dataset being visualised.
- In a multi-user system, all users are forced to view identical visualisations of the information.
- Users can only customise the visualisation in a restricted fashion, if at all. Furthermore, one user's customisation of the visualisation may well affect another users view.
- Many visualisations make use of spatial layout or object appearance, but infrequently exploit both to their full potential.

One of the main problems with the visualisation systems described above is that the user often experiences the information space vicariously in that the space has been defined and created by someone else. Usually, the mapping of real to virtual objects is hard-wired and governed by a system that the reader must learn and cannot alter. Where this imposition of metaphors is used wisely it can of course be very effective and often the structure of the information lends itself to a particular representation. However, where the data is more abstract and takes on new significance's as the readers context changes, such hard-wired approaches begin to break down and the readers desire to impose their own metaphors and spatial layouts on the information becomes greater.

These restrictions can often be 'papered over' within a single visualisation and quite often the specific targeting is used to reduce cognitive overhead for the user. The problems can arise however if the user wishes to combine two types of data within a single visualisation. In many cases this is simply not possible. Where it is possible, it would often be desirable to modify the metaphor to incorporate the new data more fully.

6.7 Conclusions

Visualisation systems have migrated over the years from specialist hardware systems to software that can run on standard workstations. 3D modelling languages such as VRML have become integrated with hypermedia systems and have been seen as a way of helping users understand the underlying information available to them. Many different approaches have been tried and systems developed to tackle the issues of presenting visualisations of hypermedia information. Most of these systems however are tied into particular hardware, metaphors, or specific hypermedia systems. This often prevents their re-use in other ways, and makes modification to the visualisations problematic.

In order to try to overcome these problems it is necessary to 'open out' the visualisations enabling different forms of data to be incorporated using different metaphors which can be customised on a personal level by the user. This situation is not unlike the problems addressed in the hypermedia communities when embedded linking restricted hypermedia systems, closing off possible alternatives. The next chapter will investigate the application of an 'open' approach to the generation of visualisations and present a novel architecture that seeks to solve some of the problems of closed visualisation systems.

Chapter 7

Minerva : A Framework for Visualisations

7.1 Introduction

Ultimately, the role of many interfaces is to provide the user with a representation of the underlying data. This may seem obvious, since at the simplest level the underlying data is stored as binary information which is converted into some form of symbolic representation, be it text or numbers. A further level of representation exists above this however which I will refer to as the representation of data objects. I use data objects as a broad term that can encompass text strings, individual numbers, or indeed whole images stored electronically. For simplicities sake I'll choose basic examples although the principles are intended to be scalable.

In some cases providing a representation of a data object may just involve displaying the raw object itself. For example, where the data object is text it can be rendered onto the screen. Even here however, the representation is dependent on additional information such as font and size. Frequently, however, the representation of a data object will involve the translation of the data into some more user friendly, or user preferred representation. Figure 7.1 illustrates the variety of potential representations of a date object that has been stored electronically.

Dates appear in many different formats. Internally stored as perhaps a 32bit integer, it can be converted into many different forms for display. Whether you

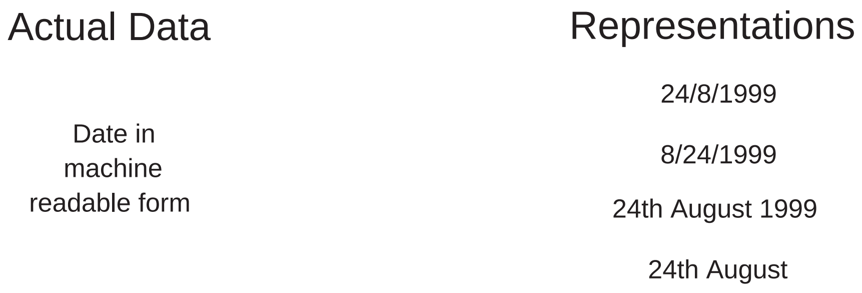


Figure 7.1: The representation of actual data.

are English or American may dictate which way round you want the month and day when expressed in a concise forward slash separated format. Various more literary formats can also be created which use words for the month etc. Each of these representations needs to be created from the original machine-readable stored format of the date.

To provide open interfaces, an abstraction has to be created between the information to be presented to the user and the presentation mechanism itself. As can be seen from Figure 7.1, in order to allow the greatest flexibility in presentation, the maximum amount of information has to be passed to the interface. If the interface were passed the piece of text '3rd March' it becomes much harder, if not impossible, to represent the date in alternative ways, both because of the initial information being textual and also due to the lack of additional information such as the year.

The rest of this chapter will look at the design issues in creating an open interface and discuss the implementation of a prototype system called Minerva, which applies the design to the Microcosm hypermedia system. In the design discussions, the Microcosm system will be drawn on heavily, but serves only as a readily available testbed and the principles illustrated could be generalised to other systems.

7.2 Design

In the design of the original Microcosm architecture, hypermedia tools were constructed as black box modules that received and sent messages. Each module would store its hypermedia information internally, often in its own format. This information would be inaccessible to other modules except if passed out in a fixed message format. Some modules, for example the history filter, kept their information private and displayed the information directly to the user in their own interface. As originally implemented no other module was able to access the history information in the system. In order to separate the underlying data from the interface it is necessary to move away from the black box approach of modular systems and allow access to the internal information in constructive ways.

In his work on the advisor agent (Wilkins, 1994), Wilkins modified the Microcosm navigational tools to provide their information on request. The process used a message passing system where a request for certain information was made and the information passed back to the advisor agent. As part of this message passing process, the navigation information was normalised to a restricted format consisting of a tuple combining a document ID with a percentage representing how important the navigational tool rates the document. By normalising the navigational information it became easier to combine the information from different sources into a cohesive set of suggestions. The results from different modules could be weighted to influence the results, for example the history filter would be given a negative weighting so that documents which had already been viewed would be less likely to be suggested to the user.

One of the main problems of this approach however is that the normalisation process sacrifices information in order to provide easier mechanisms for weighting and combining the data. By discarding information, less flexibility exists in the use of that information and the normalisation, by its very nature, dictates the representation provided by the advisor agent's user interface.

7.2.1 Design Criteria

When designing the framework, a number of different criteria were established which the framework should adhere to. These are listed, in no particular order, below :-

- The framework has to be extensible to enable additional components to be added without recompilation of the core system. This can, to a large extent, be achieved by having well defined APIs to the key modules. This allows for the replacement of modules by simply switching libraries, enabling modules to be chosen for specific tasks and a mix and match approach to the creation of interfaces.

- The system has to be able to handle a number of different types of real world data simultaneously. This includes the requirement that the system be able to function without specific knowledge of the objects it contains. By using a rigid API alongside an open format for holding objects this should be achievable.
- The system must cater for a wide variety of interfaces. The initial intention was to concentrate on 3D interfaces, utilising their expressiveness, however in order to maintain an open stance, the system should remain applicable to the construction of 2D interfaces or even experimental interfaces such as a purely audio based interface if this were desired.
- To maximise the openness of the system, it should be able to adapt to a variety of input and output devices. The ability to plug in modules to the system which allow it to utilise specialist hardware such as touch screens, trackballs and other such devices is seen as important.

With these criteria in mind, an architecture was set out.

7.3 The Minerva Architecture Design

The prototype system is named after Minerva, the Roman goddess of Wisdom. Figure 7.2 is a diagram of the Minerva framework. The data abstraction is represented by the clear separation of the real world objects, i.e. the information held within the system and the virtual representations of the information.

Three classes of module can be identified within the architecture. These can be categorised as :-

- Modules dealing with real world objects and actions.
- Modules dealing with virtual objects and actions.
- Modules translating between the real objects and virtual representations.

The following sections describe each of the components of the framework and their function within the system in more detail.

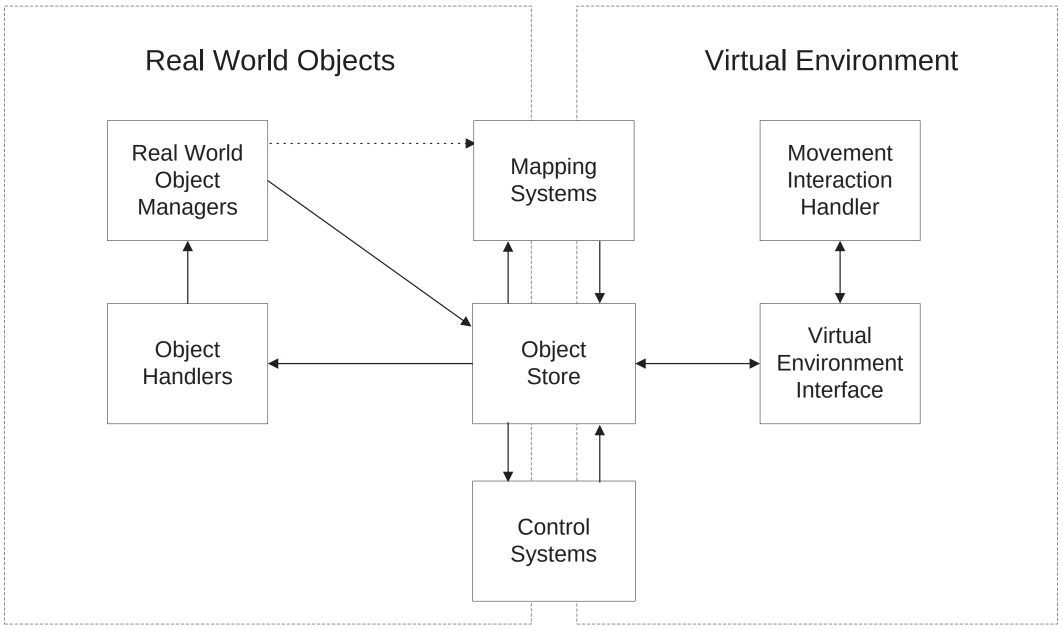


Figure 7.2: The Minerva architecture.

7.3.1 Real World Object Managers

The real world object managers keep track of the real world objects themselves. It might be a document management system, or a navigational tool that stores navigational information, such as a history device. The objects or object information are transferred from the real world object managers to the object store. In practice, a real world object manager and any object handler related to it may well be implemented within one module. This would keep all functionality relating to documents for example, in one place. A number of these modules will interact with the object store during the session. A number of different processes can supply the object store with information about the same object. The information is aggregated in the object store.

7.3.2 The Object Store

The object store is a holding place for all the real world objects. It acts as an indirection layer between the mapping systems and the object handlers. The components describing the real world objects will only need to interact with the object store. The central function for the object store is to allocate a unique object ID that can be referred to both by the real world modules and the virtual environment. When an object is altered or interacted with in the virtual world the object store is informed and can pass information back to the real world object.

In a similar manner, when the real world object is changed, the object store is informed and can remap the object and pass on the new knowledge to the virtual environment.

It should be noted that a central object database is duplicating information stored elsewhere in the system. There is no reason why the object store could not simply act as an index between a unique identifier for an object and the different modules holding information about the object. A central database has been chosen to simplify the communications of the system for this prototype and also to provide a primitive form of caching to speed up the access to the information.

7.3.3 Mapping Systems

The mapping systems are responsible for converting information about the real world objects into attributes for a virtual object to be rendered by the interface module. The modules are arranged in a chain, with messages being passed along the modules in a similar manner to the Microcosm filter chain.

The output of the mapping system is a virtual object, which can be rendered by the virtual environment interface. The format of this virtual object might be specific to the rendering engine, or possibly be in a standardised format such as VRML for direct use with a VRML rendering interface.

Although some mapping system may be written with a specific interface engine in mind, there is no reason why generic mapping modules cannot be created which are applicable to a number of different interfaces.

There are a number of different approaches to mapping the real data to virtual representations.

Direct information: In this case, the virtual attributes of the object are included as part of the real world object information. No mapping is required, the information being extracted directly. An example of this might be a file manager that lists the filename for the user. There is no mapping involved between the filename and the text presented to the user. The drawback of this technique is that it is hardwired to a particular virtual representation and can require knowledge of the representation by the real world object manager.

Attribute mapping: With attribute mapping, a virtual world attribute is associated to one or more real world attributes of the object. Mapping techniques, both functional and direct can be provided in the form of scripts, which the mapping system uses to convert between real and virtual properties. The dotted line on figure 7.2 between the real world objects and the mapping system is used to indicate that the real world object can provide scripting information to aid the mapping system. An additional module that generates mapping information could of course provide this information. To give an example, information about the number of pages of a document could be mapped to the size of the virtual object in the visualisation.

Automatic type mapping: If the system is heavily typed and the mapping system is given information on how to map between types, then it would be possible to script a mapping such as Object Type to Colour. When the mapping system finds this, it can produce its own map of type to colour, based on its knowledge of the types. It would be important that the mapping be consistent if the user is to make use of the information from session to session. Once generated the mapping can be stored and a key produced to provide information to the user.

7.3.4 Virtual Environment Interface (VEI)

The Virtual Environment Interface (VEI) co-ordinates all interactions with the user in the virtual space and is responsible for rendering the virtual representations on the screen for the user. As input it receives a number of virtual objects which it renders using the virtual attributes of the objects. Where attributes don't exist it will use default values.

In the case of a 3D interface, the three dimensional space is created by the interface and the user positioned within this space. The user effectively views the space through a virtual camera in the 3D world. The user is able to control the camera and by doing so control what they are looking at within the 3D world. The camera metaphor is used by a large number of 3D systems as a simple mechanism to describe what is happening within the interface.

The user is also able to interact with the space and the objects they can see within it. Movement within the space, i.e. controlling the camera position, will involve the passing of the users mouse and keyboard actions to the movement interaction handler, which then returns changes to the camera position within the environment where appropriate.

When the user interacts with objects within the virtual environment, the actions are passed to the object store. This removes the need for the Virtual Environment Interface to have any object specific knowledge, such as how to view documents. The store will then forward the action to the relevant real world modules which can then manipulate the real world object according to the requested action. If the action modifies the real world object, the modification is sent to the object store. The object can then be remapped and ultimately the representation of the object in the virtual interface may change as a result of the users original action.

7.3.5 Movement Interaction Handler

The movement handler translates input to the system such as keyboard and mouse clicks into actions within the virtual environment. These actions could be camera movements within the virtual space, or actions on an object such as selecting it. This abstraction allows for a number of different movement models to be implemented. In the case of a 3D interface, different metaphors can be implemented such as walking, flying, point and go etc. A further benefit is that if at a later date new input devices are used such as datagloves or trackballs, they can be easily incorporated without rewriting the Virtual Environment Interface.

7.3.6 Control Systems

In order to create some visualisations, or indeed to integrate some navigational devices it may be necessary to control the number of objects sent from the object store to the virtual environment interface. There may be a number of reasons for this. If the number of objects in the store is very large some form of filtering might be required. Alternatively, some navigational tools may be best implemented by restricting the objects in the store that are displayed. Examples of this are discussed in later sections.

7.3.7 Object Handlers

The object handlers receive messages from the object store which indicate that an action needs to be carried out on the object. It is the job of the object handler to translate the virtual action, such as the object was clicked on, into a real world action for example launch the document. This abstraction gives greater flexibility to implement new interactions with objects within the virtual environment. Often, the object handler will be a part of the real world object manager although there is no requirement for this.

7.4 From SHEP to Minerva

When comparing the architectures of the SHEP system as shown in Figure 5.4 and that of the Minerva system shown in Figure 7.2 it is easy to see the similarities in their structures. Both have a central communication system that routes messages between the various components of the system. The SHEP architecture has a chain of shepherds that modify the information in turn, with the results being passed back to the central hub. In the Minerva architecture a chain of mapping libraries can be identified which modify messages to create virtual objects from the original real world objects supplied by the object managers.

Where the SHEP framework connects together only two types of processes, namely sheep and shepherds, the Minerva framework provides facilities for a wider range of tasks. Fundamentally though, both systems work towards presenting information and it is here that real difference is observed. SHEP sought to control purely the interface of the system, with the content of these interfaces still in the hands of the individual modules. The Minerva framework abstracts the information at a deeper level. The raw hypermedia information is what flows through the framework before it has been processed into scrollable lists or map diagrams.

At one stage of the design, the extension of the SHEP framework to include the underlying information was considered however the need for a bigger variety of controlling processes, and the greater separation between the interface representation and the underlying data representations made this too problematic. So instead, despite deep rooted similarities, the SHEP and Minerva frameworks currently exist alongside each other within the Microcosm system.

7.5 Navigation Tools

One of the main aims in designing an open interface framework is to allow the integration of navigation information with other aspects of the interface. A number of different navigation tools have been discussed in previous chapters and this section will examine how the navigation information can be utilised by the interface. A history tool provides a simple example to start with.

The history tool traditionally presents the user with the list of documents that they have seen so far. Internally, the information it stores may be as simple as a list of documents that have been viewed by the user. The history navigation information could be represented in a number of different ways. Two possible implementations of this within the Minerva framework might be as follows.

A trail of bread crumbs: As the user moves around the document set in the virtual space they might leave behind them a trail of virtual objects. Each object would be a virtual representation of a real world piece of history information. These objects would represent the path taken by the user. They might form discrete objects or perhaps be a continuous line through the virtual space. As was suggested by Bernstein (Bernstein, 1988), a limited length breadcrumb trail might be implemented to avoid over cluttering the information space.

Use of document properties: Rather than add the history information as a separate object in the store, it can be added to the document object which will have already been added by the document management system. A property of the object, for instance colour, could then be allocated to the history information. This would allow a simple visible indication to be provided as to whether the user has viewed the document before. A side effect of this scheme is that it would not be obvious in which order the documents had been viewed, but simply that it had been looked at. Shades might be used to indicate time, the darker the shade, the more recently the document had been viewed.

As an aside, if the virtual space has been constructed so that virtual documents have unique and readily identifiable appearances then the user is more likely to remember having visited a document and thus the path aspect of a history tool might prove more important.

One problem associated with the path approach however is that if the document locations in the virtual space are dynamic, then the path cannot be fixed as it will no longer correctly reflect the movement of the user through the document set once the virtual objects have changed position. One way of counteracting this would be to make the extrinsic properties of the path related to the extrinsic properties of the virtual objects. This would metaphorically tie the history path to the documents, thus maintaining a correlation between the documents viewed and the path of the user. It would not maintain any relationship between the path and documents that were passed by the user while travelling along the path but not viewed by the user.

7.6 Use of the Microcosm Hypermedia System

The Minerva prototype has been initially implemented on top of the Microcosm hypermedia system. The main motivation was a desire not to re-invent the wheel. Many aspects of the Microcosm system provided working solutions to basic problems of the framework. Where these problems did not constitute the main objectives of the framework it seemed sensible to solve them using software that was at hand rather than expending effort in minor parts of the system so detracting from the effort available to spend on more important aspects. A number of aspects of Microcosm lent themselves to this, some of which are briefly covered below.

- The Microcosm system is open in that it is easy to add new modules without re-compilation. This provides a flexible architecture with which to integrate the Minerva framework.
- Microcosm has a powerful and extensible message format which can be used to represent both the real world objects in the object store and the virtual objects.
- Microcosm has a hierarchical registry that can be used to hold the mapping information as well as any virtual environment information that needs to be saved from session to session. The hierarchy allows different views and mappings to be stored for each user, allowing them to create their own

personal interfaces. It also allows separate settings for each application allowing different visualisations to be created for different application areas.

