HYPERMEDIA INTEROPERABILITY: NAVIGATING THE INFORMATION CONTINUUM

By
David Edward Millard

A thesis submitted for the degree of
Doctor of Philosophy

Department of Electronics and Computer Science,
University of Southampton,
United Kingdom.

Dec 2000

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

ELECTRONICS AND COMPUTER SCIENCE DEPARTMENT

Doctor of Philosophy

Hypermedia Interoperability: Navigating the Information Continuum

by David Edward Millard

Open Hypermedia Systems are designed to allow links to be authored and followed on top of any media format. The link structures are held separately from the documents in a software component called a Link Server. As hypermedia has matured as a research topic attention has turned to standardising the way in which components talk to Link Servers in order to provide interoperability.

The Open Hypermedia Systems Working Group took up this challenge and proposed an Open Hypermedia Protocol (OHP). However, the scope of this proposal proved to be too large and the protocol was divided into domain specific parts (Navigational, Spatial and Taxonomic Hypermedia), tackling each domain differently, but consistently. It is questionable whether this step was the correct one, as the domains share many similar features.

In this thesis I present a detailed examination of the information spaces that the OHP was attempting to model (from all these considered hypertext domains), which incorporates notions of both behaviour and context. This examination looks at what it means to navigate around the many dimensions of information, across these domains, and reveals a cohesive and continuous structure that I call the Information Continuum.

The Fundamental Open Hypermedia Model (FOHM) is presented, which is capable of representing the structures of this continuum in a consistent and meaningful way. FOHM is coupled with an agent infrastructure to produce an implementation that demonstrates the model being used for cross-domain interoperability.

Contents

List of Figures

Acknowledgements

I would like to thank my supervisor, Hugh Davis, for his support and advice throughout the work described in this thesis. In particular I appreciate his willingness to involve me in his international research and also his guidance through the minefield of Hypertext Literature. I would also like to thank Wendy Hall for her enthusiasm and encouragement, not to mention her amazing ability to find the funds that sent me to numerous workshops and conferences, seemingly out of thin air.

My thanks are also extended to the many members of the IAM research group that have helped me in one way or another over the years. Mark Weal, Ian Heath, Luc Moreau, Dave DeRoure, Steve Blackburn, Gareth Hughes, Danius Michaelides and Jon Griffith. Most of all I am indebted to Sigi Reich for his incredible patience during our numerous discussions. Without his contributions I could never have reached as far as I have.

I must also thank my parents for their support and encouragement, not only over the last six years, but over the last twenty four. Its been a long time coming but it finally looks like I might get a job.

Finally, I must thank all the friends who have listened to my ramblings and helped keep me sane. In particular special thanks go to my girlfriend Jo Bird for all her love and understanding, and for helping me remember what it's really all about.

Chapter 1

Introduction

> *"Far from being economic and demonstrably advantageous, the immense number and variety of human idioms, together with the fact of mutual incomprehensibility, is a powerful obstacle to the material and social progress of the species."*
>
> GEORGE STEINER, AFTER BABEL

Hypermedia systems were first described hypothetically in Vannevar Bush's landmark 1945 paper 'As We May Think' (Bush, 1945). Bush describes machines designed to help people think and communicate. Machines that bridge the physical, social and intellectual gaps that otherwise divide us, by helping people locate, navigate and absorb information.

Due to the tremendous technical challenges involved it wasn't until the advent of the electronic computer and the work of Engelbart some two decades later that a real system, NLS/Augment, was developed (Engelbart, 1963). During the same period Ted Nelson was developing his system, Xanadu, with the aim of creating a global literary environment. It was Nelson who first coined the term 'hypertext' to refer to non-linear text, a method with which to present and navigate this environment (Nelson, 1987).

The early 1990's saw the emergence of a dichotomy of approaches to creating these hypertext systems. One approach resulted in the development of Open Hypermedia Systems (OHSs) (see Section 2.6 for an overview). These systems maintain links as first class objects which are stored separately from documents. This enables the system to be applied to an open set of file formats (including video and audio media) and also enables powerful searching and indexing of the information held in the system, as well as assurance of link consistency. The other approach resulted in the World Wide Web

(WWW) (Berners-Lee *et al.*, 1994) a distributed file retrieval system (Nürnberg & Ashman, 1999) that uses embedded link information to enable a browser to follow hypermedia pathways through documents.

Although far further from the vision of Bush, it is this latter system that has enjoyed unimagined popularity over the last few years, while the functionally more advanced OHSs have enjoyed only limited commercial success. This is because of two overwhelming factors that support the WWW, distribution and interoperation. Because the system is distributed, many people can access the same pool of information. This is aided by the standard formats employed in the WWW, both the file format (HTML) and networking protocols (http) are standard across all browsers and platforms. This results in network effects, where the value of the system grows proportionally to the number of users (Whitehead, 1999).

In recognition of the importance of interoperation the Open Hypermedia Systems Working Group (OHSWG), of which the author is a part, has been working to create a standard Open Hypermedia Protocol (OHP) that will enable different OHSs to communicate (Davis *et al.*, 1996). This has brought many different approaches to hypermedia together in a common research effort. It was quickly realised that to successfully create a standard protocol for interoperability a standard data model was required, which could represent all the structures that the various systems required in a sensible and complimentary fashion.

To complicate matters further it was realised that some OHSs operated in a totally different paradigms then others. This resulted in the definition of several hypertext domains, the most frequently discussed of which are:

1. *Navigational Hypertext* : Point and click navigation between documents.
2. *Spatial Hypertext* : The spatial organisation of information, where implicit colour, shape and size structure is managed explicitly by the system.
3. *Taxonomic Hypertext* : The structuring of information into related categories, forming taxonomies, such that the information can be easily reasoned about.

1.1 Overview of Research

The interoperability work of the OHSWG was originally based on a simple text based version of OHP that enabled clients to retrieve link information from servers. Much work was undertaken by the group, including myself, to implement and test this basic protocol, resulting in the first demonstration of Open Hypermedia interoperability at the 1998 ACM Conference on Hypertext.

However, the OHSWG is a diverse group with varied research interests and the OHP grew as a result of different peoples concerns. While other researchers worked on issues such as collaborative hypertext and workflow I concentrated on extending the basic definition of services that existed in the early protocol into a more powerful generic mechanism for the dynamic discovery and invocation of functionality, enshrined in OHP-Service a parallel protocol to the OHP. At the same time the groups understanding of protocol design was growing and OHP was redefined using XML and split into a suite of complimentary protocols, OHP-Nav, OHP-Space and OHP-Tax. Initially only OHP-Nav was defined.