- There is a document management system within Microcosm that can be used as an underlying information system. Mechanisms are already in place to extract information about the documents and display them in the correct viewers, providing ready made object handlers.
- There are a number of hypermedia navigation tools implemented within Microcosm that can be easily adapted to provide information to the object store. These include linkbases, a history tool and a search engine as well as various viewers used to display the results of these actions.
- The open architecture of Microcosm makes it easy to plug in any new navigation tools that are designed specifically to take advantage of the Minerva framework.

7.7 Implementation

The sections below describe the implementation of the Minerva prototype. Figure 7.3 shows the architecture, including a number of Microcosm hypermedia components adapted to provide the real world data.

7.7.1 A Supporting Metaphor

The framework design lists all of the relevant components needed, but it was felt that an encompassing metaphor might help in the description of the framework and its API. The idea of a film production was taken as this maps nicely onto the various components and helps describe not only the individual processes but also the way they interact with each other. Figure 7.4 shows the framework as represented by the film metaphor.

In providing a visualisation to the user, the object of the framework is seen as presenting a scene. The users viewpoint of the visualisation is represented by the camera. One of the reasons for creating the metaphor was to simplify the descriptions of the framework. Terms such as real world, virtual, objects, processes, handlers, quickly become overloaded and cumbersome. By using a clearly defined existing language with no computer terminology, some of the ambiguity and overloading can be removed.

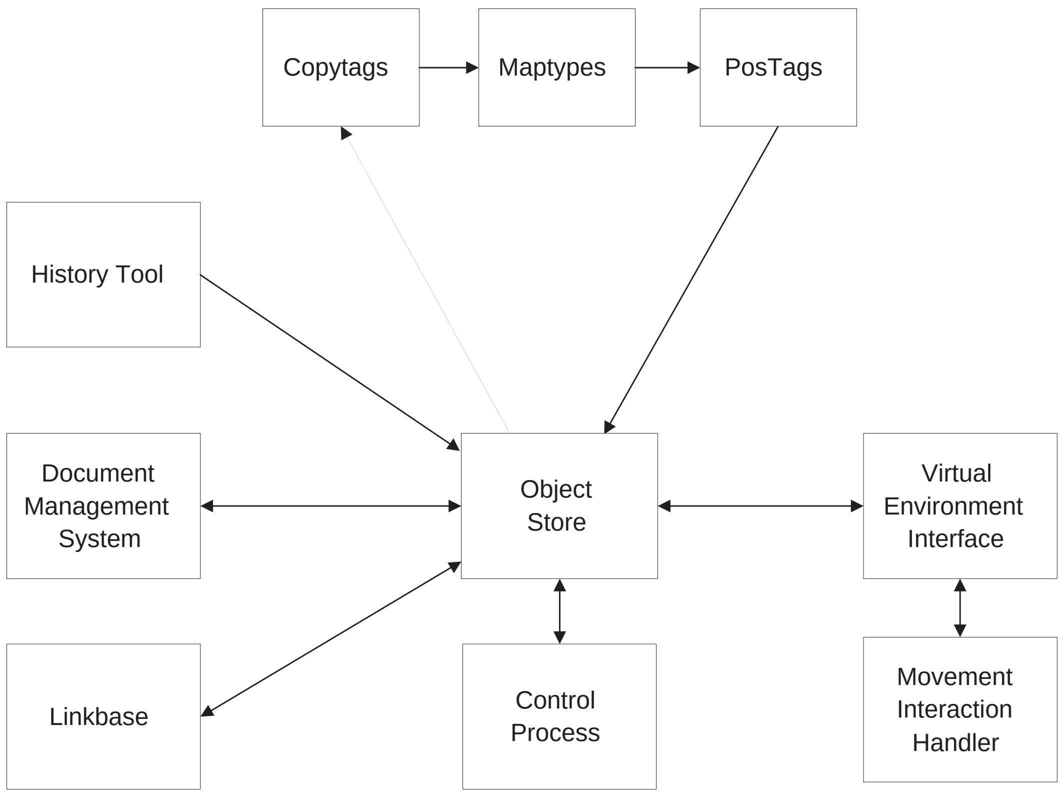

Copytags
Maptypes
PosTags
History Tool
Document Management System
Object Store
Virtual Environment Interface
Linkbase
Control Process
Movement Interaction Handler

Figure 7.3: The implementation of the Minerva prototype

The Objects Populating the Scene

Three different types of objects can be added to object store, and hence the scene. The list below describes the differences between them.

Actors/Characters Each real world object that is added to the Minerva store is referred to as an actor. When the actor is added to the store, it will be mapped to a virtual representation. This virtual representation is called a character in the metaphor. These characters can be interacted with in the interface that may lead to actions being taken on the real world object or actors.

Props Some objects in the system are intended simply to be represented in the interface but not to be interacted with. These objects have been termed props and do not require an object handler. Like actors however, they are mapped to virtual representations, which keeps the level of abstraction between the actual data and the representation required for the interface. An example might be a signpost that indicates the topic of the documents

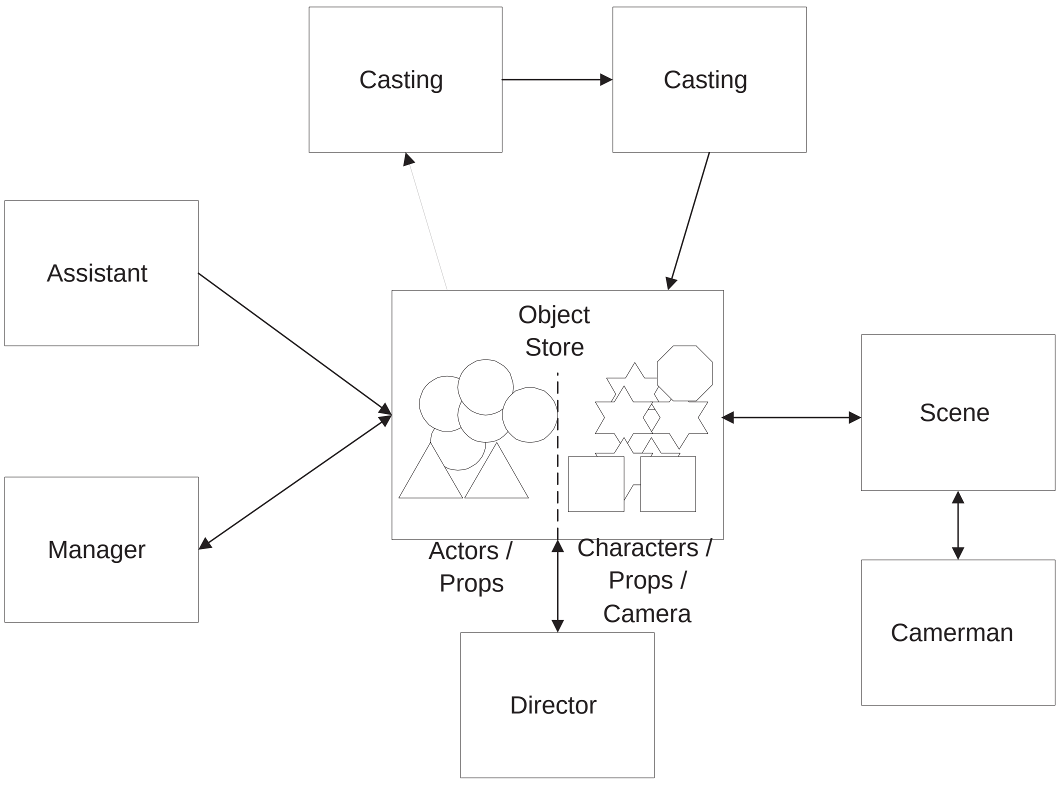


Figure 7.4: The metaphor as it relates to the Minerva prototype.

that surround it. The virtual representation provides information to the user but there is no necessity for the user to interact with it.

Camera The interface process can add a camera to the store as an object. This provides the system with information about the users current view. This information might be used by the mapping processes to arrange objects (characters) relative to the users viewpoint. For example, if the mapping works on a relevancy measure, more relevant documents can be placed closer to the user by positioning the representations of the documents closer to the camera.

The Crew

Each of the components of the framework corresponds to a member of the crew. The sections below describe each crew member in turn and how they relate to the framework and the objects that exist within it.

Managers The object handlers of the framework are represented in the metaphor as managers. A manager deals in actors(real world objects) and is responsible for them. Managers can add their actors to the current scene by registering them with Minerva store. The manager must also update the store when an actor changes, and is also responsible for removing it from the store if the real world object is deleted. When an action is taken on an actor in the interface, such as double clicking on it, the manager is contacted by the system to handle the action. A manager will invariably be responsible for handling a number of different actors.

Assistants If a process wishes to supply information about actors but is not the object handler, it can register as an assistant. An example might be a history tool that registers as an assistant and supplies information about whether a document has been viewed or not. The document itself would be supplied as an actor object by the document management system that would act as the manager. The history tool would simply supply additional information during the session about the actor.

Casting The mapping processes register with the framework as casting processes. It is their responsibility to cast the actors into characters that can be used in the scene. The casting processes are maintained in a chain by the system. Each time an actor is added to the store it is passed along the chain of casting processes with each one adding to the character information. When the actor reaches the end of the chain it is passed back to the object store where the character is stored along with the original actor. The casting processes are also called whenever the actors (real world objects) are modified, so recasting the actors into characters. Because of the chain, it is possible for character information from casting processes early in the chain to be modified or even removed by casting processes later in the chain.

Scenes The Interface environment registers with the system as a Scene. Upon registering it will request from the store all of the characters and props which it has to render in the scene. It is notified if any of the characters or props are updated. More than one scene can register and they are all notified of the updates. When the user interacts with the scene by clicking on a character (virtual representation of an object) the store is notified so that the agent that handles the corresponding actor (real world object) can take whatever action is necessary.

Director A control process can register with the system as a Director, which gives it control over which characters and props from the store are viewed in the scene. By default, if no Director is registered, all of the characters and props in the store will be rendered in the scene.

7.7.2 The Object Store

The object store serves both as an object cache for real objects and their representations and also as a central communication system connecting the different parts of the system.

The different modules of Minerva register with the object store, allowing them to send and receive messages to and from the Minerva system. When it registers, the process informs the object store as to what type of process it is, agent, casting, scene etc. This information is used to route the messages around the system.

Ideally, the object database would not be needed, rather the process would serve simply as a communication system and whenever information about an object is needed it would be obtained directly from the object agent, cast by the mapping modules and delivered to the calling process. In the case of the prototype however it was decided that caching the information would speed up and simplify the system.

7.7.3 Object Handlers / Object Managers

The Microcosm Document Management System (DMS) handles all the underlying documents referenced by the Microcosm system. The DMS registers with the Minerva system as a manager.

The DMS registers each of the documents in the system as an actor with the object store. The identifier given for each object is the Microcosm UniqueID. This enables any part of the Microcosm system to refer to the object in the object store. The object given to the store is the whole record held by the management system.

It provides a callback function so that it can receive messages from the object store. These messages might include a request to provide the latest information about a particular document, or perhaps pass on an action to be carried out on the object. One such action is the object request action. This is passed to the object store from the interface when an object has been selected in the interface. The object store passes the request on to the crew member that registered the object. In the case of the DMS, when the request is received it launches the document through Microcosm.

When Microcosm is shutdown, the Document Management System unregisters from Minerva. All the objects registered with the system by the Document Management System are also unregistered as there will be no object handler available if the user interacts with the virtual representations of the documents.

The Microcosm linkbase also registers with the framework as a manager, submitting all of its links to the object store as actors. When links are activated, the linkbase responds by following the link within the Microcosm system. More detail of this is given in a later section on navigational tools.

7.7.4 Assistants

As currently implemented, the history tool registers with the system as an assistant. It adds no objects to the object store, but instead supplies information about documents that have been registered by the Document Management System. It modifies the stored object to indicate that the user has viewed it. This information can then be mapped onto the representation of the virtual object. More detail on the integration of the history tool and other navigational devices is given in a later section.

7.7.5 Mapping Modules

The mapping modules are responsible for mapping, or casting, the objects placed into the store onto virtual representations that can be displayed by the Virtual Environment Interface. A number of different mapping modules can be plugged into the Minerva architecture, with each responsible for mapping to specific aspects of the virtual representation. Some mapping modules may well be specific to the type of information being processed, where as others may be generic enough to be usable on a wide variety of information.

A number of generic mapping modules were implemented which can be used in the Minerva framework. Three of these modules are described below.

Copytags Mapping Module

Although fairly trivial in its implementation, the copytags module provides a vital level of indirection between naming conventions in the objects provided by the object managers and the naming conventions of the interface manager.

Using the 3D interface as an example, it can display a text label when the mouse is moved over an object in the interface. It achieves this by looking in the virtual object for a description tag and displaying the value of the tag in a pop up window. Not all of the objects placed in the object store will have a description tag and the interface designer might choose to display a more appropriate tag. An entry can be made in the registry that tells the copytags module to place the value of an alternative tag in the description tag. Perhaps the object has a title field. This can be copied into description and will be displayed by the interface. The description is just one instance where the ability to copy tag values to a new tag might be useful when constructing the interface.

An alternative approach to this problem might be to directly manipulate the interface to tell it what tag to display. Although this would achieve the same result in the simple case, where multiple sources of objects exist, the interface would need to display the correct tag for each object entered. By moving the problem to the mapping stage, the interface only needs to specify the tag it will display and the mapping system can be in charge of mapping the tags to the correct values.

Postags Mapping Module

In the case of a 3D interface, objects need to be assigned 3D coordinates based upon their attributes. In some cases the objects will have spatial coordinates but where the objects are more abstract in nature the spatial positioning will have to be derived in other ways. The approach of the postags module is to produce a simple spatial layout of objects based on a textual name and where applicable date information.

The overall affect of the mapping is to produce a form of time tunnel. Originally the mapping was created for positioning bibliographic references within a visualisation. The references are all placed in a circle based on the reference name, and the year of publication is used to determine the Z order of the objects, with more recent publications being closer to the front and older publications being in the distance. This is illustrated in Figure 7.5. More details of this visualisation are given in Chapter 8.

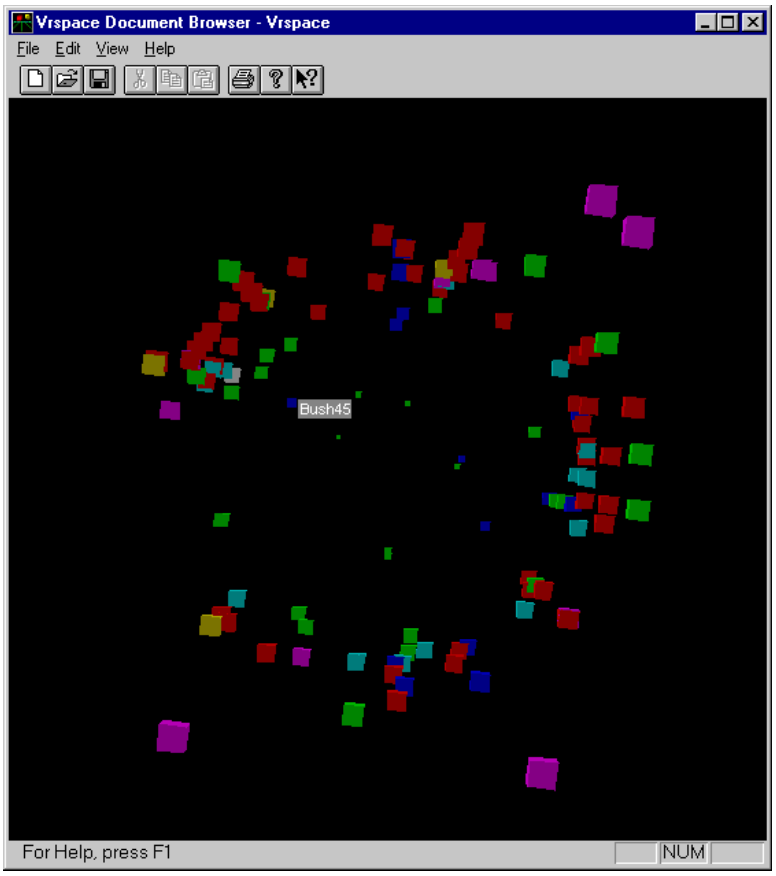


Figure 7.5: The time tunnel of bibliographic references.

The postags mapping, positions objects on the edge of a circle based on a specified tag for the object, i.e. the description. Figure 7.6 Illustrates how the objects are placed on the circumference based on the first letter of their description. The formula to calculate the x and y coordinates is given below.

$$\begin{aligned}\theta &= 2\pi \times ((\text{FirstCharacter} - '\text{a}') \div 26)\\ X &= \cos\theta \times \text{radius}\\ Y &= \sin\theta \times \text{radius}\end{aligned}$$

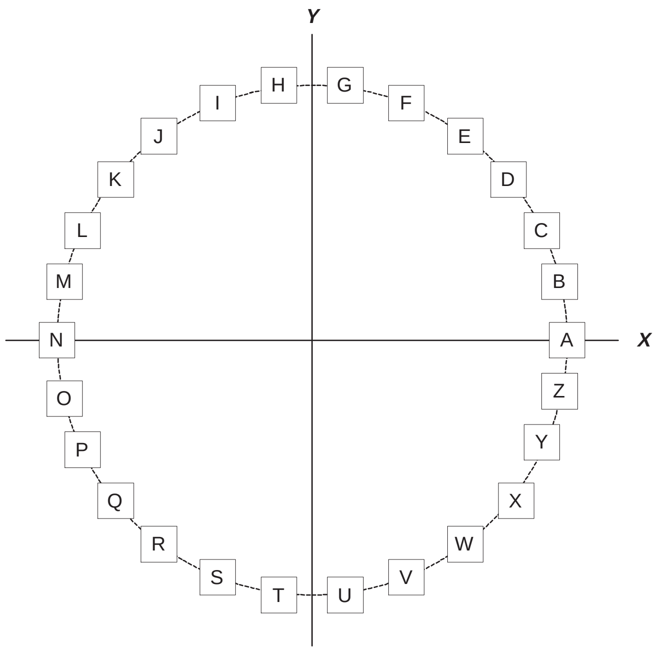


Figure 7.6: Positioning of objects based on their description.

and to calculate the z coordinate.

$$Z = (\text{CurrentYear} - \text{Year}) \times \text{radius} \div 4$$

The main problem with this basic mapping is that if two objects are from the same year and their descriptions start with the same letter then they will have exactly the same coordinates. In this case, the larger object will obscure the smaller object from view. To overcome this, the second and third letters are also used to calculate the angle although to a lesser degree.

$$\begin{aligned}\theta = {} & (2\pi \times ((\text{FirstCharacter} - '\text{a}') \div 26)) + \\ & (2\pi \times ((\text{SecondCharacter} - '\text{a}') \div 26) \div 30) + \\ & (2\pi \times ((\text{ThirdCharacter} - '\text{a}') \div 26) \div 60)\end{aligned}$$

The slight shifting in position that this causes is often enough to make an object visible that would previously have been obscured. An alternative approach would have been to add a random element to help avoid overlapping objects however the scheme adopted here has the advantage that the same object will always be mapped onto the same location, ensuring that the spatial positioning is consistent from session to session.

Maptypes Mapping Module

In many cases, an attribute of the real world object will be one of an enumerated set of possible values. Where this is the case, the interface designer might wish to map a value for an attribute to a specific value for an attribute of the virtual representation.

For example, the interface designer may wish to map the type of the real world object to the shape of the virtual representation of the object. The maptypes module allows the designer to specify the information within the registry that allows it to make a mapping from type to shape. An example mapping is given in Table 7.1.

Type	*Shape*
Text	Cube
Bitmap	Cylinder
Video	Cone

Table 7.1: Mapping from type attribute to shape representation.

The mapping is stored in the registry and loaded by the maptypes module on start-up. The hierarchical structure of the registry allows different mappings to be used for different applications, or even for different users. When the module is asked to create a virtual representation from a real data object, it searches for the Type tag in the object and if it is present, creates a Shape tag with the corresponding value. If there is no Type tag, or the Type value is not present in the table, the Shape tag is not created. If no mapping module creates a Shape tag, the interface will use its default value if it requires the attribute. To create the mapping table, the designer needs knowledge of both the real data, and the necessary virtual representation required by the interface. This enables the module to bridge the gap between the two domains.

Alternatively a mapping could be created dynamically by the module if it has some knowledge about the tag values. For example, the designer might decide that they want the mapping module to automatically create a mapping from type to colour. The module must assume that the type is a finite, enumerated set of values. If it has some built in knowledge about colour representation it can create the mapping table itself on the fly as it is presented with objects to map to virtual representations.

The module creates a blank mapping table from type to colour. As new values of type appear in data objects, they are assigned unique values of colour by the system. As the objects are mapped to virtual representations, the mapping table slowly builds up. There are three important criteria that should be met however, if the mapping table is created automatically by the mapping module.

1. The mapping module must have some knowledge of the destination domain. This could be in the form of a list of values to use, or might simply involve a numbering scheme that assigns each tag a different number. This might equate to a position on a bar chart for example.
2. The mapping has to be made available to the user in the form of a key, which explains the association between virtual representation and actual object attributes. i.e. they must be able to find out that blue represents video documents for instance.
3. The mapping should be stored at the end of the session for re-use next time. If the mapping were to be different every time the user uses the system it becomes difficult for them to build up a mental model of the visualisation.

Automatic mapping, although useful, will not always be applicable. In general, it will work best when mapping a small number of unique values for a tag to a limited set of possible values for the mapped attribute.

7.7.6 The Virtual Environment Interface (VEI)

The Virtual Environment Interface (VEI) has been constructed using RenderWare (Criterion Software, 1998). The software, developed by Criterion Software, provides a toolkit for constructing three-dimensional interfaces, both in terms of displaying objects and lights within a 3D scene and using a camera within the scene to create a first person perspective view. As with other components of the implementation, the Virtual Environment Interface is just one possible interface that can be used with Minerva.

On starting, the VEI registers with Minerva as a scene. It then requests from Minerva all of the characters which are present in the Object Store. When it receives the characters they have already been mapped from actors and hopefully will contain the attributes that the VEI uses to create the virtual representation of the object. These attributes include VRPosX, VRPosY, VRPosZ, VRColour, VRShape and Description. If any attribute is not present in the object then the interface will use its own default value for the attribute. Each virtual object is created and rendered in the interface.

As new objects are added to the store they are mapped and passed on to the Virtual Environment Interface. It will create a new object based on the information in the virtual object it has been passed, and add it to the scene. Similarly, if the object handler or an assisting module modifies an object in the store, the object is remapped and the interface is passed the newly modified object. The existing virtual object is removed and the new representation of the object added to the scene. This is carried out between refreshes so from the users viewpoint it will look as if the existing object has changed. The need to remove the old object and add a new object is only a technicality of the current implementation.

Users interact with the VEI using the mouse. Holding down the mouse buttons and moving the mouse causes various events to occur in the interface. These events are passed to any registered Movement Interaction Handlers that can modify the behaviour of the interface. The returned message is interpreted by the interface and the requisite actions carried out. If no movement interaction handlers are registered, the default action is taken by the interface. The interaction handling is simply a mapping between a user action, i.e. Left Mouse button + dragging left, and a resultant operation in the interface, for example pan the virtual camera left.

Using the interface, the user can interact with the objects represented in the scene. When the user has selected an object in the interface, the interface sends a message via Minerva to the object handler for the object. The object handler can then carry out the action specified for that event. The action might be to launch the represented document in an appropriate viewer or perhaps to display the properties of the object to the user.

7.7.7 Movement Interaction Handler

The idea behind the movement interaction handler is to provide an abstraction between the control mechanism within the interface and the results of those control actions by the user. In the original design, the movement interaction handler is shown as a plug in module with a control path running from the interface to the interaction handler and back to the module. The intention was for the interface to receive a control message such as 'left mouse button clicked over object' and pass this to the movement interaction handler. The handler would map this to an action, perhaps 'select object', which would be passed back to the interface. The interface then carries out the action, in this case informing the relevant object handler that the object has been selected.

In practice, the object handler is providing a mapping between an action and a re-action. During the use of the interface this mapping is unlikely to change. For this reason, the movement interaction handler has been modified to speed up the interface. Where an interaction could be the mouse moving, the control path is unnecessarily slow particularly since the mapping is static during the session. Rather than have every control message routed through the handler, the interface calls the movement interaction handler once at start up to request the mapping. The mapping is then stored internally to the interface and used during the session. The end result is the same with the interaction being abstracted, the difference being that the interaction remains constant during the session and cannot be modified by the handler once the interface is in action. This optimisation was implemented to help the speed of the prototype, but given a sufficiently efficient control path there is no reason why the original design could not be implemented successfully.

It is necessary for the handler to be aware of the actions that it will be sent and the re-actions that the interface can carry out. The mapping is scripted in the registry so it is not necessary to hard-wire these into the handler. Like the mapping modules, the interaction handler forms a bridge between the real world and the virtual world.

7.7.8 The Control Process

Where the number of objects in a visualisation is large, the interface designer might like some method for limiting the number of objects displayed at any given time. A control process can be registered with the framework which tells the object store which objects to let the VEI display and which not to.

This process could be automated to restrict the display to objects within a certain distance of the camera position, or might form part of a navigational tool. For example if a search engine is present, the results of a search could be used to restrict the objects displayed to the user. A later section discusses this in more detail.

Where a control process is not registered with the framework, by default all of the objects contained in the object store will be passed onto the VEI for displaying to the user.

7.8 Existing Navigation Tools

Rather than talking about adding navigation *tools* to the virtual space it might be more appropriate to talk about presenting navigational information. There are two ways in which the navigational information might manifest itself.

A hypermedia object: The information may form a hypermedia object in its own right. This would be represented as a virtual object in the space and the object would have properties derived from the information in much the same way as the virtual document objects.

Properties of virtual document objects: Here, the navigational information is used to generate one or more of the properties of the document objects. For example the colour of a virtual document might denote some navigational information about it. This is illustrated in more detail below.

Which of the above techniques is used to represent the information will depend on the type and complexity of the information. In Chapter 2, a number of different types of navigation tool were examined in detail. The sections below discuss the implementation of these commonly occurring navigation tools within the Minerva framework.

7.8.1 The History Tool

The Microcosm history information is held by the results filter, which keeps a record of all of the documents that have been launched by the system. The results filter registers with the Minerva system as an assistant. As documents are opened, it sends messages to the system updating the document objects to reflect that they have been viewed. This simply involves adding a tag called history with a value of 1. The document objects are then re-mapped to virtual representations which might result in the representation of the document changing in the interface if a facet of the object representation is based upon the value of the history tag.

An alternative approach might have been for the history device to add props to the system such as flags for instance, which sit next to the documents that the user has already viewed. The positioning of these objects would have to be calculated in terms of the documents that they are to be placed next to. A casting module for this might extract the document position from the store and place the flag slightly above the documents position. Further props might be placed in the scene which join the flags together providing a visual representation of the path the user has taken through the documents.

7.8.2 Links

A link in its most basic form can be described as connecting two objects. Where the two objectss are represented in a virtual space the link can be shown as a virtual object in the scene. Its properties include start and end positions and it is implemented as a line within the virtual space.

Here, each link is registered as an object in the scene in its own right. The positioning of the object has to be calculated based on the positions of the two objects that the link connects. This information is accessible to the casting agent from the object store. The linkbase also responds when the user clicks on the virtual representation of the link. This either causes the link to be followed, or information about the start and destination of the link to be displayed. Additional information about the link can also be encoded in the virtual representation, for example the colour of the link object can be used to indicate whether the link has been followed already.

7.8.3 Maps

Maps are often provided within hypermedia systems to provide a sense of spatial positioning to what are in fact just arbitrarily stored documents. By producing a 3D visualisation, a map of sorts is being presented as the main interface. What would be navigation tools in the underlying systems become the mapping processes in the framework. The positioning of the documents might be authored by explicitly positioning documents within the 3D space, or might be created automatically using mapping tools like the postags module discussed in the previous section on mapping modules.

Although not explicitly implemented within the Minerva framework, the two classes of map navigational tools outlined in chapter 2 can be identified within the implementation of the framework.

Global Maps

When the user is presented with a visualisation which displays all of the objects in the object store as virtual representations, they can be considered to be viewing a global map.

Local Maps

Local maps present to the user a subset of the documents and links in the hypermedia system arranged spatially. As the user moves around the visualisation they are in fact creating their own local maps in that at any given time only a subset of the documents represented in the space are visible to them. By zooming in on clusters of documents in the visualisation they are in effect moving from a global context where all the information is visible to a local context where a portion of the information is visible but at a greater detail.

It is possible to implement more explicit versions of a local map within the Minerva framework. The Microcosm local map tool displays at its centre a document and arranged in a circle around it are all the documents that can be reached from the central document by following one link. This could be generated from the information in the object store by a tool that is aware of which document is the centre of the map. Acting as a control process it could control the store so that only the relevant documents are displayed and any links that are appropriate. The positioning of the documents could be modified to provide a circular arrangement around the central document.

Alternatively, to provide a context for the local map, another approach might be tried. Rather than eliminating all the irrelevant documents from the visualisation, their appearance could be modified, for example making them darker, or partially opaque, to distinguish those documents that are relevant to the local map. The positions of the documents would remain the same, so some documents might be quite a distance spatially from the central document, but still only a link away in hypermedia terms. This would present to the user all of the same information as a simple local map, but in the context of a global map.

7.8.4 Guides

The basic purpose of a guide is to suggest to the user which documents they should view next. As with history devices, there are a number of ways in which guides could be implemented within the Minerva framework.

- Perhaps the simplest method would be to make the appearance of a document dependent on whether the guide thinks it should be viewed next. If the guide adds information to the object in the store indicating how important it is, say for example a percentage figure, this could be mapped by a casting process onto the colour of the object. The higher the percentage, the brighter the object, the more important to the user.
- A second method would be to have the guide as a virtual object in the environment. It could exist as a prop that moves around the document space positioning itself above objects of interest to the user. As the user moves around the document space they follow the guide through the space in much the same way as tourists might follow a guide through a museum.
- A third method would be for the guide to register as a director of the scene. It can then narrow down the options for the user by removing objects from the scene that it considers to be no longer relevant to the user. The user then knows that any objects that are visible are likely to include information that is useful to them. If the 3D interface contains a concept of opacity, the guide might combine this technique with the ability to manipulate the objects appearance in order to make the representations of documents become increasingly transparent as their relevance decreases. Ultimately they

would disappear entirely when they are considered completely irrelevant to the users current goal.

7.8.5 Search Tools

Hypermedia systems can use a wide variety of search tools. When it comes to integrating search tools with the Minerva framework there are a number of possible approaches.

- A simple approach would be to just visualise the results of the search. The user enters the search into the search engine using its own dialogue interface. The search is carried out and the results are added as document information to the object store. The new information would lead to new mappings of the documents in the store leading to visible changes in the virtual representations. A simple mapping might involve highlighting the documents returned as results of the search by mapping the result tag to the colour or the brightness of the virtual object.
- Alternatively, a director process could use the search results to restrict the documents being displayed in the visualisation to those being returned from the search. This makes the visualisation more akin to simply listing the results of the search. By removing documents that didn't match the search information from the visualisation the user will then only be presented with documents that match the search. This approach does discard potentially useful information however, as it is possible that the context of a matched document amongst unmatched documents might be information the user can utilise.
- A third approach would be to form the query in the actual visualisation using an approach similar to that used in the VR-VIBE project (Benford, 1995). The search queries exist as objects in the system and are mapped to virtual representations that are positioned in the virtual space. The position of the document representations is based on their relevance to the queries in the space. The closer the document is to a query object the more relevant it is.

These are just three possible approaches to integrating search tools into the Minerva framework and it is clear that there are many other possibilities that could be tried.

7.9 New Navigation Tools

The open framework of Minerva provides plenty of scope for developing new tools that can exploit the visualisation of hypermedia information. These tools would draw on the added benefits of a 3D environment to provide even more navigational clues to the user. The two sections below describe proposed navigation tools that could be implemented within the Minerva framework described previously. Although as yet unimplemented, it is hoped that the variety and expressiveness of the tools described help illustrate the openness of the framework and the potential it gives for novel and interesting visualisation approaches.

7.9.1 The Graffiti Tool

In the real world users often customise objects, improving their ability to recognise them. An example of this might be the customisation of the appearance of ring binders. If a person uses a number of ring binders they may find they need to distinguish between them. Although they can be purchased in different colours, this is not always enough to tell them apart. In some cases the customisation may be simply writing a title on the front of the binder or perhaps adding a number coding system to the spine. In other cases it might involve adding colourful stickers to the front of the ring binders which need bear no relation to the content, but simply help to distinguish it from the others. An important element here is that the visual clues are often a personal thing and may mean nothing to anyone else.

The virtual representations of the documents within Minerva can have textures mapped on to them. These textures might be used to display properties of the object, for example a wood texture might be used for text documents, an image used for image documents etc. An alternative however would be to allow the user to customise the texture for each document. The texture need only be a small bitmap, which the user can draw on using a standard paintbrush package. This tool would allow them to impose their own look and feel on an object, and provide a visual clue that helps them to distinguish the document from those around it.

An important part of the interface should be easily adding the graffiti. This could be achieved by allowing the user to click on an object and select graffiti. A small bitmap editor appears which lets them draw using a simple brush and a few colours. When they close the window, the texture on the side of the object immediately changes to the bitmap just created. The next time graffiti is added to the object, the current texture is used as a starting point. If the user were forced to create all the graffiti off line and then paste them into place, associating documents to bitmaps, the process becomes an authoring task rather than a spontaneous statement about the document. A user should be able to read a document, decide it is not particularly relevant to their work, select the graffiti tool and draw a big red cross on it. This should be as simple and easy to use as possible.

7.9.2 The Egocentric Visualiser

Many visualisation systems treat the user as a voyeur of the visualisation, perhaps able to affect the world around them by interacting with the objects in it, but in other respects disassociated from the content of the visualisation. The egocentric visualiser takes a different approach and treats the user (camera) as the focus of the visualisation and the virtual space around them is affected by everything they do, be it movement within the space, or modification of their preferences.

It is a common aspect of work that it is often necessary to switch tasks. This may be to a completely unrelated task, or possibly just a switch of contexts within the current piece of work. For example, while writing this document I often found myself switching between chapters as new ideas struck me or I reached a temporary impasse on a particular section. It is quite possible that the same general resources might be used in a number of tasks, but depending on the task in hand the resources gain new contexts.

The VR-VIBE visualisation tool developed in Nottingham (Benford, 1995) allowed the user to generate a number of queries. These queries, when placed in a 3D environment, gave rise to a visualisation of the document set with their position relative to the queries illustrating their relevance. One of the problems encountered was that if a document was equally strongly related to two queries it would appear half way between them. However if a document were equally weakly related to the two queries it would also appear half way between them.

The solution they adopted was two use the brightness of the document to indicate how strongly related to the queries it was.

The queries might be considered to be contexts within which the user is working. To take the resources used in producing this document as an example, one context might be Visualisation and another context hypermedia navigation. Within the VR-VIBE environment the user performs a voyeuristic role, and their current location has no effect on the position of the documents. A slightly different model exhibits interesting properties.

If the queries in the visualisation are likened to contexts, then as the user switches contexts, this might be denoted by the user moving between contexts, i.e. queries in the world. In the VR-VIBE system the queries all have a fixed weighting which exerts a 'pull' on the documents. Once the queries have been placed in the visualisation the document locations are fixed. What happens though if the position of the document is not based solely on the relationship of the documents to the queries, but takes into account the relative position of the user to the queries or contexts?

For example, as the user moves around the visualisation they wish to work within the context of VRML. They approach the VRML context, which might be represented as a signpost within the virtual environment. As they come within the sphere of influence of the signpost the documents re-arrange themselves around the user, with the documents which are most closely related to VRML being closest to the user. Those with no relationship to VRML are in the distance. The user can then remain in that position while they work on the VRML section, with the references they are likely to need being metaphorically within easy reach.

If the user then needs to work on hypermedia navigation, they traverse the virtual space to the hypermedia navigation signpost. As they reach it the documents have again re-arranged themselves so that those most relevant are closest to the user. It should be noted that a document that refers to hypermedia navigation within VRML is likely to be as close to the user as it was while they worked in the context of VRML. This can be contrasted with the situation in VR-VIBE where highly related documents and loosely related documents can end up being positioned in the same location. In the case where the user wishes to work in the context of VRML and hypermedia navigation, the user can drag the VRML signpost over to the hypermedia navigation signpost. With both exerting an influence the documents would re-arrange themselves so that documents closely related to both contexts would be close to the user, those in the middle distance are likely to be related to one or the other, or possible vaguely related to both. Those in the far distance would not be related to either context.

In addition, further helper tools might be envisaged within the visualisation. If, for example, the user is interested in web documents but less interested in images they might carry with them a 'virtual magnet' which attracts text documents, and one which repels image documents. Because the positioning of the documents are relative to the user these magnet devices would be easy to implement, with web documents being closer than they would otherwise be and image documents being moved further away.

For this egocentric tool to work it will be necessary to ensure that documents have a unique, location independent, representation in the visualisation since the position of the documents will be highly transient. The visualisation could still incorporate information from navigation tools such as history tools or link representations. In practice any tools which rely on path information will be difficult to use, since the location of the documents is not constant.

It should be noted that the Egocentric Visualiser is, by its very nature, a tool for use by a single user. Where the document locations are query oriented rather than user oriented it is possible to integrate multiple users as was one of the advantages of the VR-VIBE system.

7.10 Conclusions

When designing the Minerva architecture, a clear separation between the virtual objects displayed in the visualisation and the underlying hypermedia objects was critical. By keeping this separation the architecture could be constructed in a modular fashion, with new tools and processes plugged into the system without the need to modify other modules.

Using lessons learnt from the SHEP framework, an implementation was created in the same modular vein, with clear interfaces to the central communication process providing the openness of the system. Existing hypermedia modules could then be adapted to feed information into the newly created framework. In this way, a number of the Microcosm navigational tools were adapted to use the framework and supply information to the central object store. By creating a number of flexible mapping libraries this information could be mapped to objects suitable for displaying in the simple Virtual Environment Interface created using a 3D modelling system. The addition of a module to control the interaction with the VEI provided further flexibility in the generation of visualisations.