To start with the OHP was concerned only with the domain of Navigational Hypertext and it was assumed that other protocols for other domains would be created at a later date (Davis *et al.*, 1999). However, I and a few other researchers took the view that since the relational information represented in all three domains has approximately the same form, it would be appropriate to consider the commonalties of the domains and explore the possibility of creating a single unifying data model. This would allow a navigational browser to explore information generated in a spatial editor, or a taxonomic 'crawler' to reason about the information it found on a variety of servers, whatever the domain the information was authored in. To do this required an investigation into not only the scope and form of each domain but also the nature of hyperstructure and information spaces.

The result of my work in this area was the Fundamental Open Hypermedia Model (FOHM) (Millard *et al.*, 2000). FOHM builds on the data model that was developed for OHP-Nav, but excludes the communication and formatting information that complicated the OHP suite. FOHM takes the approach that the hypertext model should be general enough to support all the structures in the three supported domains, but specific enough that performance does not suffer.

Each of the domains contains structure not found in the others, but to enable a common model FOHM has to incorporate all structures. This results in some surprising consequences, as many advanced features peculiar to specific hypertext systems emerge. For example, MEMOIR (Pikrakis *et al.*, 1998) is a system that, on request, records the paths that users travel throughout the system. These 'trails' are then available to other users as first class objects in the system. Structures found in Spatial hypertext (such as lists and queues) would appear in Navigational hypertext in much the same way. Conversely a Navigational trail would appear as a Spatial list in a Spatial browser.

This cross-fertilisation of domains results in a powerful generic model, where structures from one domain appear sensibly and consistently in the others.

The earlier work that had been undertaken on OHP and later the OHP-Nav and OHP-Service protocols was moving towards an advanced infrastructure for communication between components. In particular it was becoming apparent that the dynamic discovery and invocation properties of OHP-Service would be useful to many different kinds of application; indicating that it might be better located in some form of infrastructure layer.

Independently of the OHP effort other researchers within the IAM group at Southampton were working on agent frameworks and developing a prototype system known as SoFAR (Moreau *et al.*, 2000). This system fitted in very well with ideas of service discovery at the application level and after an investigation into the communication paradigm of autonomous agents and their proposition based languages, a proof of concept implementation of FOHM was created using the SoFAR agent framework.

1.2 Contributions

The work presented in this thesis contributes to the area of hypermedia interoperability, more specifically it looks independently at both *how* hypermedia systems communicate and also *what* they communicate.

This contribution has taken form in three ways :

1. *The Work of the OHSWG* : As part of the OHSWG I was a major contributor to the development and implementation of OHP and the subsequent creation and definition of OHP-Nav.
2. *A Conceptual Model of the Information Continuum* : The work of the OHSWG became a foundation for my own exploration of the different hypermedia domains. The development of the 6D Model and notion of the Information Continuum represents an advance in the understanding of how hypermedia can be used to navigate information spaces and how the various domains of hypermedia relate to one another.
3. *The Development of FOHM* : This cross-fertilisation of hypermedia domains resulted in the definition of FOHM as a cross-domain model for communication. An implementation has been constructed based on the SoFAR agent framework, maintaining a clean separation between the mechanisms and content of communication that was not present in the OHP suite of protocols.

1.3 Outline of this Document

This thesis describes the evolution of the OHP draft proposal into the OHP suite of protocols, of which OHP-Nav, the navigational protocol, is only one part. It then uses that

work as a basis to consider the semantic content and syntactic form of any interoperability attempt. Finally it describes the FOHM model and presents the FOHM SoFAR implementation.

Chapter 2 presents a review of past hypermedia research and systems in order to provide a context for the following interoperability work.

Chapter 3 describes the OHP definition that I first worked on and helped evolve into OHP-Nav. It also describes the implementation and subsequent demonstration of interoperability at Hypertext'98 in Pittsburgh, PA.

Chapter 4 builds on this OHP-Nav definition to present OHP-Service, a parallel protocol developed by Sigi Reich and myself for the dynamic discovery and invocation of services. It also looks at parallel work in the field of Structural Computing. Together these two research threads throw up questions about the nature of hypermedia functionality and the place of the OHP suite in a world where generic communication infrastructures are beginning to emerge.

Chapter 5 tackles the issues of communication and syntax. In particular exploring the separation of semantic content from syntactic constraints and turning to the agent and human worlds of communication for inspiration.

Chapter 6 examines what it means to navigate through information systems. Examining the different roles of hypermedia domains in the information world and producing a conceptual model of the information continuum.

Chapter 7 builds on this conceptual model to produce FOHM, a Fundamental Open Hypermedia Model, capable of consistently and sensibly encoding the structures from any of the three domains explored. This involves examining what unique properties each domain brings to the hyperstructure, including internal link structure and a dependance on context.

Chapter 8 presents the SoFAR agent framework developed by other researchers within the IAM group and explains how this infrastructure facilitated a prototype FOHM implementation. The lessons of this implementation are explored and the FOHM approach is compared to that of Structural Computing to see in which ways they are compatible.

Chapter 9 concludes this thesis, drawing together the various threads of research and evaluating the interoperability possibilities in future information, or structural systems. It also describes some of the future work possibilities including the exploration of context and behaviour within FOHM.

1.4 Declaration

Although this entire thesis is the author's personal view of the hypermedia field, the work described has been conducted within the common research efforts of the OHSWG. The original contributions are described in Chapters 5, 6, 7 and 8, while Chapter 4 describes work undertaken with the assistance of Sigi Reich and Jon Griffith. The exceptions to this are Sections 5.3 and 8.2 which draw on other research fields and Section 7.3.2 which describes work undertaken alongside Sigi Reich and Luc Moreau within the research group at Southampton.

Chapter 2

Background

2.1 Introduction

In this chapter I will explore the historical and technical background upon which this thesis builds. Firstly I will describe the pioneering work undertaken in the hypertext field and the systems that were subsequently developed in the following decades.

Using Frank Halasz's excellent dissection of hypermedia issues as a starting point I will then go on to describe modern hypermedia and open hypermedia systems and models. Which, while evidently more advanced than their early counterparts, still do not address all of Halasz's points.

Finally I will reflect on recent trends in hypermedia research and in particular concentrate on the interoperability efforts that were beginning to emerge at the start of the work associated with this thesis.

2.2 Pioneers

When they first appeared computing machines were applied to problems of a mathematical nature, complex calculations and data analysis tasks. In Bush's landmark 1945 paper 'As We May Think' (Bush, 1945), he describes how this computing power could be used to manage and store the increasing volume of information that a complex civilisation generates. He calls for a post-war effort to mechanise the scientific literature system.