With a clear framework established the possibilities for the creation of novel visualisation tools was covered in the context of both how they might be of use to the user and how they could be implemented within the open framework.

With an open framework having been created for constructing visualisations of open hypermedia information systems, the next logical step is to explore the openness of the system by creating some actual visualisations of real information.

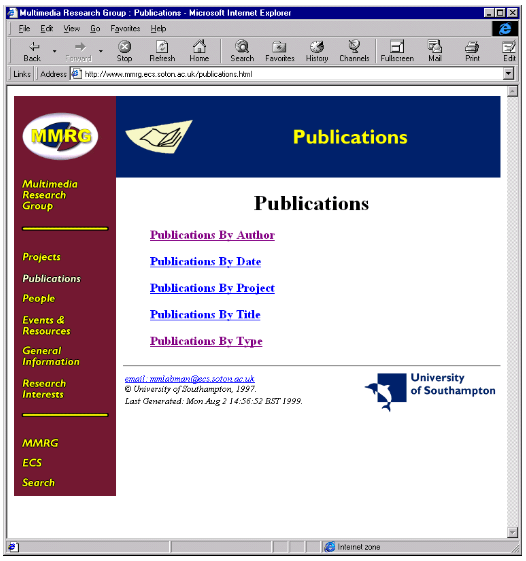
Chapter 8

Visualisation Using Minerva

In order to examine the flexibility of the Minerva system, three visualisations were created using it, each with their own specific problems and approaches. The visualisations are discussed in the three sections below. The first visualisation is of a publications database, containing electronic versions of publications along with meta-data about them. The second visualisation is of a small, tightly linked hypermedia application. The application contains a wide variety of documents and links. It also makes use of a number of hypermedia tools present in the Microcosm application. The final visualisation is of organisational information held by the Post Office Research Group. As well as documents holding information about people and projects, concept and group information is present along with links connecting concepts, groups, and documents in the system.

8.1 The MMRG Publication Database Visualisation

The Multimedia Research Group maintains its own archive of publications. There is a database entry for each publication that contains information about the authors, title, conference etc. Each publication also has its own web page which contains the abstract of the paper, all the information required to reference it, and links to copies of the paper in different electronic formats.


Multimedia Research Group : Publications - Microsoft Internet Explorer
http://www.mmrg.ecs.soton.ac.uk/publications.html
MMRG
Multimedia Research Group
Projects
Publications
People
Events & Resources
General Information
Research Interests
MMRG
ECS
Search

Publications

Publications

Publications By Author

Publications By Date

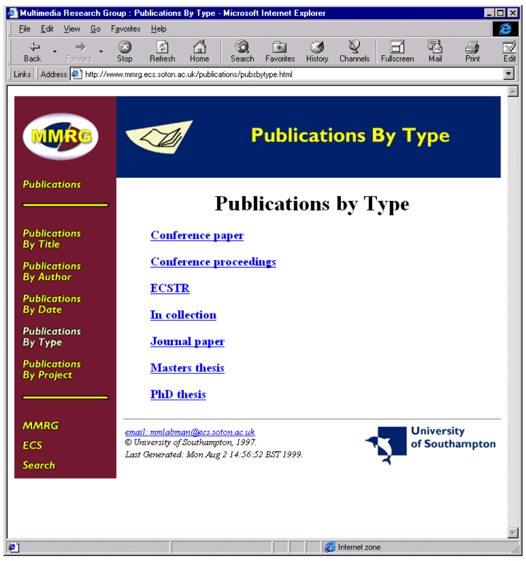
Publications By Project

Publications By Title

Publications By Type

email: mmlabman@ecs.soton.ac.uk
© University of Southampton, 1997.
Last Generated: Mon Aug 2 14:56:52 BST 1999.

(a) The top page of the MMRG publications.


Multimedia Research Group : Publications By Type - Microsoft Internet Explorer
http://www.mmrg.ecs.soton.ac.uk/publications/pubsbytype.html
MMRG
Publications
Publications By Title
Publications By Author
Publications By Date
Publications By Type
Publications By Project
MMRG
ECS
Search

Publications By Type

Publications by Type

Conference paper

Conference proceedings

ECSTR

In collection

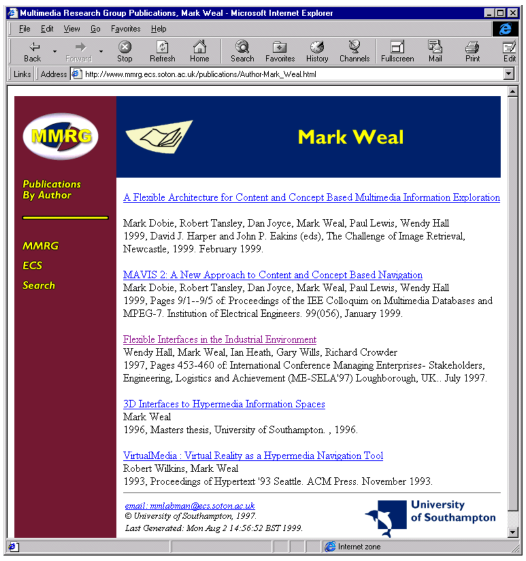
Journal paper

Masters thesis

PhD thesis

email: mmlabman@ecs.soton.ac.uk
© University of Southampton, 1997.
Last Generated: Mon Aug 2 14:56:52 BST 1999.

(b) MMRG publication categories by type.


Multimedia Research Group Publications, Mark Weal - Microsoft Internet Explorer
http://www.mmrg.ecs.soton.ac.uk/publications/Author-Mark_Weal.html
MMRG
Publications By Author
MMRG
ECS
Search

Mark Weal

A Flexible Architecture for Content and Concept Based Multimedia Information Exploration

Mark Dobie, Robert Tansley, Dan Joyce, Mark Weal, Paul Lewis, Wendy Hall
1999, David J. Harper and John P. Eakins (eds), The Challenge of Image Retrieval, Newcastle, 1999. February 1999.

MAVIS 2: A New Approach to Content and Concept Based Navigation
Mark Dobie, Robert Tansley, Dan Joyce, Mark Weal, Paul Lewis, Wendy Hall
1999, Pages 9/1--9/5 of: Proceedings of the IEE Colloquim on Multimedia Databases and MPEG-7. Institution of Electrical Engineers. 99(056), January 1999.

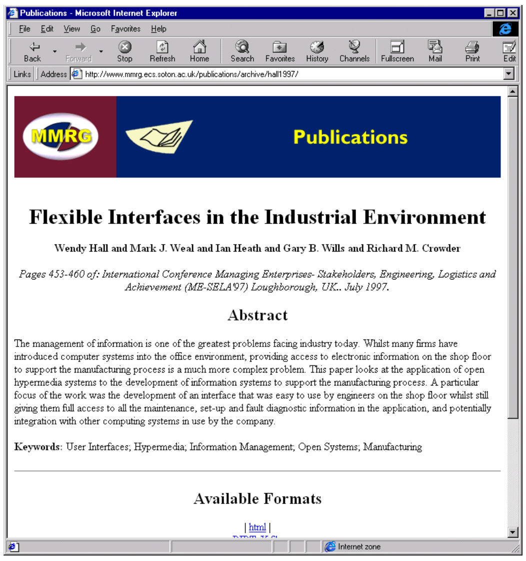
Flexible Interfaces in the Industrial Environment
Wendy Hall, Mark Weal, Ian Heath, Gary Wills, Richard Crowder
1997, Pages 453-460 of: International Conference Managing Enterprises- Stakeholders, Engineering, Logistics and Achievement (ME-SELA'97) Loughborough, UK.. July 1997.

3D Interfaces to Hypermedia Information Spaces
Mark Weal
1996, Masters thesis, University of Southampton. , 1996.

VirtualMedia : Virtual Reality as a Hypermedia Navigation Tool
Robert Wilkins, Mark Weal
1993, Proceedings of Hypertext '93 Seattle. ACM Press. November 1993.

email: mmlabman@ecs.soton.ac.uk
© University of Southampton, 1997.
Last Generated: Mon Aug 2 14:56:52 BST 1999.

(c) The publications listing of a single author.


Publications - Microsoft Internet Explorer
http://www.mmrg.ecs.soton.ac.uk/publications/archive/hall1997/
MMRG

Publications

Flexible Interfaces in the Industrial Environment

Wendy Hall and Mark J. Weal and Ian Heath and Gary B. Wills and Richard M. Crowder

Pages 453-460 of: International Conference Managing Enterprises- Stakeholders, Engineering, Logistics and Achievement (ME-SELA'97) Loughborough, UK.. July 1997.

Abstract

The management of information is one of the greatest problems facing industry today. Whilst many firms have introduced computer systems into the office environment, providing access to electronic information on the shop floor to support the manufacturing process is a much more complex problem. This paper looks at the application of open hypermedia systems to the development of information systems to support the manufacturing process. A particular focus of the work was the development of an interface that was easy to use by engineers on the shop floor whilst still giving them full access to all the maintenance, set-up and fault diagnostic information in the application, and potentially integration with other computing systems in use by the company.

Keywords: User Interfaces; Hypermedia; Information Management; Open Systems; Manufacturing

Available Formats

| html |

(d) A publication reference in the MMRG database.

Figure 8.1: The current Web interface to the MMRG publications database.

8.1.1 The Current Interface

The papers are currently accessed using a standard web interface. All of the web pages are pre-generated from a database containing Bibtex references for the papers and links to the electronic versions of the actual documents. Papers can be accessed by topic, by author, by title, by year or by type. With the exception of 'by title', which provides a complete alphabetically sorted list of papers, each of the categories leads to a sub page listing the possible sub categories.

Figure 8.1 shows four screen shots of the current interface to the MMRG publications database. Figure 8.1(a) shows the top level of the hierarchy of web pages. Figure 8.1(b) is the page presented to the user when they request a listing by publication type. It should be noted that there is no flexibility here for the reader to view all publications that are conference papers or journal papers. The categories are exclusive. Figure 8.1(c) shows the publications listing of a single selected author. It is only once the reader traverses to this level of the web site that they are presented with details of the actual publications. Finally, Figure 8.1(d) shows the abstract and details of a publication selected from the presented list.

Users can browse the various levels of the hierarchy to find papers they are interested in. If suitable formats exist, the papers can be viewed by the reader on-line in a web browser. If not however, the reader may still be able to download the paper in an alternative format for stand alone viewing.

8.1.2 Construction of the Visualisation

The sections below describe the steps that were taken to create a visualisation of the publications database using the Minerva framework.

The Object Handler

The Publications data is stored in a database that contains a record for each publication. Each record is a Bibtex entry with fields containing the information about the publication and a URL field that points to the HTML version of the document on the web server. The database program was modified to register with Minerva as an object handler. Each Bibtex entry is converted to a tagged message and added to the object store as an object. An example tagged message is given below.

> \title Flexible Interfaces in the Industrial Environment \month jul \email mjw@ecs.soton.ac.uk \Reference hall1997 \booktitle International Conference Managing Enterprises–Stakeholders, Engineering, Logistics and Achievement (ME–SELA'97) Loughborough, UK. \pages 453–460 \year 1997 \url projects/firm/SHEP.html \author wh, mjw,

ih, gbw@soton, rmc1@soton \project FIRM \keywords User Interfaces; Hypermedia; Information Management; Open Systems; Manufacturing \EntryType inproceedings

When the database receives a message from Minerva specifying that a particular object has been selected it launches the relevant URL in a web browser. If the database program receives a message from Minerva where the user has asked for information on a publication, it displays the relevant database record in a dialogue box. This is all the functionality required in the back end database.

The Mapping Libraries

Three different mapping libraries were used to generate the visualisation. The part played by each is described in the sections below.

Copytags: The interface module requires a description tag to display when the user moves over an object in the interface. Since the publications in the database have no description tag, the copytags mapping module is used to copy the reference tag of the object into a description tag in the virtual object. The title tag could just as easily have been used, indeed individual users can choose which they prefer.

Maptypes: To help differentiate between different types of publication, the maptypes module is used to colour the different virtual objects according to the type of publication being mapped. Table 8.1 details the mapping from the publication type of the object in the database to the colour of the virtual object used to represent it in the interface.

Publication	*Object Colour*
Conference Paper	Red
Journal Article	Green
Book	Blue
Ph.D. Thesis	Yellow
In Collection	Cyan
Misc.	Purple

Table 8.1: Mapping from publication type to virtual object colour.

Postags: The postags mapping library was used to position the objects within the interface. The reference tag for the publication was used to generate the x and y position for the object in the circular pattern described in section

7.6.5 . The year of the paper is used to generate the z position of the virtual object. The more recent the publication, the closer it is to the viewer in the visualisation.

The Interaction Handler

The interaction handler is scripted to allow the viewer to move through the 'time tunnel' in a straight line, and to rotate the tunnel of documents around its central access. Users can also double click on an object to view the web page associated with it. The double click action sends a SELECT.ACTOR message to the object handler of the object, informing it that the object has been selected. It can then launch the document in a suitable viewer. Right clicking on the object sends a QUERY.ACTOR message, asking for information on the object. Other forms of interaction are switched off for this fairly simple visualisation. By restricting the users motion in the visualisation, the user has less to learn in order to move around the visualisation.

The Control Process

Because the number of bibliographic references in the database is relatively small there is no need to restrict the number displayed in the visualisation. For this reason, this visualisation has no control process attached to it and by default, all of the objects in the store are placed in the visualisation.

8.1.3 The Minerva Visualisation

Figure 8.2 Shows a screenshot of the interface presented to the user by Minerva. The virtual documents can be seen as coloured cubes arranged in the tunnel pattern, with the older publications appearing further 'down the tunnel'. When the user moves the cursor over an object, the publication reference is displayed in a small window displayed at the cursor position. The colour of the object provides an indication of the type of the publication.

When the user double clicks on an object in the visualisation the relevant web page is presented to them in a standard web browser as shown in Figure 8.1(d). Right clicking on the object causes the database to display the information about the publication contained in the database.

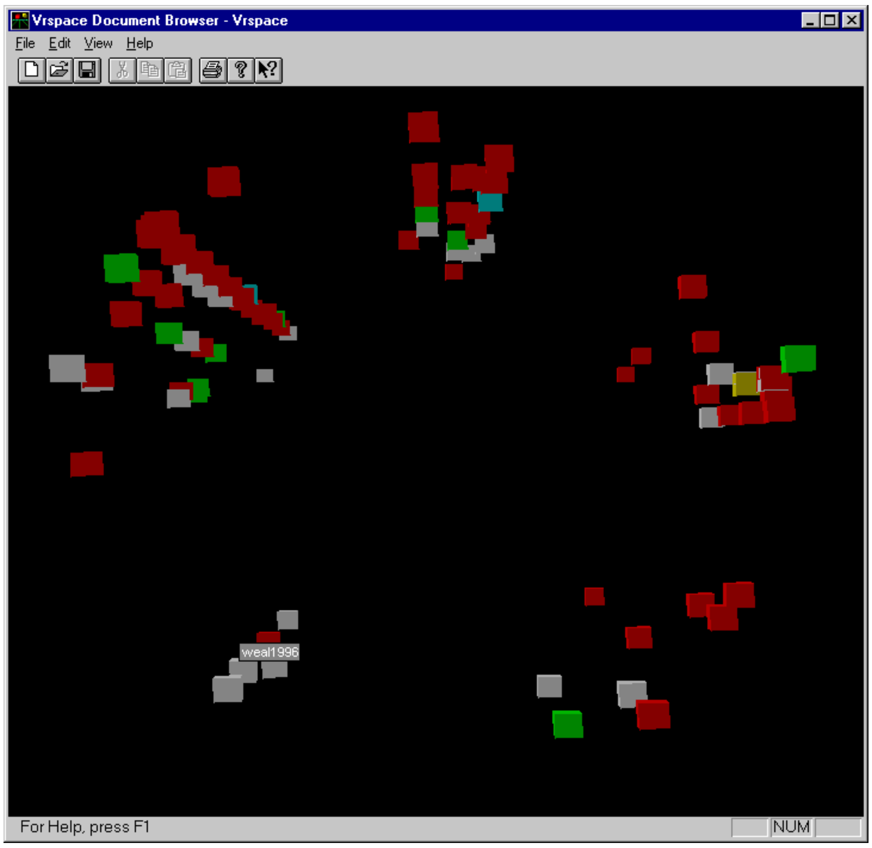


Figure 8.2: The time tunnel visualisation of the MMRG publications database.

8.1.4 Advantages and Disadvantages

A number of advantages and disadvantages can be identified with the Minerva visualisation of the publications archive.

Some advantages:-

- If the user of the visualisation has multiple publications archives they wish to view together, they can be combined in the visualisation by simply opening both databases. The publications for both databases would appear as virtual objects mapped using the same mapping modules.
- By mapping different fields to virtual object attributes, the user can see at a glance information that would not be as readily apparent when viewed using the web interface.
- It is possible to identify patterns in the visualisation that would not be as apparent or easily seen using the conventional web interface. For example, If an author has been quite prolific in their publishing over a number of years, a clear line of virtual objects will be observed within the visualisation.

and some disadvantages:-

- Although all the publications in the archive are visible to the user, only the object that the cursor is currently over will have its reference displayed. This can make it harder for the user to quickly find a particular document that might be more easily picked off of an alphabetically sorted list.
- The viewing of the publications takes place in a separate window to the visualisation of the database. In the web interface, a single window is used for browsing the database and reading the publications, which some users will prefer.

8.2 The Caerdroia Hypermedia Application Visualisation

Caerdroia is the name of the hypermedia application that comes as an example application with the Microcosm hypermedia system. It was designed as a tutorial which demonstrates the various features of the hypermedia system, and as such it provides a broad hypermedia application upon which to construct a visualisation.

The Caerdroia application concerns the history of mazes and the information contained within it ranges from descriptions and pictures of mazes to details of how they can be constructed.

The application contains 165 documents of various media types. These types include text files in various formats, i.e. RTF, HTML, DOC, as well as images, video and sound files. Each document is represented in the Microcosm Docuverse by a record containing information about the document. The information includes the name of the author, a description, information on the filename and type, and a number of keywords used to describe the contents of the document.

The application also includes a hypermedia linkbase containing close to 400 links. These links cover the full range of possible Microcosm links, including specific links from point to point, connecting selections within one document to selections in a different document. At the other end of the spectrum are included generic links, which link a particular selection, wherever it is found, to a destination document. The generic links are typically used in this application to link to glossary type information.

8.2.1 The Current Interface

The current interface to the Caerdroia application is the standard Microcosm interface described in detail in Chapter 3.

The user can browse the documents contained in the application using the Select a Document dialogue box as shown in Figure 8.3. This presents the documents in a hierarchical structure created by the author of the application. The right panel of the dialogue presents an alphabetically ordered list of all of the documents contained within the branch selected in the left hand panel. A document may appear in more than one branch of the tree. By double clicking on a listed document, the user can view the document in an appropriate viewer.