Bush describes the 'memex' a theoretical device in which an individual may store all of their books, records and communications; a device that is mechanised to allow rapid and flexible consultation of that material. He also describes the associative nature of human memory and how, by mimicking this ability in a machine, we may assist thinking.

He envisages trails, recorded non-linear paths though this information, trails that can be amended, annotated and shared easily with others.

Although the memex is described as being implemented with mechanical machinery and microfiche it is still a remarkable vision; one that, even with the electronic computing power we have today, has not yet been fully realised.

Engelbart developed the ideas for such a real system some two decades later at the Stanford Research Institute. In 1963, they had been implemented in NLS (oN Line System)/Augment (Engelbart, 1963). NLS was a sophisticated system that included the use of television images and the first appearance of the mouse. Files in NLS were structured as a hierarchy of statements. Links could exist between any two files or between statements in a file.

NLS was implemented in three major parts; a database of non-linear text, view filters that took selected information from this database and views that structured and displayed this information. The viewing filters could clip the hierarchy at a particular level (depth) or truncate the number of items displayed. The system also allowed the construction of customised filters (defined in a 'high level content analysis language') that would display only statements containing a certain content.

During the same period Ted Nelson was developing his system, Xanadu, with another idea in mind: to create a global literary environment. The long-term goal, described in his book 'Literary Machines' was to place the entire world's literary corpus on line (Nelson, 1987). It was here that Nelson coined the term 'hypertext' to refer to non-linear text. Xanadu was designed to save space via the heavy use of links (known as transclusions), only original documents and changes are saved, the system reconstructing any version of a document for display. Emphasis was placed on maintaining copyright and dealing with accounting and royalty distribution.

In 1985 Yankelovich and Meyrowitz considered the differences between printed and electronic media (Yankelovich *et al.*, 1985), formulating a set of capabilities that electronic document systems should posses to maximise the advantages of electronic media while overcoming some of the disadvantages.

For example, they make the point that printed media is more aesthetically pleasing then electronic media and not subject to inevitable technological change. It is also portable and produced using well-defined and accepted standards. While electronic documents have the advantage of being able to deal with three dimensional subjects and temporal media as well as allowing the construction of associative webs of links that allow rapid browsing of information. They also allow the same information to be stored in

a variety of versions and displayed in a variety of styles.

Considering these differences Yankelovich and Meyrowitz require a system to provide tools for:

- Promoting connectivity
- Promoting audio visualization
- Creating and revising documents
- Browsing, searching, customising, and retrieving information
- Preserving the historical integrity of information

In his classic survey (Conklin, 1987), Conklin maintains that links are the essential feature of a hypertext system and that other common features (such as text processing facilities and windowed views) are merely an extension of this basic concept. He also acknowledges the extension of hypertext to other media (hypermedia) in which the elements that are networked together can be text, graphics, digital sound, animations or any other type of data.

Conklin also discusses the advantages and disadvantages of hypertext. He concludes that there are two fundamental problems in hypertext systems. The first is described as disorientation or the ‘lost in space’ problem (van Dam, 1988). This results from having to deal with a non-linear path, the reader must know both where they are within the network and where they are going. As hypertext offers more degrees of freedom then linear text, more dimensions in which one can move, it is easier for a reader to become lost or disorientated.

The second problem involves the cognitive overhead needed to create, name and keep track of links. This is described as the ‘cognitive overload’ problem. It is important in a hypertext system that the system itself does not distract the reader from the actual material.

However, Conklin also summarises the advantages of hypertext:

- The ease of tracing reference material either forward to a referent or backward to the reference.
- The ease of creating new references and the ability of a user to generate their own networks or annotate someone else’s.
- The information structuring abilities of hypertext tools, both hierarchical and non-hierarchical.
- The facility to support global views on a large amount of data, supporting easy reconstructing of large or complex documents.

- The ability to thread text segments together in many ways, creating customised documents that can serve many functions.
- There is modularity of information since the same text segment can be referenced from several places, creating less overlap and duplication.
- References move with the associated text creating a consistency of information.
- The user can have several paths of enquiry open at any one time (task stacking) such that any given path can be unwound to the original task.
- Support for collaboration. Several authors can collaborate on the creation (or annotation) of any document.

2.3 Early Systems

The early hypertext systems that followed the work of these pioneers were monolithic in nature, in that they used their own proprietary formats and were constructed as single large applications. They did however allow experimentation with different forms of hypertext and allowed researchers to address the problems identified by Conklin.

2.3.1 ZOG and KMS

ZOG, described as a ‘general purpose human-computer interface system’ (McCracken & Akscyn, 1984), was developed at Carnegie-Mellon University (CMU) in the late 1970’s and early 80’s. It was the first hypertext system to be used and evaluated extensively in the field.

ZOG is based around a basic unit of representation called a frame; a screen full of information. ZOG itself was implemented with purely textual frames although graphics were considered in the design. The frames are represented in a hierarchy and contain both information and possible selections. The user can make selections with a pointing device (e.g. mouse or touch screen) or via the keyboard. The selection is associated with an action, where an action is a sequence of commands in the ZOG language. This language contains commands for traversing the network, invoking utilities and entering an editor.

The ZOG system was written with a broad set of principles in mind known as the ‘Philosophy of ZOG’. These ideas involved the use of a consistent, total user environment to avoid the cognitive overhead of dealing with multiple interfaces. The system was flexible and allowed the user to directly manipulate data and automate complex operations that were often repeated. There was a concerted attempt to create a low learning curve and provide a safe environment suitable for exploring and learning by doing.

In essence the ZOG system was designed to provide simple structure to information that could be used by both novices and experts with little effort. This was put to the test in 1980 when a project began to put an implementation of ZOG onboard the U.S.S. Carl Vincent (a U.S. Navy Aircraft Carrier). Early in 1983 a complete ZOG-based system was installed onboard the ship. Running over 28 networked PERQ computers, it contained (initially) over 20,000 frames.

The system was used to both write and display the Ship Organisation and Regulation Manual (SORM). In addition Technical manuals where created for the aircraft and weapons elevators. The online version of these manuals were integrated with videodisk material, the ZOG frames imposing structure on what would otherwise be an essentially flat format.

During the first month of installation there were approximately 500 sessions of ZOG use on board the carrier. After six months usage had increased considerably with 30 serious users and the involvement of over 85 percent of the ship's departments.

ZOG was a robust real system that was operationally tested and proved to be useful. It was capable of assimilating many different types of application due to the generality of the frame model. However its authors criticise it for depending on high network and disk speeds for efficient use and also for its lack of graphical ability or of a fast database query language. ZOG's dependence on a consistent, total interface prevented them from making the leap to open hypermedia.

In 1981 the team from CMU behind ZOG formed the company 'Knowledge Systems' and began work on the successor to ZOG, known as KMS (Akscyn *et al.*, 1988). KMS built on the successes of the previous system, using a database of frames that could contain any combination of text and graphics. ZOG only allowed the display of a single frame on screen at any one time. KMS allows a single card taking the entire screen or two taking half each.

In addition to the more graphical nature of the product, KMS was also different to ZOG in that it made no distinction between editing and browsing, preferring a single consistent interface across both modes of use. The KMS team likens KMS to a reduced instruction set computer (RISC) in that it achieves performance through simplicity.

They also make the important observation that a systems data model impacts significantly on system behaviour. For example in KMS, where the browsable units are screen-sized frames, collaboration becomes easier as only small segments of any single document (the frames) need to be locked at any one time. Versioning is similarly simplified, as only different versions of frames need to be stored, the system compiling them

into the chosen cohesive document at runtime.

KMS continued the simplicity of the ZOG system while involving more complex data types. Although the system itself became more complex, it had the advantage of faster hardware and they were able to keep link following times down to sub-second standards. In combination with the simple nature of the data model this meant that KMS became appropriate for a number of tasks related to hypertext browsing. Bulletin boards were easily implemented using public writable frames and e-mail was implemented using frames with access restricted to the parties involved in the conversation.

Using KMS as a general file management and communication framework was very advantageous as it led to a much higher level of system integration. In fact the KMS team even suggested that given some standardisation of hypermedia, perhaps built around the data model, hypermedia may eventually replace the desktop metaphor as the reigning human-computer interaction paradigm.

2.3.2 Notecards

Notecards was a hypermedia system developed by Randall Trigg, Thomas Moran and Frank G. Halasz at Xerox Parc. It is general hypermedia environment run on desktop workstations, designed for use by small workgroups.

It is implemented within the Xerox Lisp programming language and is designed around two primitive constructs, notecards and links. A notecard is an electronic representation of a standard paper notecard. Each one can contain text or images and is displayed in a windowed environment so that many notecards can be shown at once. A link is used to interconnect these notecards. Each link is a typed directional connection between a source and destination notecard. Links are anchored in a specific location in the source notecard but the destination is always an entire notecard.

In addition there are special cases of notecards. Browsers contain a structure diagram of a network of notecards, represented via the title of each notecard contained in a box with the links between notecards represented as lines between these boxes. The user can actually use these browsers to edit the underlying hypermedia structure directly. The system also contains fileboxes that are used to organise or categorise large collections of notecards. The system requires that all notecards (including fileboxes) must be filed in a filebox.

In 1989 Notecards was extended to include a guided tour facility. The aim was to make hypertext more intelligible for readers as there were no conventions at the time as to how a hypertext should be presented.

Marshall et al, describe how a hypertext, although network-like in structure, is still viewed in a linear fashion by the reader through the following of a series of links or relationships (Marshall & Irish, 1989). This introduces problems such as maintaining coherence and keeping track of context.

In Notecards a guided tour is a series of nodes linearly linked together using links of type ‘GuidedTour’. The author uses a graph-based editor to construct the tour and add it to the system. At each point of the tour the user has access to five special options, presented as links, one to take them to the start of the tour, one to take them to the next and previous nodes, another to jump to any point in the tour and the last to reset the tour.

The paper identifies four types of meta-information required for a guided tour:

- Expository text referring to the original network;
- Instructions to the reader on how to interpret the current display;
- Descriptions of the structure of the tour;
- Textual and annotative devices that offset the effects of fragmentation;

This meta-information forms a narration of the tour that acts as a presentation vehicle. In this case the display indicates which order cards should be read in and when and why they referred to each other.

A spine is the primary path through a tour with minimal branching and it is along this path that an author can assure coherence.

Marshal et al (Marshall & Irish, 1989) conclude that a guided tour is a valuable way of presenting a trail (as identified by Bush) through information in a way that it becomes understandable and accessible to a general audience.

2.3.3 Intermedia

Intermedia was developed in 1988 at Brown University (Yankelovich *et al.*, 1988). Essentially a hypermedia system for a University setting, it differed significantly from its peers in that it dealt in lengthy documents and not small sections that could be displayed on screen in a single chunk (like a card). To enable efficient hypermedia in such an environment it became necessary to allow links to be made to anchor points (called blocks) within a document as well as entire documents. In fact Intermedia allows bi-directional links from any block in any document to any other block in any other.

The definition of a block was very flexible, representing the user selection at the point of link creation. Thus a block could be a character, word, sentence, paragraph or indeed an entire text.

A further major distinction between Intermedia and other existing systems of the time was that Intermedia did not store link and block information within the document. Instead maintaining a web of these objects externally to the data. The web and the document were combined at runtime to construct the user view with the Intermedia system maintaining consistency between the data and meta-data.

Intermedia was built on top of an object-oriented application framework with the aim of easy extensibility. It also aided the Intermedia developers in creating a consistent user interface across Intermedia's five main browsers.

2.3.4 Neptune/Hypertext Abstract Machine

Developed at the Tektronix Laboratory in 1986, Neptune was a serious attempt to apply hypertext to the world of CAD, concentrating on versioning and configuration management (Delisle & Schwartz, 1986). The developers identified CAD systems as those that would particularly benefit from hypertext, in that they had to deal with a variety of related data types (e.g. IC layouts, logic descriptions, timing descriptions etc.).

Their original argument was that hypertext could provide an excellent storage mechanism for all this information.

Neptune was designed as a layered architecture, based on a transaction based server known as the Hypertext Abstract Machine (HAM). This presents a generalised storage model for nodes and links and presents distributed access across a network, synchronising multi-user activity and providing transaction-based crash recovery.

An application layer is then built on top of the HAM and an interface layer on top of that to create a new hypertext application. The HAM doesn't care about the contents of nodes (considered at this low level to be simply binary data). It defines operations for creating, deleting and updating nodes and links and maintains a complete version history of this web (the hypergraph). At runtime it can provide access to the hypergraph at any level of this version history.

Like Intermedia, links can be made to offsets within files. They can also refer to specific versions of nodes within the hypergraph or can refer to the latest version. Application layers can add unrestricted numbers of attribute/value pairs to any link or node, the lists of which are managed by the HAM. Semantic understanding is therefore not needed in the HAM itself.