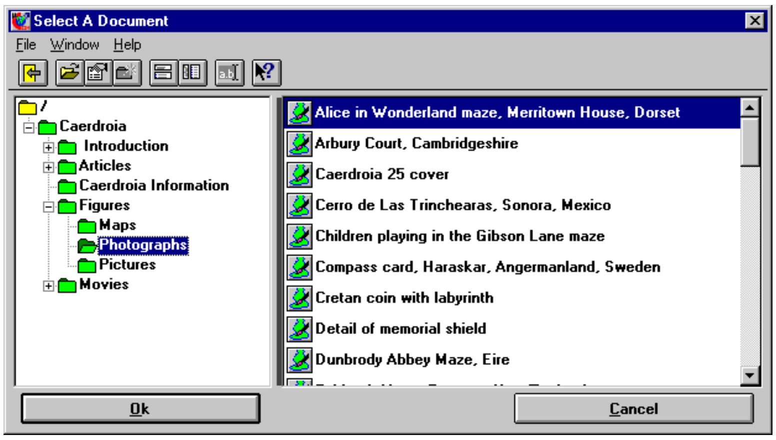
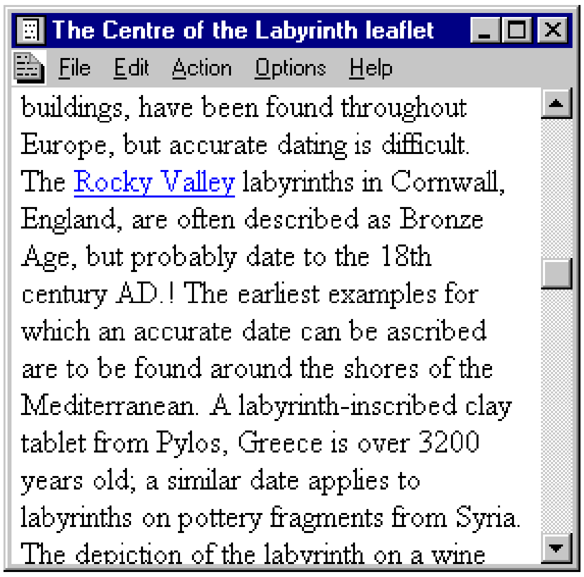

Select A Document
File Window Help
/
Caerdroia
Introduction
Articles
Caerdroia Information
Figures
Maps
Photographs
Pictures
Movies
Alice in Wonderland maze, Merritown House, Dorset
Arbury Court, Cambridgeshire
Caerdroia 25 cover
Cerro de Las Trinchearas, Sonora, Mexico
Children playing in the Gibson Lane maze
Compass card, Haraskar, Angermanland, Sweden
Cretan coin with labyrinth
Detail of memorial shield
Dunbrody Abbey Maze, Eire
Ok
Cancel

Figure 8.3: The document browser used to view the Caerdroia documents.

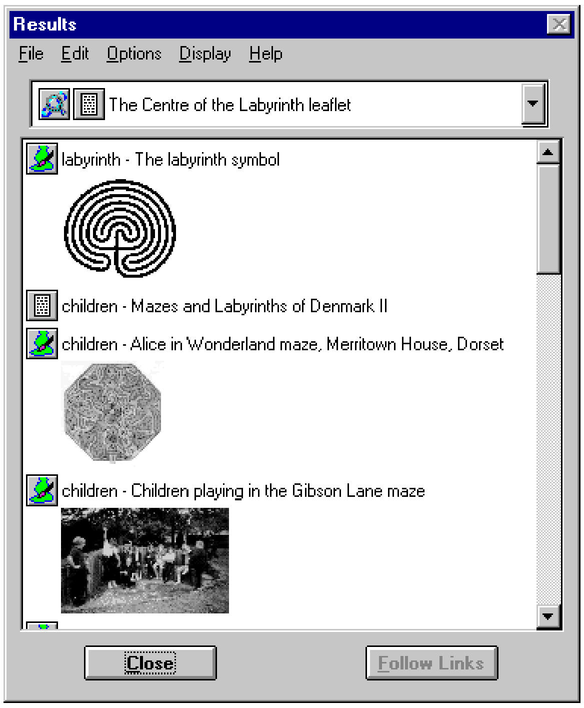
Button links are presented in the viewer in the form of a highlighted area, either a coloured box on an image, or a coloured word or phrase in a text document. This can be seen in Figure 8.4. In addition to the button links, users can also select pieces of text and ask the system whether any links exist from the chosen selection. The system will present the user with any generic links it finds on the selection.

By default, if a single link is returned to the user it will be followed automatically and the destination of the link presented to the user in a document viewer. Where a link following action results in more than one possible link, the user is presented with a dialogue giving them the choice of which link to follow. This dialogue is illustrated in Figure 8.5.


The Centre of the Labyrinth leaflet
File Edit Action Options Help

buildings, have been found throughout Europe, but accurate dating is difficult. The Rocky Valley labyrinths in Cornwall, England, are often described as Bronze Age, but probably date to the 18th century AD.! The earliest examples for which an accurate date can be ascribed are to be found around the shores of the Mediterranean. A labyrinth-inscribed clay tablet from Pylos, Greece is over 3200 years old; a similar date applies to labyrinths on pottery fragments from Syria. The depiction of the labyrinth on a wine

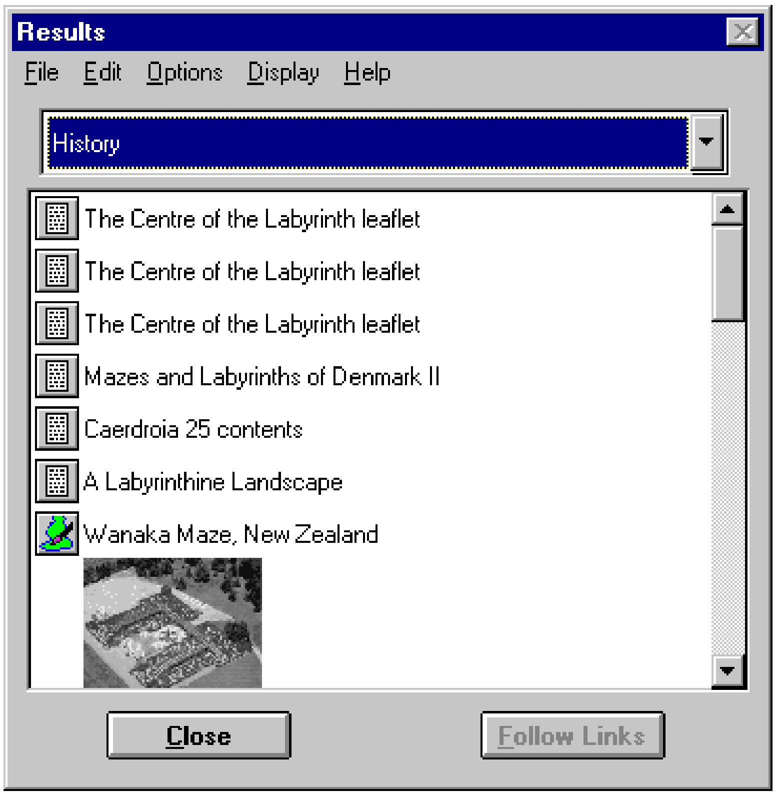
Figure 8.4: The text document containing a button link.


Results
File Edit Options Display Help
The Centre of the Labyrinth leaflet
Close
Follow Links

labyrinth - The labyrinth symbol
children - Mazes and Labyrinths of Denmark II
children - Alice in Wonderland maze, Merritown House, Dorset
children - Children playing in the Gibson Lane maze

Figure 8.5: The results viewer presenting a choice of links to the user.

The user can also find out what documents they have already viewed by calling up the history tool. It presents the user with a chronologically ordered list of what documents they have already viewed as shown in Figure 8.6. They can click on documents in the list to view them again.



The Centre of the Labyrinth leaflet
The Centre of the Labyrinth leaflet
The Centre of the Labyrinth leaflet
Mazes and Labyrinths of Denmark II
Caerdroia 25 contents
A Labyrinthine Landscape
Wanaka Maze, New Zealand

Figure 8.6: The history viewer presenting the user with a list of previously viewed documents.

8.2.2 Construction of the Visualisation

The Object Handlers

The Microcosm Document Management System (DMS) is registered as a manager, and adds all of the documents in the Docuverse into the object store as actors. Each document is already in the requisite tagged message format. An example is given below.

> \FileName $ Caerdroia_Path1$ \\Data\\images\\c92-15b.bmp \@LogType \\MultiItem0 /Caerdroia/Figures/Photographs \\MultiNumItems 1 \Type BITMAP \ImportDate 03/21/94 \UniqueID 100.03.21.94.14.52.51 \Description Wanaka Maze, New Zealand \@Author \\MultiItem0 Stuart Landsborough \\MultiNumItems 1 \@Keyword \\MultiItem0 hedge maze \\MultiItem1 puzzle \\MultiItem2 modern \\MultiNumItems 3

The Microcosm linkbase also registers as an agent, and each link in its database is submitted to the object store as an actor. An example link message is given below, illustrating the type of information that can be expected to be in the object.

> \SourceFile 100.03.22.94.17.07.31.4839112 \SourceSelection Scandinavia \SourceOffset 689 \SourceDocType TEXT \SourceLogType \\MultiItem0 /Caerdroia/ Introduction \\MultiNumItems 1 \Action CREATE.LINK \DestFile 100.03.22.94.17.19.15.5543415 \DestSelection \DestOffset 0 \DestDocType TEXT \DestLogType \\MultiItem0 /Caerdroia/Articles \\MultiNumItems 1 \Description A New Volundarhus in Jutland \Link Generic \RealSourceSelection Scandinavia

Documents and links form the two main types of object rendered in the visualisation.

The Information Assistant

The Microcosm history tool registers with the system as an assistant. As the user views documents, the history tool adds history information to the actor representing the document in the store. This history information, a simple history tag of value 1 if the document has been viewed, can then be mapped onto an aspect of the virtual documents representation.

The Mapping Libraries

Three mapping libraries are used to create the visualisation. The libraries, and the part they play in the mapping process, are described below.

Postags: The postags mapping library is used to position the virtual documents in the virtual space. The x and y position of the documents are based upon the description of the document. The documents are arranged in a rough circle based on the first few characters of their description, as described in Chapter 7. The z positioning is based upon the unique identifier of the document in the Microcosm system. Although this makes the positioning of the documents along the z-axis fairly arbitrary, it is consistent from session to session, so over time the user of the visualisation will be able to build a mental map of the documents in the virtual space.

Maptypes: The maptypes mapping module is used to provide a number of mappings to attributes of the virtual objects. Table 8.2 details the mapping of document type to virtual object colour. This allows the user to identify the type of the object visually from its representation.

Document Type	*Object Colour*
Text	Red
Image	Green
Video	Blue
Animation	Blue
Audio	Cyan
WWW	Purple

Table 8.2: Mapping from document type to virtual object colour.

The history value of the object is mapped to its opacity. The idea behind this being that if a document has not been viewed by the user it appears completely opaque. If it has already been viewed however it will appear partially transparent, reducing its impact within the visualisation and suggesting that it is probably less worth viewing as the information has been seen by the user already.
The maptypes is also used to map the Type of a link to its colour. Table 8.3 shows the mapping. This helps users to see at a glance the types of links.

Link Type	*Object Colour*
Specific Link	Cyan
Generic Link	Yellow

Table 8.3: Mapping from link type to virtual object colour.

Maplinks: The maplinks mapping module is applied to objects generated by the Microcosm linkbases. It has some understanding of the contents of a link record and uses this to generate a virtual representation of the link. On being asked to cast a link, it extracts the information about the source and destination document of the link. It then queries the object store to obtain the positions of the virtual representation of the documents. These positions are used in the virtual link representation, leading to a virtual object being created which joins the source and destination in the virtual space.

8.2.3 The Minerva Visualisation

Figure 8.7 shows the visualisation of the Caerdroia Microcosm application. The documents can be seen as coloured objects. Text documents appear as red cubes, image documents as green cylinders. By moving the cursor over the objects the user calls up the description of the document in a pop up window, as shown in Figure 8.8. The Links are represented by yellow lines connecting the various document objects. Where a document has a large number of links emanating from it, it is possibly an index or contents page.

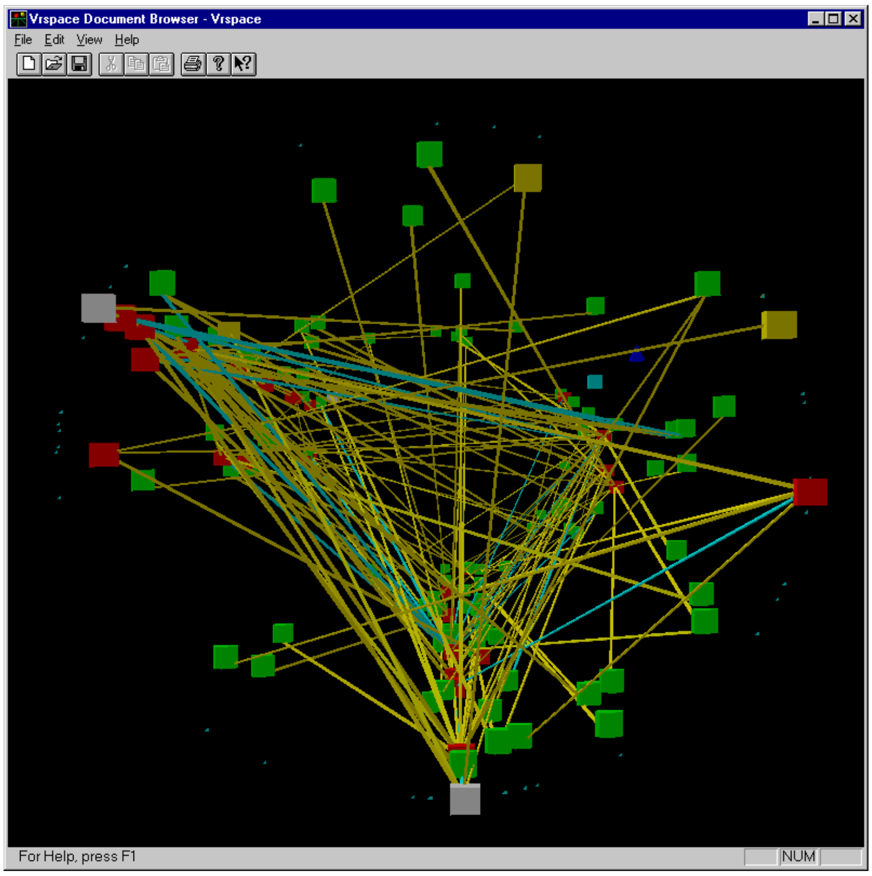


Figure 8.7: A view of the Caerdroia application with all the documents and links visible.

Figure 8.8 shows a more zoomed in view of the visualisation. The user can use the mouse to move around the visualisation and here, has moved into the area containing the links and documents. Only a subset of the documents are now visible, and links often disappear out of frame. By clicking on the links, the user can call up the destination even though the destination document object is not currently visible. By selecting a document object in the visualisation the user can call up information about the document by right clicking on the object, or view the document itself by double clicking on the object. The document will appear in an appropriate viewer. Once a document has been viewed, the object representing it will become partially transparent to indicate that it has already been seen.

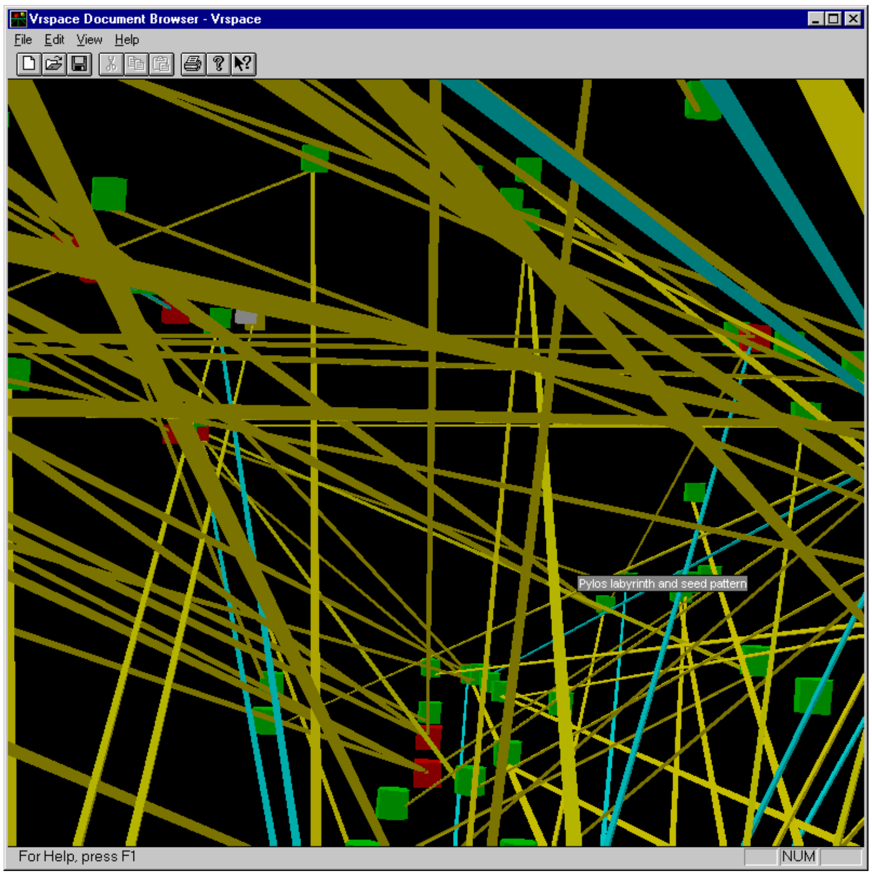

Vrspace Document Browser - Vrspace
File Edit View Help
Pylos labyrinth and seed pattern
For Help, press F1
NUM

Figure 8.8: Amidst the web of links, a document object is selected.

8.2.4 Advantages and Disadvantages

Although created purely as an example of the flexibility and openness of the framework, it is worth considering some of the advantages and disadvantages of the Minerva visualisation compared to the original interface to the hypermedia system.

First of all some advantages:-

- The user is in a position to view information about all the documents at the same time. Using the select a document dialogue, users can only view the documents in the currently selected branch. In the visualisation, the view is dependent upon the users position within the space.
- In the original interface, to find out what documents they have already seen the user has to open up a specialist dialogue box which presents them with the history list. In the Minerva visualisation, this information is presented

in the same interface as the document information. The dynamic representations of the documents help the user to build up a cognitive map of the space that reflects what they have previously viewed in the virtual document representations.

- Under the Microcosm system, the mechanisms for accessing information about a hypermedia object are different depending on the object. To find out about a document the user can call up a dialogue from the Select a Document interface, which gives the details of the object. To find out about a link the user must run the link editor and select the link they want information about. In the Minerva visualisation, both are achieved by right clicking on the virtual representations of the links and documents.
- By colour coding the links, the user can see at a glance which documents have which types of links going into and out of them. If a virtual document object is connected to a large number of generic links then it is quite likely that it is a glossary document. If, on the other hand, the document is the source of a large number of specific links then it is possible the document might be a contents or index page.
- By placing the history information into the visualisation as modifications to the virtual objects, the information is always presented in the same context. Under the Microcosm interface, the information appears in disparate dialogue boxes and the representation of the documents in these dialogue boxes is often different even though a Microcosm 'look and feel' has been attempted.

But there are some disadvantages:-

- Users can only view the description of the document when they move the cursor over the representation of the document. It would be possible to display the descriptions of all of the documents in pop-up windows, however this would quickly become confusing. The user must therefore rely on the representation of the object to remind them of the document.
- Some of the clustering created with the folders in the select a document is lost. In the case of Caerdroia this is not a big problem, as the folders were used to divide the documents up by type, i.e. maps in one folder,

photographs in another. Since the colour of a document represents its type in the visualisation, this information is preserved albeit in another form.