Neptune provides three main browsers. A Graph Browser to view a sub-graph of the hypergraph in a pictorial way. A Document Browser allows users to view hierarchical

documents in a sensible manner and a Node Browser allows the viewing of single nodes within the system.

Specialist commands are available within the system to allow Neptune to function as a practical CAD system. For example the ‘linearizeGraph’ function flattens a hierarchical document so that it is suitable for producing hard copies for distribution.

Neptune’s version control did not stretch to branching, a feature that the authors acknowledged was important in a real world setting.

The functioning of the HAM was further explained in the 1988 Communications of the ACM (Campbell & Goodmann, 1988). The HAM offers seven categories of operations that can be applied to a variety of objects:

- Create operations. These create new HAM objects, each object is given a unique identifier and is also given a creation time (the first entry in its version tree);
- Delete operations mark objects as deleted, but the objects themselves remain as historical entities. The last version entry being the time of deletion;
- Destroy operations free all the space previously required for an object;
- Change operations modify data within an existing object again making an entry in that objects version history;
- Get operations retrieve data from existing objects. The operation takes both a unique identifier and a version time for the object to be retrieved;
- Filter operations selectively retrieve information from a given graph. It takes a predicate, a version time and a list of attributes and returns objects that fulfil the appropriate conditions for the specified version time;

In addition there are several operations that do not quite fit into these categories. These include string-searching functions, context merging algorithms and transaction management.

The team behind the HAM hoped that it represented a first step toward a standard terminology and storage model that would form the basis for future hypermedia systems.

2.4 Issues for Second Generation Systems

In 1988 Frank Halasz, one of the creators of Notecards, drew a line under the successes of these earlier works and proposed ‘Seven issues for the next generation of Hypermedia Systems’ (Halasz, 1988). He describes NLS/Augment and ZOG as first generation systems, originally mainframe based, with support for large teams of collaborating knowledge workers. He also defines second generation systems such as Notecards, Intermedia,

KMS and Neptune as similar in concept but with more advanced user interfaces, dealing with media other then text and primarily designed for single users or small workgroups.

The issues he proposes were based on the areas where he thought Notecards (and the other second-generation systems) showed weaknesses. He believed that they represented an agenda for hypermedia research for the next generation. These issues were:

2.4.1 Search and Query in a Hypermedia Network

Halasz recognised navigational access of data to be the defining feature of hypertext, however he claimed that experience with Notecards had indicated that navigational access alone was not enough. He claims that effective access to the data is only achieved when query-based access is added to navigational access.

He proposes two classes of query/search mechanisms: content search and structure search. In content search all the objects in the network are considered separately and are examined independently for a match to a given query. In structure search the hypermedia network is examined for subsets of objects that match a given pattern. He suggests that a standard language is needed in which to formulate structural queries, and that work must be undertaken to develop engines to process these queries quickly and efficiently.

Halasz also suggests that, once developed, search and query facilities will become critical parts of a hypermedia system, acting as filters so that users can describe the information that interests them and the network can be displayed accordingly.

2.4.2 Composites - Augmenting the Basic Node and Link Model

In Notecards Halasz had noticed many attempts to utilise existing mechanisms to implement the composition of individual nodes (cards) and link into higher level entities. For example a ‘head card’ used to gather a group of associated cards together. He suggests an important step in hypertext design would be to add composition as a primitive construct to the basic hypermedia model. This enables structures to be constructed and referenced in the hypermedia system on a par with other primitives.

He also identifies a number of problems and issues regarding composites;

- Can a given node be included in more then one composite?
- Do links refer to a node, or a node in the context of a given composite?
- How does one handle the versioning of composite nodes? Does a new version of the node result in a new version of the composite?

- Should composites be implemented using typed links or is a different mechanism necessary?

2.4.3 Virtual Structures for Dealing with Changing Information

A Virtual structure is one that does not exist statically in the structure server, instead the concept or conditions for the structure are stored and it is only instantiated when needed by the system or a user. Otherwise it is identical to a real structure and can be treated identically by all external programs and people. Halasz identifies the need for virtual structures having watched how people construct Notecard sub-networks.

He realised that they constantly revised the structure of the network as it was developing and that they had no real idea of the structure until the network was almost complete. He envisaged virtual structures of being of use when designing such networks, they could represent concepts not yet finalised in reality. For example a virtual node could represent the supporting evidence of a claim, even when that evidence had not yet been found or added to the system. The system resolves these at runtime as they become exposed to the user.

He adds that coupled with a query mechanism and composite structures, virtual structures become even more powerful. For example a virtual link could associate a real source anchor with a query which would be resolved at runtime (e.g. link to evidence if it exists, else link to another node, perhaps explaining the absence of that evidence).

Halasz argues that virtual structures allow hypermedia systems to adapt to changing information in a way that is not possible with the current static models.

2.4.4 Computation in (over) Hypermedia Networks

Hypermedia systems were typically passive systems, in that they allowed the retrieval and storage of hypermedia structures and data but did not actually process or analyse that data. Halasz recognised the importance of active computational engines and commented on their use externally with Notecards. He draws a parallel between knowledge based systems, which involve AI processing of the data, and hypermedia systems.

Acknowledging the generality of the external computational engine approach, Halasz comments that it would be much more efficient if the computational engine were built into the hypermedia system itself. He concludes that whichever approach is taken the hypermedia system of the future must take into account the integration of hypermedia with computation.

2.4.5 Versioning

The Notecards hypermedia system did not support versioning, only one copy of each card was maintained with changes written directly to it. Yet Halasz records several requests for a versioning system and notes that one is almost essential for some tasks (his example is software configuration). Thus he identifies versioning as an important feature that hypermedia systems should support.

He identifies two possible levels of versioning. The first is done at the lower level by keeping a complete version history for all the objects in the system. Whenever an object is changed its version history is updated. The second involves considering the hypermedia network as a whole, in this case an author may decide to collect several changes into one version change.

Another issue is that an objects version history may branch, one object evolving into two or more with the same original 'parent'. Dealing with branched version histories raises some interesting questions regarding references between objects. An anchor can now be a particular version of an object or the latest version of that object. In a branched version environment what constitutes the latest version is not entirely obvious.

Composites add the complication of how a sub-objects versioning (and branching) should propagate to the composite(s) it is contained within.

2.4.6 Support for Collaborative Work

Halasz identifies collaborative work as one of the most important and under-supported feature of a hypermedia system. He identifies two areas of collaborative systems in need of attention. The first involves the actual mechanics of a system where multiple authors are allowed access to the same area of data. The second involves the social interactions between people in the process of collaborative work.

The mechanics of a collaborative system essentially involve the locking of data while a user has access. Typically, better locking has been achieved through increased granularity of the system, e.g. locking sections of a document rather then the entire node. Halasz points out the further granularity can be obtained by classing different types of access. For example one user could still modify links within a document while another user was actually editing the text.

Support for social interactions between authors is less supported. This social support includes early notification of important events (such as the locking of some data). Halasz

describes how previous work has identified three types of collaboration. Substantive activity with involves the actual creation of data, annotative activities involve annotating the substantive work using comments, questions etc. The last type of collaborative activity is the least supportive and comes under the banner of social interaction. It is the procedural activities evolved in collaboration, e.g. discussions, decisions and other actions focusing of the use of the collaborative system.

Halasz concludes that future systems need to concentrate not only on finer locking and notification mechanisms and systems but also on improving the social interactions between collaborators.

2.4.7 Extensibility and Tailorability

The last of the seven issues regards the difficulty hypermedia systems having in balancing their generic nature with the specific tasks they are prepared to do in the real world. Halasz regards the generic nature of the hypermedia system as a problem to users as each is required to undertake a major design task before using the system.

To overcome the lack of support for specific tasks, Halasz believes that future hypermedia systems need to be extensible and tailorable, to enable them to cope with specific problems more effectively and also those that lie outside the domains for which the systems were originally designed.

He identifies the use of interpreted languages as being of particular value and draws upon experience with GNU Emacs to show how a language such as LISP can be extended to deal with a particular domain, enhancing the effectiveness of the tools in that domain considerably. The key is that such extensions should be available to every user not only those who are technically trained with the system.

He concludes his paper with a call to the community to address these seven issues and produce the next generation of hypermedia systems.

2.5 Hypermedia Models

Towards the end of the 1980's it was realised by hypertext researchers that a major failing of the current hypertext systems was their inability to inter-operate. They could neither interact at a system level nor exchange data easily. To start toward a solution to this problem several Hypermedia models were proposed. These models were supposed to provide a common vocabulary for hypertext discussions in the coming years and act as a basis for emerging hypertext metrics. In this section we shall take a look at three of the better known proposals.

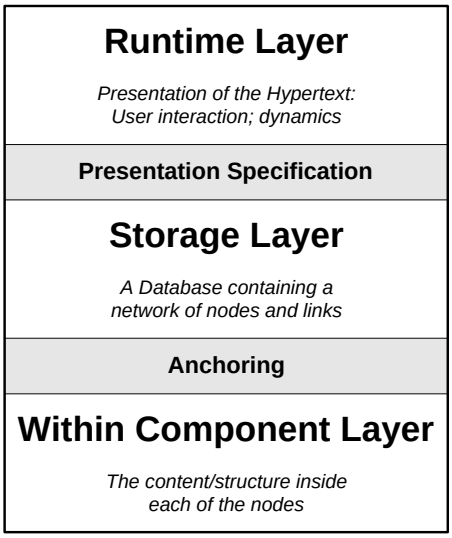

Figure 2.1: Dexter Hypertext Model

2.5.1 The Dexter Model

In December 1990 a workshop (Moline *et al.*, 1990) was held on hypertext standardisation at which various models of hypertext were discussed. One of the results of this workshop was the 'Dexter Reference Model' (Halasz & Schwartz, 1994) the goal of which was to allow easy comparison between systems and thus work towards interoperability standards.

The model divides systems into three layers. The Runtime Layer contains all the facilities a user requires for constructing and browsing a hypertext. The Storage Layer represents the actual structures of the hypertext (e.g. links and nodes). The Within-Component Layer represents the content of the particular node or document.

There are also two interfaces within the model. The Presentation Specifications Interface lies between the Runtime and Storage layers. It represents information on how the runtime layer is to process (represent) the objects in the storage layer. For example whether a viewer should open a file for viewing or editing. The Anchoring Interface is a mechanism for addressing locations or items within the content of an individual component. This maintains a clean separation between the Storage and Within Component layers.

The Within Component layer is responsible for maintaining the position of the anchors within the various components (which remain opaque to the storage layer). While the storage layer deals with maintaining anchors within the node link structure.

The Within Component Layer is purposefully not elaborated on in the model. This is because of the range of possible contents that can be included as components. The

model assumes that other reference models will be used that are specific to a particular application or data type.

The Dexter model was a major step forward, both technically in the sense that it allowed for n-ary links and composite nodes (no system at the time supported both these features) and also as it represented a major collaboration between hypertext researchers and institutions. It's impact on interoperability work, the limitations identified and the ways in which it has been extended are discussed in Section 2.9.

2.5.2 The Amsterdam Model

One of the concerns voiced about the Dexter Model was the lack of consideration it showed for emerging multimedia systems. The Amsterdam Hypermedia Model (AHM), developed by Lynda Hardman and Dick Bulterman at CWI (Hardman *et al.*, 1994), attempted to marry the hyperstructures of the Dexter model with these new systems.

One of the biggest problems they came across was deciding on how a multimedia presentation ordered itself in Dexter terms. There are two aspects to this: *Collection*, where several items are presented together as one, and *Synchronisation* where those items are ordered in a presentation relative to one another. The authors suggest three Dexter based possibilities:

1. *Hidden Structures*: All the information is placed in the content, i.e. inside Dexter's within-component layer. The problem with this is that it does not scale very well to more complex multimedia presentations.
2. *Separate Structures*: Each multimedia item is defined as a separate block. This makes all the structure transparent but synchronisation must be done explicitly and can become a big issue for large presentations.
3. *Composite Structures*: This combines the two approaches above; internalising complex relationships but leaving more basic ones exposed. Although this would work, it requires a specific solution for each presentation and does not provide a uniform solution.

The AHM was produced by combining the Dexter model with the CWI Multimedia Interchange Format (CMIF) multimedia model and adding extensions to cover what was left. It is based on atomic and composite components as shown in Figure 2.2.