- The sheer number of links does give the visualisation a fairly cluttered look. If the generic links are not shown on the visualisation an improvement is achieved. Since the generic links have an abstract source it seems reasonable to restrict the visualisation to true point to point links.

8.3 The Post Office Research Group Visualisation

The Post Office Research Group (PORG) maintains a database containing information about the group and the activities that go on within it. The information is arranged in clusters that represent a variety of things from activities to individuals within the group. The clusters are connected by relationships. A number of different types of relationship exist within the database, for example:-

- Has research area of
- is connected to
- is interested in

Each cluster can also have associated with it a number of inputs and outputs. Inputs are generally pieces of text which form synonyms for the cluster name. Outputs are usually files that describe the cluster. In the case of clusters representing people, one of the outputs will be a simple CV of the person listing all the relevant contact information. Cluster representing research groups may have the reports pertaining to that group as outputs.

8.3.1 The Current Interface

An application containing the information in the database has been created using the Microcosm system. The documents were placed in the Document Management System and are viewed using the standard Microcosm viewers. The cluster and relationship information is held in a back end database connected to Microcosm using a specialised filter process acting as an interface between the Microcosm system and the external database application.

A number of bitmap images were created which display bubble diagrams showing the clusters and the relationships between them. A portion of one of these images is shown in Figure 8.9 below. The diagrams were created manually rather than automatically, so that the position of the bubbles has been arranged to provide neat layouts of the clusters represented in the diagram. As the overall organisation chart has a large number of clusters, each bubble diagram represents only a subset of the organisation, with any given cluster possibly appearing on more than one bubble diagram.

Each of the bubbles in the diagram has a link over it. When the user of the system clicks on the link the back end database is fired up and all of the relevant documents and related clusters to the cluster clicked on are presented to the user in a cluster browser. This is shown in Figure 8.10.

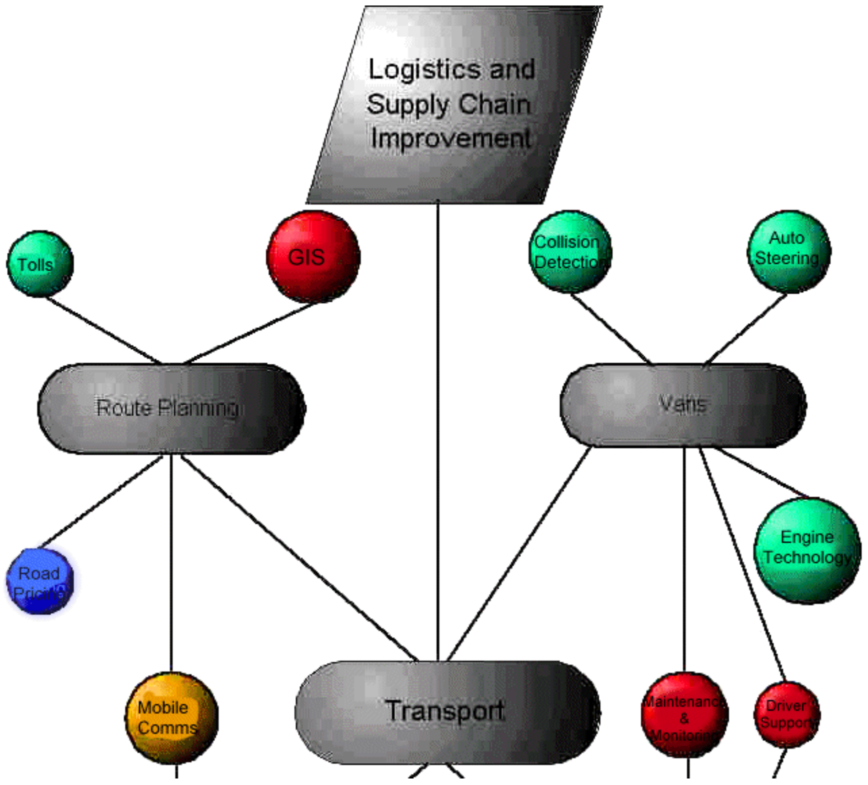
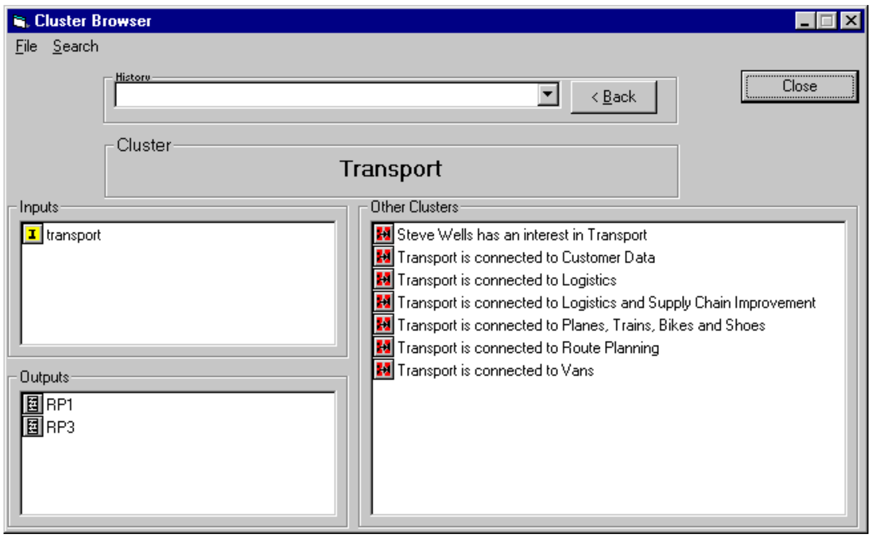


Figure 8.9: An example bubble diagram showing cluster connections.

The cluster diagrams are created by hand. The size, shape and colour of the bubbles in the diagram may have originally been intended to provide additional information, but to all intents and purposes they simply serve to differentiate between the different bubbles. Eight separate diagrams show the relationships between the various clusters. Some clusters will appear on more than one diagram and how the diagrams interconnect is not always clear.



Transport

Inputs

transport

Outputs

RP1
RP3

Other Clusters

Steve Wells has an interest in Transport
Transport is connected to Customer Data
Transport is connected to Logistics
Transport is connected to Logistics and Supply Chain Improvement
Transport is connected to Planes, Trains, Bikes and Shoes
Transport is connected to Route Planning
Transport is connected to Vans

Figure 8.10: The interface to viewing information about the clusters.

One of the problems of using hand created diagrams is that if new clusters are added to the database the diagrams will have to be altered manually. Also, each diagram represents a particular view of the database. The diagrams represent clusters relating to certain topics such as maintenance, communication, logistics etc. If a user of the database has their own personal requirements, these might not be covered by the pre prepared selection of diagrams available.

8.3.2 Construction of the Visualisation

The Object Handlers

Two separate databases of information are present in the Post Office Research Group application. The Microcosm Document Management System holds all of the documents in the application, and registers with Minerva as an object handler for the documents. They are all added into the object store when the application starts up. A typical Microcosm document record in the application might look like :-

> \@LogType \\MultiItem0 /PORG/People \\MultiNumItems 1 \FileName $PORG_{P}ath0$\\Data \\People \\JBallant.rtf \Type TEXT \ImportDate 14 Oct 1998 13:17:15 GMT \UniqueID 793.15.17.13.14.10.98 \Description Ballantyne, Jim

Cluster information and inter-cluster linking information is stored in a separate database. The database also registers as an object handler and submits each cluster as an object in the object store. The cluster information is also added to the object store. An example cluster object is given below.

> \Name BallantyneJim \Type Cluster \Description Jim Ballantyne \@connect \\MultiItem0 vissim,relpeopleint \\MultiNumItems 1 \@input \\MultiItem0 ballantyne \\MultiNumItems 1 \@output \\MultiItem0 793.15.17.13.14.10.9 \\MultiNumItems 1

Also stored in the database are the relationship objects that link the clusters together. Below is an example relationship object.

> \SourceFile BallantyneJim \Type Relation \Name BallantyneJim-connect1 \Description BallantyneJim-connect1 \Action CREATE.LINK \DestFile vissim \RelType relpeopleint

Each cluster also has a number of inputs and outputs associated with it. The inputs refer to a selection that relates to the cluster, the equivalent of a generic link. Outputs refer to files within the Microcosm Document Management System. Examples of each are given below.

> \DestFile BallantyneJim \Type Input \Name BallantyneJim-input1 \Description BallantyneJim-input1 \Input ballantyne

> \SourceFile BallantyneJim \Type Output \Name BallantyneJim-output1 \Description BallantyneJim-output1 \Action CREATE.LINK \DestFile 793.15.17.13.14.10.98.100

If the user selects a document object in the visualisation, Microcosm will launch the document in the appropriate viewer, or display information about the document, depending on the nature of the selection. If the user selects a cluster in the visualisation then the cluster database will send a message to Microcosm asking it to find links with the cluster name passed as the selection. The resultant links will be displayed in the results dialogue. An example link is shown below.

\SourceFile 768.50.52.11.28.10.98.100 \SourceDocType BITMAP \Action CREATE.LINK \ButtonAction FOLLOW.LINK \TEXT Logistics and Supply Chain Improvement \Description Logistics and Supply Chain Improvement \UniqueDocID 2 \RealSourceSelection 274,10,401,137 \SourceSelection 274,10,401,137

The Information Assistants

The Microcosm history tool acts as an assistant for the documents in the system. When the user views a document, its information is updated in the object store by the history tool to reflect this fact.

The Mapping Libraries

A number of different mapping libraries are used in the visualisation, as described in the sections below.

Maptypes: The maptypes library is used to generate a colour for the virtual objects in the scene based on the objects type. This helps differentiate the links from relationships and the clusters from the underlying documents. Table 8.4 lists the mappings used in the visualisation.

Object Type	*Object Colour*
Text Document	Red
Image Document	Green
Cluster	Blue
Relation	Yellow
Output Link	Cyan

Table 8.4: Mapping from object type to virtual object colour.

The maptypes library was also used to map the history information to the opacity of the virtual objects in the case of documents. If a document has been viewed by the user and so has a history value of 1, then the virtual object is displayed as partially opaque indicating to the user that they have already viewed the information.

Postags2: Rather than place the documents in a circular pattern as was the case with the previous postags object mapping library, a planar approach was adopted. The documents are placed in the x, z plane in a grid pattern based upon their title and UniqueID fields. The title places them along the x-axis, the UniqueID is used to generate a z coordinate. Although the UniqueID has no inherent meaning, it is used to provide a reasonably spread out grid of documents.

Placetags: The placetags library was used to position the clusters within the visualisation. The cluster information contained an indication of the level of the cluster in the organisational hierarchy. This information was used to generate the y coordinate for the virtual object. The x and z coordinates were generated from the description and name tags of the cluster, with the clusters being arranged alphabetically along the x-axis based on their description and the name being used to generate a separation in the z-axis. The distance spanned from A-Z by the clusters was less for the higher levels, inducing a pyramidal effect so that the higher the level, the less clusters, and the closer together they appeared. The size of the cluster was used to reflect the number of inputs, outputs and relations that a cluster had. Therefore the more connection to a cluster the larger it became. The placetags mapping library uses specific knowledge of the cluster object to generate a size for the object.

Maplinks: The map link library is used to generate the virtual objects for both the Microcosm links, and also the cluster relations and outputs. In generating the relation and output objects for the object store, the fields SourceFile and DestFile were used to make it compatible with the maplinks library. If this had not been possible, the copytags library could have been used to convert the equivalent tags to the fields required by maplinks. The maplinks library operates by generating a virtual link object between the positions of the two objects it is given. The fact that it is generic to any named objects allows it to be used for the different types of links.

8.3.3 The Minerva Visualisation

Figure 8.11 shows a screenshot of the Post Office Research Group visualisation. The view is zoomed back showing all the clusters and documents in the visualisation.

In Figure 8.12 the user has moved in closer, and some of the clusters are no longer visible. The user is in the process of selecting an individual document with the cursor.

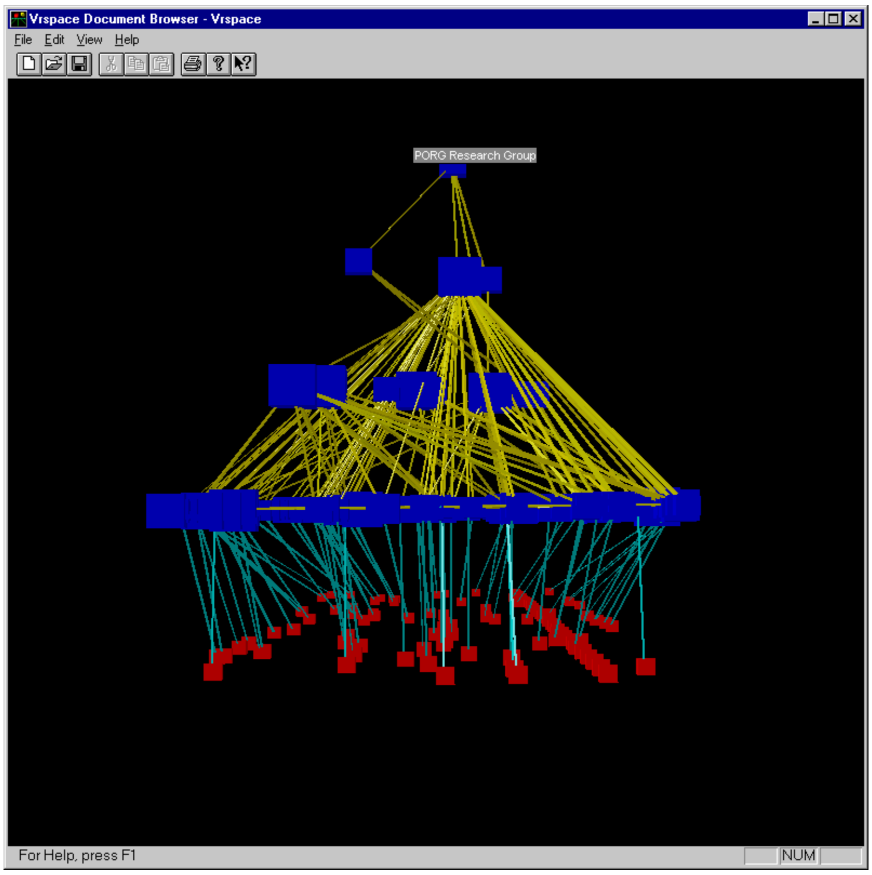


Figure 8.11: The Post Office Research Group visualisation.

The user can use the mouse to move around the visualisation to view the clusters and documents from different positions. Double clicking on a cluster causes the cluster information to be displayed as shown previously in figure 8.10. Double clicking on a document will result in the document being launched in the appropriate viewer. Right clicking on a document will display the document information as was the case in the Caerdroia visualisation. Right clicking on a cluster will lead to the cluster information being displayed to the user. The user can also click on the relation links and the output links. In both cases this will produce the same result as if the destination of the link had been selected.

8.3.4 Advantages and Disadvantages

Some advantages of the Minerva visualisation:-

- Using the original interface, the user was only ever able to see a subset of the overall organisational structure. Each bubble diagram only showed some of the clusters and their connections. Even though a cluster may appear

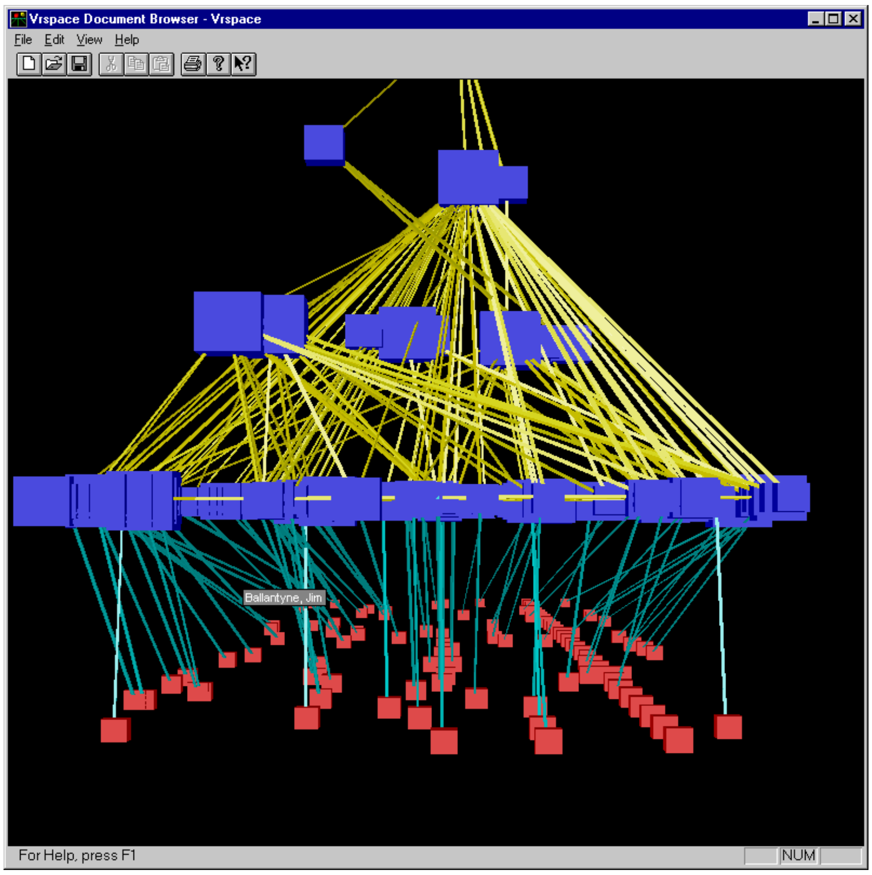


Figure 8.12: The user zooms in and selects a document.

on more than one diagram, the connections between the diagrams is not always apparent. With the Minerva visualisation, the whole organisation is available.

- In the original Microcosm application, users interacted with clusters and relation information in a special cluster interface dialogue, and the documents using the Microcosm system. The Minerva visualisation allows the user to access the clusters and documents through a common interface.
- If the cluster database were to be modified to add a new cluster, or change the relationships within it, the original bubble diagrams would need to be edited by hand to reflect the changes and possibly modify the layout to clarify the new structure. Because the Minerva visualisation is generated automatically, the layout would alter itself to reflect the changes, without the need for intervention from the application designer.

Some disadvantages of the visualisation:-

- The bubble diagrams provide the user with nicely laid out subsets of the information which have been arranged manually to give more clarity than can be achieved with the simple automatic layout strategy adopted with the example visualisation. Unfortunately this layout information is not encapsulated in the data but resides solely in the diagrams themselves. An alternative layout strategy for the visualisation could have involved manually positioning the clusters by specifying their position in the 3D space in advance rather than automatically.

- The small amount of information that exists in the database about each cluster and relationship results in the virtual representations all looking fairly similar. The size of the clusters can be varied but essentially they are all represented as blue cubes. The variety of bubble appearances in the bubble diagrams was generated by hand and although holding no intrinsic meaning, does serve to differentiate the clusters in the diagram. Perhaps a random element to the look of the clusters would improve the visualisation by providing cues to help users identify individual clusters more easily.