In the AHM the presentation information in atomic components has been expanded to include temporal and higher level presentation attributes. In contrast the composite

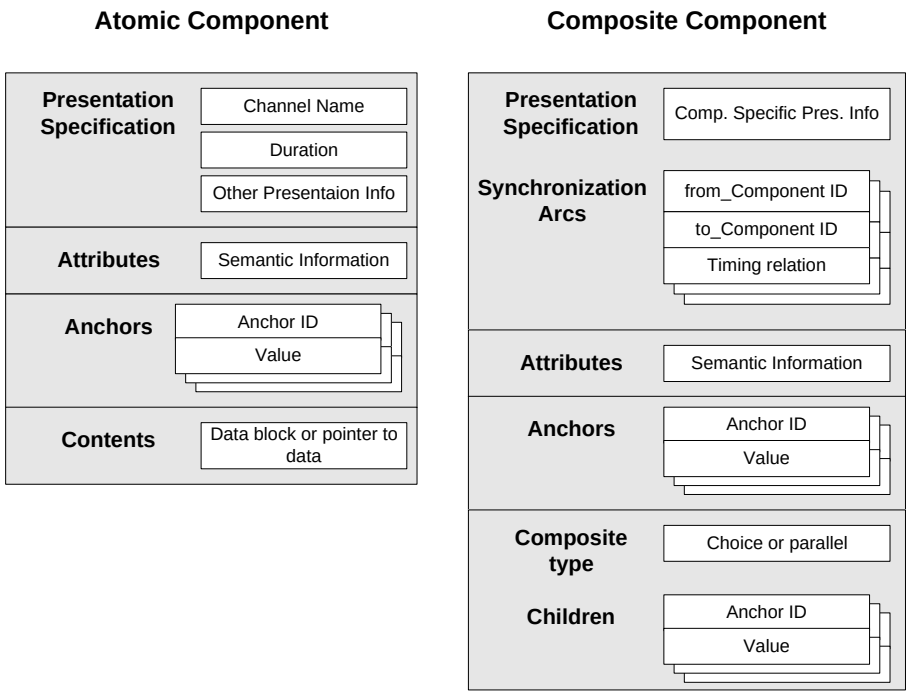

Figure 2.2: Amsterdam Hypermedia Model Atomic and Composite Components

component contains no temporal information as this can all be calculated from the components it contains. Instead it contains collections of *Synchronisation Arcs*, structures that define fine grained relative ordering information about the contained components.

The model also defines *Channels*. These are abstract devices for defining global presentation attributes. By abstracting this information out of components the AHM allows the same document to be presented in different ways simply by respecifying which channel to use for a particular component.

Lastly the AHM has a notion of anchor context. The *Context* object is a composite that contains a collection of components that are effected by a linking operation. Contexts allow specific display options to be associated with each link; in particular the source component can be retained or replaced, enabling a finer granularity of control over the presentation.

The AHM was implemented in the form of the CMIF editor (CMIFed) (Hardman & Bulterman, 1995) which allowed users to construct presentations using the AHM model. The authors found that although the model coped well with the users' requirements, a lot of authoring effort was still required on their behalf to produce a complex multimedia presentation.

2.5.3 The Trellis Model

While the Dexter and Hypermedia models attempt to provide a framework against which to compare existing systems and aid in the creation of new ones, other researchers have pursued models that aid the formal analysis of hypertexts. The Trellis hypertext model (Stotts & Furuta, 1989) (Furuta & Stotts, 1990b) was developed by P. David Stotts and Richard Furuta at the University of Maryland. Hypertext is often depicted as a directed graph of nodes and connections. Trellis extends this notion to Petri nets. A Petri net permits the specification of a hypertext's *browsing semantics*, attaching meaning to the connections between nodes.

This Petri net structure can be viewed in two ways. Firstly it can appear as a formal language; describing a set of strings of symbols where each symbol represents a transition in the net. Secondly it can be thought of as an automata where the Petri net represents a state transition system. This means that semantics of the Petri net are defined mathematically as opposed to residing in any code that manipulates a directed graph.

A hypertext is built up of a Petri net structure, a set of document contents, a set of components (windows, buttons etc.) and collection of mappings between them.

The Trellis team point out that the model could be used to provide a basis for solving several hypertext problems:

1. *Display Parameters*: Certain characteristics of the display can be calculated, e.g. the maximum number of simultaneously required windows;
2. *Concurrent Browsing Paths and Synchronisation*: The model lends itself naturally to multiple browsing paths, in addition an author can specify that several paths are concurrent and also that they must be synchronised at certain points;
3. *Node Reachability and Unreachability*: It is possibly to verify that all the nodes within the hypertext can be reached and also discover which nodes can be reached from different starting points (this can vary according to access control and tailored hypertext semantics);

Using the Trellis model as their formalisation, the Trellis team extended their work to describe a hypertext 'meta-structure' that can be used as a reference model in a similar way to Dexter and AHM. They called this the Trellis hypertext reference model, or Trellis r-model (Furuta & Stotts, 1990a).

The r-model is separated into five levels:

1. *Abstract Component Level*: This contains the components that will be associated together to form the hypertext;
2. *Abstract Hypertext Level*: This contains the connections between the components of the Abstract Component Level;
3. *Concrete Context Level*: This contains the mapping from the hypertext's abstract representation (the two levels described above) to its physical representation and presents a physically based description of the hypertext;
4. *Concrete Hypertext Level*: This level maps the concrete representations generated in the layer described above into a set of windows for display;
5. *Visible Hypertext Level*: This level represents the display and interaction of the hypertext with one or more users;

It is important to realise that the r-model does not enforce any particular data or link model. These distinctions are recognised by the model but specific details are encapsulated within the Abstract Levels. Likewise content and presentation is modeled separately from this structure and itself is hidden within the Concrete Levels.

Thus the r-model is less operational than Dexter. Rather than being an abstract machine it is a functional organisation showing relationships between elements in the various different layers, that might be used to serve as a high-level design guide for system implementers.

2.6 Open Systems

At the beginning of the 1990's it was realised that the closed systems of previous years would not be sufficient in the future, particularly for industrial scenarios where companies wished to continue using existing data and products (Davis *et al.*, 1992). In particular it was thought that hypermedia functionality should be available to all applications across the desktop, this meant a complete separation of link information from content. This approach gave birth to *Open Hypermedia Systems*.

By separating out the link structures, Open Hypermedia Systems (OHSs) gain the capability to process these structures more easily. This allows advanced queries to be made across the hyperstructure, enables systems to maintain link consistency and provides for more powerful linking functionality within individual systems.

2.6.1 Microcosm

In 1990 researchers at the University of Southampton identified four problems common to the hypertext systems of the time and designed a system, Microcosm, to overcome

them (Fountain *et al.*, 1990).

1. *Authoring effort required*; Documents become available far faster then they can be converted for a hypertext system. This can limit the data that a user has access to.
2. *Hypertext systems are closed*; Usually run as stand-alone systems that will not communicate with other software packages. This results in a lack of extensibility that means it can be difficult to cope with new data formats.
3. *Proprietary document formats*; it is not possible to take documents present in one system and use them in another, for example documents written in a proprietary word processor would have to be reformatted for the hypertext system. This actually makes the 'Authoring' problem worse.
4. *Problems with read-only media*; Many hypertext systems use tags or pointers within the data to represent links. When that data is read only (e.g. a CDROM) then this is not possible unless the links are already on the medium, in which case they cannot be updated or removed.