8.4 A Variety of Layout Techniques

The layout techniques used to position objects in the three visualisations above are all reasonably simplistic, relying on simple alphabetical ordering, or sometimes just attaining a separation through use of arbitrary data. One of the key problems encountered when generating the position of objects was the lack of useful information to base it on.

At best, the layout can be said to be fixed from session to session so that once a user learns where an object is, they will be able to locate it on subsequent occasions. By using the year information to generate the z coordinate in the publications visualisation, depth cueing is used, suggesting to the user that more recent publications are closer and therefore more relevant.

In the case of the Post Office Research Group visualisation, the bubble diagrams of the original interface were produced by hand, and the relative positioning information was decided arbitrarily and not encapsulated in the data in the database. It is possible to use the level information to create a hierarchy of clusters. Aside from this, only the description and name of the clusters provide much useful information.

Clearly more complicated layout strategies could be devised if the visualisation went beyond simple mappings of value to position. One approach would be to create a dynamic layout by using the relationships between clusters as a form of spring mechanism, causing clusters that are related to be drawn together as in the Hyperspace system (Wood *et al.*, 1995) described in section 6.5.17. By examining all of the possible attractions between clusters, a layout could be formed which minimised the 'tension' in the springs. This does have the disadvantage of the objects moving as clusters and relationships are added and removed from the system. It would also negate any hierarchical structure that might be divined from the information.

8.5 Automatic Visualisation from Scripts

There is clearly scope for automating the creation of some of the mapping tables used by the maptypes library. Mapping from an enumerated type, such as the document type, to an attribute such as colour should be fairly straightforward. This would remove the need for the mapping tables to be created by hand.

8.6 Representation of Links within a Visualisation

One of the pleasant surprises in producing the visualisations was the versatility of the maplinks library. Because it treats links as simple lines between a start and destination, the library proved to be general enough to map both Microcosm links and also relationships stored in the PORG database. The library only needs to know what tags to look in to find the name of the source and destination objects. It can then look up those objects in the object store, find their positions, and generate the link object based on those positions. The nature of the source and destination objects are irrelevant in this, providing a helpful abstraction in the process.

Perhaps the biggest problem that arose however was one of synchronisation. The position of the links is dependent on the position of the source and destination objects and therefore cannot be calculated until the source and destination have been mapped to their positions. This was just one of a number of synchronisation issues which will be covered in Chapter 9.

8.7 Conclusions

The three visualisations described in the sections above illustrate how the Minerva framework can be used to create visualisations of quite diverse sets of information.

The Bibtex information in the publications database lent itself well to the 'time tunnel' visualisation approach, where the user is likely to be more concerned with recent publications which will appear closer to them in the interface. By mapping colour to the type of publication, the user can see at a glance which are conference papers and which represent books for example.

The visualisation of the Caerdroia application combined two different sources of information, documents and links, into a single visualisation. Other navigational information such as that produced by the history tool could also be incorporated rather than displayed in a separate interface. Because the mappings of position are fixed from session to session, the user will gradually build up a cognitive map of the visualisation enabling them to find documents within the visualisation on subsequent visits.

The PORG visualisation illustrates how information from two separate applications, Microcosm links and documents and cluster information stored in its own database, can be combined within a single visualisation. The hierarchy of clusters was partially preserved and the user can readily see which clusters are related to each other.

The three visualisations described in this chapter serve as examplars for the power if the Minerva architecture. In themselves, they are nothing more than simple visualisations. Good arguments can be made that they provide no significant improvement over the original interfaces to the systems. In the case of the Caerdroia visualisation, the large number of links presented in the visualisation merely obscure the documents which the user of the system is trying to access. The reason behind the visualisations was not to create novel, powerful, visualisations, but to show how a number of different soures of information can be brought together in a single unifying visualisation whilst maintaining an abstraction between the underlying hypermedia information and the front end interface.

To produce the visualisations, minimal modifications needed to be made to the navigational tools. In the case of the Document Management System, publication database and cluster database, the modification involved simply outputting their information in a simple tagged format. Other tools, such as the history filter, added their information based on the documents unique identifier. If hypermedia systems are to call themselves open, then this level of access to the underlying information should be normal practice anyway.

The Virtual Environment Interface simply requested objects from the store and rendered them. In the examples above, the interface was a 3D world generated using RenderWare. It could just have easily been constructed using the DIVE system or VRML, since it required no specific knowledge of the database supplying the information. All it needed was the ability to ask for objects in the object store and to render them as objects in the interface. By publishing the format in which it wishes to receive its objects, the mapping libraries are able to supply the necessary information.

The mapping libraries provide the abstraction between the underlying information and the objects in the interface. They can range from simple scripted mappings from text string to text string or might be complex libraries providing very detailed mappings from the hypermedia information to the rendered objects. In the three examples given in this chapter, the mappings were all relatively simple. Benedikine mappings were used, with colour representing some aspect of the document, in the case of the publications database, the publication type. The documents were arranged around the inside of a virtual cylinder, in a similar manner to the WINONA circular wall visualisation. By contrast, the PORG clusters and documents were placed on planes according to their relative position in the hierachy, documents appearing on the lowest plane.

In order to keep the visualisations simple, document mapping was only dependant on the attributes of the documents themselves. Virtual attributes of documents were not based on their relevance to other documents, as in Lyberworld or Starwalker, nor were they based on the relevance to queries in the system, as was the case in VR-VIBE or Vinetta. There is no reason why this form of mapping could not be implemented within the Minerva framework however. The positioning of links within the visualisation is based upon the poition of the documents or clusters within the visualisations in that the link must join the source and destination anchors.

By keeping the architecture open, even the control mechanism for the user can be modified in a modular fashion. Where the visualisation requires it, the user can be restricted to only moving in certain directions. In the case of the publications visualisation they are restricted to moving up and down the centre of the cylinder of publications. Here, the user does not necessarily require six degrees of freedom in their movement and the simplified interface is designed to reduce the overhead of moving around the system. A similar approach was used in the perspective wall visualisation and the VIRGILIO system. Where the visualisation requires it, the control system can allow the user to move with more freedom, as was the case in the PORG visualisation.

The Minerva architecture is designed to allow flexible visualisations to be constructed which combine different forms of hypermedia information within a single unifying interface. By keeping all aspects of the architecture open, everything from the mapping systems to the control systems can be configured allowing a wide variety of visualisations to be constructed without the need to hard-wire the interface to the underlying hypermedia system.

The next chapter examines some of the problems with these simple visualisations and discusses where the Minerva framework might go in the future and how it can be improved to provide for the creation of more useful visualisations.

Chapter 9

Future Work

Both the SHEP architecture and the Minerva framework were designed and implemented with the aim of illustrating how an open approach to information visualisation and interfaces in general can have large benefits to interface designers. Once these systems have been created and proven to be effective, there is obviously much more work that could be carried out, extending the systems and going beyond proof of concept towards creating tools which will genuinely improve the way in which users can interact with hypermedia systems. Some ideas as to where to go from here are suggested below.

9.1 The SHEP Architecture

Since its initial implementation and testing, the SHEP architecture has been successfully incorporated into the Microcosm Pro system. It was subsequently used to help create an interface for use on wearable computers. A shepherd was created which integrated a voice activation process with the architecture. User speech input was translated into SHEP commands that were sent to the appropriate interface components. This allowed users to minimise and maximise windows, move them around the screen and alter aspects of their appearance using voice activation only. This ease of integration helps illustrate the power of an open approach to interface construction.

The modules constructed to prove the concept of the SHEP architecture were quite simplistic and designed as much to test the various aspects of the framework as to provide practical interface assistance to the user. With the framework now in place and in a position to be tested there is clearly scope for the design and implementation of more advanced shepherds to assist users in more complicated screen management tasks.

9.2 Minerva Data Objects

As it is currently implemented, the Minerva framework uses the Microcosm message format as a means of storing and passing information. The Microcosm tagged message format provides easy mechanisms for creating data objects and passing them around between the various modules of the framework. Although at the time this had the advantages of available tools and an easy to use format, a good argument could now be made for switching to using a more common language such as XML. Now firmly established, XML is portable and has the advantage that parsers are now available in a number of programming languages. Also, many database and hypermedia systems have their information already in XML format or easily exported to XML. By using XML, the work required to convert existing systems to use the Minerva framework would hopefully be reduced by using a commonly used representation of information.

9.3 Minerva as a Protocol

As well as establishing XML as the information carrying mechanism, the actual information passed around the system could be formalised more. Currently a number of messages have been defined within the architecture such as NEW.CHARACTER and UPDATE.CHARACTER. It is true that the messages currently implemented do not cover every eventuality within the framework. Once a finite set of messages has been decided upon, a formal protocol could be created which provides a concrete specification of the message passing within the framework. When integrating new modules with the framework a protocol would provide a base for developers to work from.

Although the set of messages passed through the system is finite, the content of these messages is not prescribed in any way. This is an important part of the openness of the framework in that it allows modules to add a wide variety of data objects to the system without forcing them to be normalised to a preset format leading to the discarding of potentially useful information.

The current mapping systems however, must be aware of the information that the Virtual Environment Interface requires and carry out their mappings accordingly. Tags such as VRColour and VRXPosition are used to describe the virtual object and interpreted by the VEI. Some of the tags used to describe the virtual objects could be formalised in the protocol that can then be adhered to by VEI's and mapping systems alike. This is not to say that interfaces can't have their own specific tags, but where two interfaces are referring to the same information it would seem appropriate for them to call the information by the same name. This would promote the creation of generic mapping modules.

For example, if it was decided that a VRColour tag would always be used to represent the colour of a virtual object and would always be in a given format, then mapping systems would become more generic still and be less dependent on the implementation of the interface. Other tags that might be standardised could include coordinate information describing positions for example. This would assume a fixed coordinate system that is adhered to by all of the interfaces. Where an interface has its own peculiar coordinate system, it would of course be able to use its own specific tags to hold the information. Perhaps this might necessitate the creation of a specific mapping module that maps between the standard coordinate system and the interface specific one.

For the above reasons and more, I believe that a re-implementation of the Minerva framework would benefit from the formalisation of a protocol to accompany it.

9.4 The Web Browser as an Integrating Visualisation

For historical reasons previously discussed in this thesis the present implementation of Minerva is tied into the implementation of the Microcosm system and uses the third party rendering software RenderWare. With hindsight, although providing a suitable platform for the prototype, Minerva would no doubt benefit from re-implementation using more generic tools.

There is no reason why the architecture should not be usable across multiple platforms and the Java language would have the added advantage in that it is easily integrated with systems such as the World Wide Web. The production of Virtual Environment Interface within a web browser as a plug in would also add to the distribution of the architecture. A slightly modified approach would see the Interface as a client connecting to a Minerva server, which in turn is attached to a variety of information sources.

If a 3D interface were created within a web browser, the VRML modelling language discussed in Chapter 6 would provide a useful tool for its construction. The conversion of virtual objects represented in XML to VRML objects would be a trivial matter, and many VRML plug-ins exist for current web browsers.

9.5 Evaluation of Visualisations

The evaluation of the Minerva framework has so far been limited to demonstrating the flexibility and openness of the framework by creating three visualisations of hypermedia information. The actual value of these visualisations over the previously used interface has not been determined.

Now that the architecture exists there is clearly scope for the rigorous design of visualisations intended to benefit the user rather than designed simply to evaluate the functionality and scope of the framework. These new visualisations could then be evaluated for their usefulness as tools for users. The openness of the architecture lends itself to the creation of novel visualisations beyond those that have been discussed in this thesis. Another added benefit of the open framework is that tools could be created that plug into the framework and assist in the evaluation by collecting information about the users activities while using the system. These tools could output information for statistical analysis as part of the evaluation process.

Various measures can be be used in order to assess the usefulness of a hypermedia system. In their study of the use of navigational tools in hyperspace (McDonald & Stevenson, 1998) asked test subjects to read a hypertext until they felt they had read all the material present. The number of different nodes opened, the number of repeated nodes visited and the number of navigational tools used was recorded. The subjects were then asked to answer ten questions whose answers could be obtained from the documents in the test data set. Information was recorded on the number of nodes opened above the minimum required to locate each answer, the time taken to find each answer, the accuracy of each answer and the number of times navigational tools were used while answering the questions. A similar test could be envisioned to test the visualisations described in the previous chapter. The results for test subjects using the visualisations could be compared to those of subjects just using the basic systems.

9.6 Synchronisation Issues

During the construction of the three visualisations presented in the previous Chapter, a number of synchronisation issues were encountered. The position of virtual link objects was dependent on the position of the objects they were connecting. This placed a requirement that the position of the source and destination objects be established *before* the position of the link was mapped. This can be achieved in the simple case by adding the link objects to the store last, or by carrying out the mapping of all the objects in the store twice at the cost of some redundancy.

The problem will only increase however once more complicated visualisations are created where objects move around the visualisation due to actions on the part of the user. Each time an object changes position, all objects dependant on that object will have to be remapped. This may well require dependency information to be included in the object store so that objects can be remapped whenever an object they are dependent upon is modified. With such a system, problems of cyclic dependencies will obviously have to be given careful consideration.

9.7 Implementation of Novel Visualisations

One of the driving forces behind the creation of the Minerva framework was that 3D visualisations of information could provide valuable assistance to users and that there is plenty of scope for implementing novel and powerful visualisation tools. Having created an open framework for the construction of such tools, the next step would clearly seem to be to design and build these tools and find out whether they do in fact enhance the users navigation of hypermedia information spaces. Two such tools, the graffiti tool and the egocentric visualiser were described in Chapter 7. The Minerva framework contains all of the tools needed to construct these visualisations and evaluation could be carried out to identify whether these novel approaches would in fact help users carry out their tasks.

Chapter 10

Conclusions

Historically, hypermedia systems have been written as self-contained applications that import information and manipulate it within the confines of the application itself. The information is stored processed and presented within a single process. Out of this environment came the approach of open hypermedia which sought to provide greater separation between the hypermedia linking and navigation information and the content upon which it was based. This allowed more flexible, dynamic linking strategies and allowed additional navigational tools to be connected to the systems in a more modular fashion.

By storing the hypermedia links separately from the content, it was no longer necessary to modify the documents when manipulating links. This allowed a wider range of media to be used within hypermedia applications, including read only media. Links could also be processed independently of the documents and different linkbases could be used to represent different threads through the document set. This provided a greater expressiveness in the construction of hypermedia applications as different authors could create their own linkbases, and hence structures. The reader of the application was then free to choose which hypermedia structure they wished to overlay on the underlying document set.

The openness of hypermedia systems need not be restricted to just the separation of links from content. The SHEP framework shows how openness can be applied to the interface as well as links, separating out the manipulation of the different components of the interface. This greater freedom provides the interface designer with new tools for producing scalable interfaces. This allows the interface to be easily adapted to changes in environment, hardware and user ability. The openness is not just of benefit to the designer of the interface however. With increased separation comes increased opportunity to express personal preference, with the user being able to tailor the interface to their own personal needs and whims. Instead of the user having to adapt to many different types of interface when dealing with different information systems, they can tailor the interfaces to look the same from system to system.

Exposing the interface provides a first stage at opening up hypermedia systems even further. The openness need not stop there however. The underlying hypermedia navigational information is often hidden from the user, to be presented in preset interfaces, often with a separate interface for each piece of navigational information. The user might be presented with a beautifully rendered map of the documents in the dataset showing their connections. The user might also wish to know which documents they have viewed at which point. This might lead to a list of documents being presented to them. Although both interfaces are related to the same dataset, the information is displayed separately and it is up to the user to make the connection between the two.

The Minerva framework overcomes this problem by exposing the raw navigational information in the system and providing a framework that allows flexible visualisations to be constructed from this information. The front-end interface displays information generated dynamically from the underlying information. This allows the information to be combined more cohesively, leading to more detailed visualisations.

The framework is constructed in an open fashion, with modules being plugged into the framework to deal with different aspects of the visualisation. The underlying navigational tools supply the framework with the raw information to be presented to the user. A variety of mapping processes convert and combine the information into objects to be displayed by the front-end interface. In addition to this, further intervention can be made by processes to control the interaction between the user and the objects and to restrict the amount of information displayed to the user.

The flexibility and openness of the framework were illustrated by creating visualisations of three very different forms of hypermedia information. Each visualisation required a different approach and made use of both tools designed specifically for the application as well as generic tools suitable for a wide range of visualisations. Within each visualisation there was also scope for user customisation.

With the underlying information exposed and an open framework available, new tools for visualisation can be created which aren't hard-wired to specific applications and can utilise any information that is fed into the framework.

Clear similarities can be seen between the SHEP and the Minerva architectures both in terms of aims and objectives and topology. In the SHEP architecture, information is passed from the interface component A, through a chain of processors B, and back to A again. In the Minerva architecture, the information is passed from an object manager A, through a chain of processors, B, and on to a visualisation interface C. There is a route back from C to A in the Minerva architecture which serves as a control path rather than a return path for the modified information.

Despite this minor difference, the question still remains, are both frameworks needed, or could they be combined? Although concerned with different types of information, both frameworks perform deferred decision making to provide separation between underlying data and interface.

In practice, the Minerva framework has more specialist modules connected to it than the SHEP framework. It could be argued that the specialist roles of control and movement interaction modules are in fact types of mapping libraries that act upon specific types of information. Here, the boundaries begin to blur however, and some of the benefits of clear separation between module function is lost.

Following along similar lines, SHEP could implemented within the Minerva framework itself. In this case, the scene is the desktop, with interface components registering as managers of their own state information. The shepherds form mapping modules, still arranged in a chain, but mapping the original state into its final state. In order for interface components to receive their mapped state they would register as handlers for the state objects and a virtual environment interface would simply take each message it received from the Minerva system and pass it straight back to the interface component. Arranged like this, the SHEP framework can be implemented as a special case interface within the Minerva framework.

Openness should not be restricted to linking practices, but provides flexibility in dealing with interfaces and visualisations. The modular construction of a framework promotes re-use and provides a stable platform for systems to feed their information into. The separation leaves the hypermedia systems free to concentrate on generating their information without worrying about the presentation of it to the user and the interface designers free to create novel and exciting interfaces without having to tie their interfaces to specific systems and representations.

Bibliography

Akscyn, R. M., McCracken, D. L., & Yoder, E. A. 1988. KMS: A Distributed Hypermedia System for Managing Knowledge in Organisations. *Communications of the ACM*, **31**(7), 820–835.

Alan Taylor. 1999. *The Evolution of Style Sheets.* http://www.webreference.com/dev/style/evolution.html.

Anderson, Kenneth M., Taylor, Richard N., & Whitehead, Jr, E. James. 1994 (Sept.). Chimera: Hypertext for Heterogeneous Software Environments. *Pages 94–107 of: ECHT '94 Proceedings, Edinburgh, Scotland.*

Andrew B. King. 1999. *Cascading Style Sheets.* http://www.webreference.com/dev/style/.

Andrews, Keith, Kappe, Frank, & Maurer, Hermann A. 1995. The Hyper-G network information system. *The journal of universal computer science*, **1**(4), 206–220.

Bederson, B. B., Stead, L., & Hollan, J. D. 1994. Pad++: Advances in Multiscale Interfaces. *In: ACM SIGCHI '94 (short paper).*

Benford, Steve. 1995. Information Visualisation, Browsing and Sharing in Populated Information Terrains. *In: Proceedings of the Seminar Series on New Directions in Software Development : The World Wide Web.* University of Wolverhampton.

Berners-Lee, Tim. 1992 (Jan.). *Uniform Resource Locators.* http://info.cern.ch/hypertext/WWW/Addressing/Addressing.html.

Berners-Lee, Tim. 1993a (Jan.). *Hypertext Markup Language (HTML).* http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html.

Berners-Lee, Tim. 1993b. *Protocol for the Retrieval and Manipulation of Textual and Hypermedia Information.* http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html.

Bernstein, Mark. 1988. The Bookmark and the Compass. *ACM Bulletin 9(4) of the Office Information System Group*, Oct., 34–45.

Bieber, Michael, Vitali, F., Ashman, Helen, Balasubramanian, V., & Oinas-Kukkonen, H. 1997. Forth generation hypermedia: Some missing links for the world wide web. *International Journal of Human-Computer Studies*, **47**(1), 31–65.

Brewster, Stepher A., Wright, Peter C., & Edwards, Alistair D. N. 1993 (Apr.). An Evaluation of Earcons for use in Auditory Human-Computer Interfaces. *Pages 222–227 of: Proceedings of INTERCHI '93 Human Factors in Computing Systems, Amsterdam, The Netherlands.*

Brown, Peter. 1987 (Nov.). Turning Ideas into Products: The Guide System. *Pages 33–40 of: Hypertext '87 Proceedings, Chapel Hill NC.*

Bush, Vannevar. 1945. As We May Think. *Atlantic Monthly 176*, July, 101–108.

Byrne, M. D. 1993 (Apr.). Using Icons to Find Documents: Simplicity Is Critical. *Pages 446–453 of: Proceedings of INTERCHI '93 Human Factors in Computing Systems, Amsterdam, The Netherlands.*

Card, S. K., & Henderson Jr, A. H. 1987. A multiple virtual workspace interface to support user task switching. *Pages 53–59 of: Proceedings of the CHI+GI 1987, Toronto.* ACM Press.

Card, Stuart K., Robertson, George G., & Mackinlay, Jock D. 1991. The information visualizer and information workspace. *Pages 181–188 of:* Robertson, Scott P., Olson, Gary M., & Olson, Judith S. (eds), *Proceedings of CHI 1991 Human Factors in Computing Systems, New Orleans, Louisiana.*

Card, Stuart K., Robertson, George G., & York, William. 1996 (Apr.). The WebBook and the Web Forager: An Information Workspace for the World-Wide Web. *Pages 111–117 of:* Tauber, Michael J. (ed), *Proceedings of CHI 1996 Human Factors in Computing Systems, Vancouver, British Columbia, Canada.*

Carlsson, C., & Hagsand, O. 1993. DIVE – A multi–user virtual reality system. *Pages 394–400 of: VRAIS'93, IEEE Virtual Reality Annual International Symposium.*

Carroll, J. M., Mack, R. L., & Kellogg, W. A. 1988. Interface Metaphors and User Interface Design. *Pages 67–85 of:* Helander, M. (ed), *Handbook of Human-Computer Interaction.* Elsevier.

Chalmers, M., & Chitson, P. 1992. Bead: Explorations in Information Visualisation. *Pages 330–337 of: Proceedings of SIGIR '92.* ACM Press.

Chen, Chaomei. 1999. Visualising Semantic Spaces and Author Co-Citation Networks in Digital Libraries. *Information Processing & Management*, **35**(3), 401–420.

Chen, Chaomei, Thomas, Linda, Cole, Janet, & Chennawasin, Chiladda. 1999. Representing the semantics of virtual spaces. *IEEE Multimedia*, **6**(2), 54–63.

Clarke, Arthur C. 1968. *2001: A Space Odyssey.* Hutchinson.

Conklin, Jeff. 1987. Hypertext: An Introduction and Survey. *Computer*, **1**(9), 17–40.

Criterion Software. 1998. *Renderware API Reference Manual V1.3.* Criterion Software Guilford, UK.

Crowder, Richard M., Wendy Hall, Rory Bernard, & Heath, Ian. 1993 (Nov.). Open Hypemedia Systems for Training and Maintenance. *In: European Conference on Automation and Robotics Training, London.*

Crowder, Richard M., Wills, Gary B., Heath, Ian, & Hall, Wendy. 1997 (Dec.). An Open Hypermedia Solution to Information Overload in Industrial Applications. *In: IEE colloquium on IT Strategies for Information Overload.*

Davis, Hugh. 1995 (Nov.). *Data Integrity Problems in an Open Hypermedia Link Service.* Ph.D. thesis, Department of Electronics and Computer Science, University of Southampton.

DeRoure, Dave C., Hall, Wendy, Reich, Sigi, Pikrakis, A., Hill, Gary J., & Stairmand, M. 1998. An open architecture for supporting collaboration on the web. *WET ICE 98 - IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford University, California*, June, 90–95.

Dieberger, Andreas. 1996. Browsing the WWW by interacting with a textural virtual environment – a framework for experimenting with navigational metaphors. *Pages 170–179 of: Hypertext '96 Proceedings, Washington DC, USA.* ACM Press.

Engelbart, D. G., Watson, R. W., & Norton, J. C. 1973 (June). The Augmented Knowledge Workshop. *Pages 9–21 of: AFIPS, National Computer Conference and Exposition, New York.*

Engelbart, Douglas. 1963. A Conceptual Framework for Augmenting Man's Intellect. *Vistas on Information Handling*, **1**, 1–29.

Ensor, J. Robert. 1998. Multimedia Communications Projects. *IEEE Multimedia*, Jan., 97–101.

Erickson, Thomas D. 1990. Working with Interface Metaphors. *Pages 65–74 of:* Laurel, Brenda (ed), *The Art of Human Computer Interface Design.* Addison Wesley.

Fairchild, K. M., Poltrock, S. E., & Furnas, G. W. 1988. Semnet: Three-dimensional Graphic Representations of Large Knowledge Bases. *Cognitive Science and its Applications for Human-Computer Interaction.*

Fountain, Andrew, Hall, Wendy, Heath, Ian, & Davis, Hugh. 1990. Microcosm: an Open Model With Dynamic Linking. *Pages 298–311 of:* Rizk, A., Streitz, N., & J.André (eds), *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext, INRIA.* Cambridge University Press.

Furnas, G. W. 1994. Generalised Fisheye Views. *Pages 206–216 of: Proceedings of CHI 1986 Human Factors in Computing Systems, Boston, Mass.* ACM Press, New York.

Furnas, George W., & Bederson, Benjamin B. 1995 (May). Space-Scale Diagrams: Understanding Multiscale Interfaces. *In: Proceedings of CHI 1995 Human Factors in Computing Systems, Denver, Colorado, USA* .

Gibson, William. 1984. *Neuromancer.* Harper Collins, London.

Gloor, P. A. 1991. CYBERMAP - Yet Another Way of Navigating in Hyperspace. *Pages 107–122 of: Hypertext '91 Proceedings, San Antonio, Texas.* ACM Press.

H. Gleitman. 1986. *Psychology (2nd Ed.).* W.W.Norton & Co. Inc. USA.

Halasz, F., & Mayer, S. 1994. The Dexter Hypertext Reference Model. *Communications of the ACM*, **37**(2), 30–39.

Halasz, F., & Schwartz, M. 1990. The Dexter Hypertext Reference Model. *Pages 95–133 of: Proceedings of the Hypertext Standardization Workshop, Gaithersburg.* US Government Printing Office.

Halasz, F. G., Moran, T. P., & Trigg, R. H. 1987 (Apr.). Notecards in a Nutshell. *Pages 42–52 of: Proceedings of CHI '87 Human Factors in Computing Systems, Toronto, Ontario, Canada.*

Harrison, Beverly L., & Vicente, Kim J. 1996 (Apr.). An Experimental Evaluation of Transparent Menu Usage. *Pages 391–398 of:* Tauber, Michael J. (ed), *Proceedings of CHI 1996 Human Factors in Computing Systems, Vancouver, British Columbia, Canada.*

Heath, Ian. 1992 (Aug.). *An Open Model for Hypermedia: Abstracting Links from Documents.* Ph.D. thesis, Department of Electronics and Computer Science, University of Southampton.

Hemmje, Matthias. 1993. A 3D based user interface for information retrieval systems. *Pages 194–209 of: Database Issues for Data Visualization IEEE Visualization '93 Proceedings, San Jose, California, USA.* Springer Verlag.

Henderson, D. A., & Card, S. K. 1986. Rooms: The use of multiple virtual workspaces to reduce spatial contention in a window-based graphical user interface. *ACM Transaction on Graphics*, **5**(3).

Hendley, R. J., Drew, N. S., Wood, A. M., & Beale, R. 1995 (Oct.). Narcissus: Visualising Information. *Pages 90–96 of: Proceedings of IEEE Symposium on Information Visualisation (InfoVis'95), Atlanta, Georgia, USA.*

Hightower, R. R., Ring, L. T., Helfman, J. I., Bederson, B.B., & Hollan, J. D. 1998. Graphical Multiscale Web Histories: A Study of PadPrints. *In: Hypertext '98 Proceedings, Pitsburgh, USA.* ACM Press.

Hill, Gary J. 1994 (June). *Extending an Open Hypermedia System to a Distributed Environment.* Ph.D. thesis, Department of Electronics and Computer Science, University of Southampton.

Hill, W. C., & Hollan, J. D. 1992. Edit Wear and Read Wear. *Pages 3–9 of: Proceedings of CHI 1992 Human Factors in Computing Systems.*

Hoschka, Philipp. 1998. An Introduction to the Synchronized Multimedia Integration Language. *IEEE Multimedia*, **5**(4), 84–88.

Jukka Korpela. 1999. *Why style sheets are harmful.* http://www.hut.fi/%7Ejkorpela/styles/harmful.html.

K. Grønbæk and R. H. Trigg. 1992 (Nov.). Design Issues for a Dexter-Based Hypermedia System. *Pages 191–200 of: ECHT '92 Proceedings, Milano, Italy.*

Kandogan, Eser, & Shneiderman, Ben. 1997 (Mar.). Elastic Windows: Evaluation of Multi-Window Operations. *In: Proceedings of CHI 1997 Human Factors in Computing Systems, Atlanta, Georgia, USA.*

Kay, Alan, & Goldberg, A. 1977. Personal dynamic media. *Pages 31–42 of: IEEE Computer*, vol. 10.

Ken Perlin and David Fox. 1993. Pad : An Alternative Approach to the Computer Interface. *Pages 57–64 of: SIGGRAPH '93 Computer Graphics conference proceedings, Anaheim, California.* ACM SIGGRAPH, New York.

Krohn, U. 1996. VINETA: Navigation Through Virtual Information Spaces. *In:* Cartaci, T. (ed), *Proceedings of AVI: Advanced Visual Interfaces, Gubbio, Italy.* ACM Press.

Li, Zhuoxun, Davis, Hugh, & Hall, Wendy. 1992. Hypermedia Links and Information Retrieval. *In: The Proceedings of the 14th British Computer Society Research Colloquium on Information Retrieval.* Lancaster University.

Licklider, J. C. R. 1960. Man-computer symbiosis. *Pages 4–11 of: IRE Transactions on Human Factors in Electronics HFE-1*, vol. 1.

Lieberman, Henry. 1997 (Mar.). Autonomous Interface Agents. *In: Proceedings of CHI 1997 Human Factors in Computing Systems, Atlanta, Georgia, USA.*

Lynch, K. 1960. *The image and the city.* MIT Press & Harvard University Press.

Mackinlay, Jock D., Robertson, George G., & Card, Stuart K. 1991. The perspective wall : detail and context smoothly integrated. *Pages 173–179 of:* Robertson, Scott P., Olson, Gary M., & Olson, Judith S. (eds), *Proceedings of CHI 1991 Human Factors in Computing Systems, New Orleans, Louisiana.*

Malcolm, Kathryn C., E., Poltrock S., & D., Schuler. 1991. Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise. *Pages 13–24 of: Hypertext '91 Proceedings, San Antonio, Texas.* ACM Press.

Marshall, Catherine C., & Shipman III, Frank M. 1993 (Nov.). Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext. *Pages 51–62 of: Hypertext '93 Proceedings, Seattle, Washington USA.*

Massari, A., Saladini, L., Hemmje, M., & Sisinni, F. 1997. Virgilio: A Non-Immersive VR System To Browse Multimedia Databases. *Pages 573–580 of: Proceedings of the IEEE International Conference on Multimedia Computing Systems 1997, Ottowa, Canada.* IEEE Computer Society Press.

McDonald, Sharon, & Stevenson, Rosemary J. 1998. Navigation in hyperspace: An evaluation of the effects of navigational tools and subject matter expertise on browsing and information retrieval in hypertext. *Interacting with Computers*, 129–142.

Mereu, Stephen W., & Kazman, Rick. 1996 (Apr.). Audio Enhanced 3D Interfaces for Visually Impaired Users. *Pages 72–78 of:* Tauber, Michael J. (ed), *Proceedings of CHI 1996 Human Factors in Computing Systems, Vancouver, British Columbia, Canada.*

Morningstar, Chip, & Farmer, F. Randall. 1991. The Lessons of Lucasfilm's Habitat. *Pages 273–302 of:* Benedikt, Michael (ed), *Cyberspace : First Steps.* MIT Press Cambridge, Massachusetts.

Nelson, Theodore Holm. 1980 (Oct.). Replacing the Printed Word: A Complete Literary System. *Pages 1013–1023 of: IFIP.*

Nielsen, Jacob. 1995. *Multimedia and Hypertext. The Internet and Beyond.* Academic Press Professional, Boston.

Olsen, K. A., Korfhage, R. R., Sochats, K. M., Spring, M. B., & Williams, J. G. 1993. Visualisation of a Document Collection: The VIBE System. *Pages 69–81 of: Information Processing and Management*, vol. 29. Pergamon Press Ltd.

Pearl, A. 1989 (Nov.). Sun's Link Service: A Protocol for Open Linking. *Pages 137–146 of: Hypertext '89 Proceedings, Pittsburgh, Pennsylvania USA.*

Pesce, Mark D., Kennard, Peter, & Parisi, Anthony S. 1994. *Cyberspace.* http://vrml.wired.com/concepts/pesce-www.html.

Pintado, X., & Tsichritzis, D. 1990. Satellite: Hypertext navigation by affinity. *Pages 274–287 of:* Rizk, A., Streitz, N., & J.André (eds), *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext, INRIA.* Cambridge University Press.

Rao, Ramana, & Card, Stuart K. 1994 (Apr.). The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. *In: Proceedings of CHI 1994 Conference on Human Factors in Computing Systems, Boston, Massachusetts.*

Rheingold, Howard. 1991. *Virtual Reality.* Secker and Warburg.

Rizk, Antoine. 1992 (Nov.). Multicard: An open hypermedia system. *Pages 4–10 of: ECHT '92 Proceedings, Milano, Italy.*

Robertson, George G., Mackinlay, Jock D., & Card, Stuart K. 1991. Cone trees : animated 3D visualizations of hierarchical information. *Pages 189– of:* Robertson, Scott P., Olson, Gary M., & Olson, Judith S. (eds), *Proceedings of CHI 1991 Human Factors in Computing Systems, New Orleans, Louisiana.*

Schneiderman, B. 1987 (Nov.). Hypermedia topologies and user navigation. *Pages 189–194 of: Hypertext '87 Proceedings, Chapel Hill NC.*

Seligmann, Doree Duncan, Laporte, Cati, & Bugaj, Stephan Vladimir. 1997. The Message is the Medium. *Pages 631–641 of: Proceedings of the Sixth International World Wide Web Conference, Santa Clara, California, USA.* .

Shipman, F. M., Furuta, R., Brenner, D., Chung, C. C., & Hsieh, H. W. 1998. Using paths in the classroom: Experiences and adaptations. *Pages 267–276 of: Hypertext '98 Proceedings, Pitsburgh, USA.* ACM Press.

Shipman III, Frank M., & Marshall, Catherine C. 1995 (May). Finding and Using Implicit Structure in Human-Organized Spatial Layouts of Information. *In: Proceedings of CHI 1995 Human Factors in Computing Systems, Denver, Colorado, USA* .

Silicon Graphics. 1995. *3D File System Navigator.* http://www.sgi.com/fun/freeware/3d_navigator.html.

Smith, D. 1982. Designing the Star User Interface. *BYTE*, Apr., 242–282.

Stephenson, Neal. 1993. *Snowcrash.* Roc.

Storey, M-A. D., & Müller, H.A. 1995 (Oct.). Manipulating and Documenting Software Structures Using SHriMP Views. *Pages 275–284 of: Proceedings of the ICSM '95 conference on Software Maintenance, Opip (Nice), France.*

Sutherland, Ivan. 1965. The Ultimate Display. *Proceedings of the IFIP Congress*, 506–508.

Sutherland, Ivan. 1968. A Head–Mounted Three–Dimensional Display. *Proceedings of the Fall Joint Computer Conference*, 757–764.

Sutherland, Ivan E. 1963. Sketchpad: a man-machine graphical communication system. *Pages 329–346 of: AFIPS Conference Proceedings 23.*

Took, Roger. 1990. Surface Interaction: A Paradigm and Model for Separating Application and Interface. *Pages 35–42 of:* Chew, Jane Carrasco, & Whiteside, John (eds), *Proceedings of CHI 1990 Human Factors in Computing Systems, Seattle, Washington.*

Trigg, R., Suchman, L., & Halasz, F. 1986 (Dec.). Supporting collaboration in Notecards. *Pages 153–162 of: CSCW '86 Conference, Austin, Texas.*

Van Dam, A. 1988. Hypertext '87 Keynote Address. *Proceedings of the ACM*, **31**(7), 887–895.

Wardrip-Fruin, N., Meyer, J., Perlin, J., Bederson, B. B., & Hollan, J. D. 1997. A Multiscale Narrative: Gray Matters. *Page 141 of: ACM SIGGRAPH 97 Visual Proceedings.*

Ware, Colin, Hui, David, & Franck, Glenn. 1993 (Oct.). Visualizing Object Oriented Software in Three Dimensions. *Pages 612–620 of: Proceedings of CASCON '93 (IBM Centre for Advanced Studies), Toronto, Ontario, Canada.*

Wiil, Uffe Kock, & Leggett, John J. 1996. The HyperDisco Approach to Open Hypermedia Systems. *Pages 140–148 of: Hypertext '96 Proceedings, Washington DC, USA.* ACM Press.

Wilkins, Robert James. 1994 (Sept.). *The Advisor Agent: a Model for the Dynamic Integration of Navigation Information within an Open Hypermedia System.* Ph.D. thesis, Department of Electronics and Computer Science, University of Southampton.

Wood, A. M., Drew, N. S., Beale, R., & Hendley, R. J. 1995 (Apr.). Hyperspace: Web Browsing with Visualisation. *Pages 21–25 of: Third International World-Wide Web Conference Poster Proceedings, Darmstadt, Germany.*

Worlds Inc. 1993. *Alphaworld home page.* http://www.worlds.net/alphaworld/.

Yankelovich, N., Haan, B. J., Meyrowitz, N. K., & Drucher, S. M. 1988. Intermedia : The concept and the construction of a seamless information environment. *Computer*, **1**(1), 81–96.