Microcosm was designed with a strict set of guiding principles:

- *No distinction between author and user*; All users can also be authors, and authors can browse their material in the same way as users.
- *Loosely coupled system*; Microcosm is a set of communicating tasks, open in structure. It should be possible to add further components to the system to increase its functionality.
- *Separation of links from data objects*; Information about relationships between documents is separate from the documents themselves. Thus we have two levels of information; data (text, graphics, sound) and meta-data (relationships and links between data).

In 1993 Microcosm had become a mature hypertext system; the separation of links from data allowed Microcosm to let users navigate in a variety of ways (Davis *et al.*, 1992).

- *Specific Links*: Links an object at a specific point in a source document that connects to a particular object in a destination document.
- *Dynamic Links*: As Microcosm uses a database of links and nodes it is possible to dynamically try and create associations between nodes. For example based on a user selection in a source text document the system could do a grep for the same text in other documents.
- *Local Links*: A dynamic link from a given object at any point in a *specific* document that connects to a particular object in a destination document.

- *Generic Links*: A dynamic link from a given object at any position in *any* document that connects to a particular object in a destination document. Thus new documents added to the system may already contain links.
- *Relevance Links*: Alternatively the node set could be pre-indexed to assist this process. In which case it is possible to cluster documents according to their relevance to each other. It is possible for a user to then request all relevant documents (those in the same cluster).

Because of its open nature Microcosm could be used with any viewer. This could be a problem as the viewer may not actually be aware of Microcosm, particularly since at the time hypertext functionality was not common in off the shelf applications.

Microcosm thus works with three types of viewers. Fully aware Microcosm viewers were written explicitly with Microcosm in mind and allow the user to access the full range of Microcosm functionality. Partially aware viewers were applications that were tailorable in some respect, which enables them to become Microcosm aware, although the full range of features may not have been accessible. Unaware viewers were applications that could not be adapted. These could not natively use any Microcosm features but could still act as destination points of link traversal.

Microcosm also incorporated a feature that allowed MS Windows users to use Microcosm from Unaware viewers by communicating through the clipboard. Microcosm was then responsible for reading the information and constructing an appropriate message for the application.

Experience with Microcosm allowed a general reflection on open systems.

Advantages:

- Ability to cope with large numbers of documents with the added value of link following devices.
- Data remains accessible to the applications that originally created it.
- Ability to process the links. E.g. keep separate sets of links for different users and manipulate those sets.
- Authoring effort is reduced via the use of local and generic links
- Allows links to be authored to and from media other than text (e.g. sound, video, etc.)

Disadvantages:

- Position of Anchors. When a user follows a link into a large document the viewer is normally required to move the view to the exact endpoint of the link within that document. This is not possible with unaware viewers, although it is less of a problem in partially aware viewers.
- Editing problem. Because the links are not kept within the documents, if the documents are edited away from the Microcosm system then the links may no longer be accurate. Similarly if the file is deleted then the link will dangle. This is combated in Microcosm be recording a time stamp on all links and nodes, when these change in relation to each other then Microcosm can either try and repair the link or will warn the user (Davis, 1995).

Experiments were also undertaken to see if Microcosm could be distributed (Hill & Hall, 1994) using TCP/IP as a transport mechanism for the messages. These were successful using a peer to peer and client/server approach due to the openness of the system, with only a small performance hit. These experiments included multiple users accessing the same linkbase (database of links) however true collaboration was not supported.

2.6.2 DeVise Hypermedia (DHM)

In 1992 researchers at the University of Aarhus, Denmark decided to build a new Hypertext system based firmly on the Dexter model. Both to prove the validity of that model but also to demonstrate its flaws and suggest improvements. The DeVise project resulted in the first version of DeVise Hypermedia (DHM) (Grønbæk & Trigg, 1994) an open and extensible architecture based around an Object Oriented Database (OODB).

The major revision that the DHM team made to the model was the inclusion of dangling links, i.e. links with zero or one endpoint were allowed in the system. They point out that this is a natural extension of the model, which allows incremental construction of links within the system and lazy updating and garbage collection when endpoints or nodes are deleted from the system.

DHM also explored the notion of link directionality. They identify three main notions:

1. *Semantic direction*: A semantic relationship between one endpoint and another, for example A ‘Supports’ B.
2. *Creation direction*: The direction corresponds to the order of creation of endpoints within the link. The first becomes the source anchor and the second becomes the destination.

3. *Traversal direction*: In this case the user, during the creation process, explicitly gives the link a direction. The link can only be traversed in this direction, relying on a menu of 'back-links' as a mechanism to traverse them in the opposite direction.

The Dexter model allows for the specification of direction in a link but does not explain which of these notions it actually assumes. In DHM they tried to avoid this question by specifying all links as bi-directional at creation and allowing the user to modify this if they so wish.

Another problem they identified with the model concerned the use of Anchors as an abstraction between content and the actual hypermedia structures within the storage layer. The problem is caused when composites are used within the storage layer. Anchors to these composites are not defined, even though both composites and anchors are transparent to the model. DHM includes composites as first class objects alongside nodes and links as defined in the Dexter model but it also extends the notion of composites to include virtual composites, computed components and structured composites.

DHM was extended to include support for collaboration (Grønbæk *et al.*, 1997). This was seen as an important area as many large design projects that could benefit from hypermedia also had requirements in the collaboration area. The DHM researchers identified six modes of collaboration, describing the various ways in which users could utilise the same hypermedia system together.

1. *Separate responsibilities*. The material is split into several parts, each part is manipulated by at most one user (although viewing another's material is possible). This was seen as a very loose form of collaboration mainly consisting of one user making use of material controlled by another.
2. *Turn taking*. This is the same as 'Separate responsibilities' but each part may alternate between users. At most one user is allowed to modify a given part. This requires some support in the system to manage the part to user relationship.
3. *Dynamic exchange*. During a session, users may exchange parts dynamically. Thus a user who wishes to modify a part controlled by another user may request that user to transfer their lock.
4. *Alternative versions*. Different users may develop different versions of the same part. These may be merged later.
5. *Mutual sessions*. Two or more users may work on the same part at the same time, changes being immediately updated on any shared sections of the part. A co-operative commit is required to update the part in the OODB.
6. *Fully synchronous*. This is identical to 'mutual sessions' except that all users have an identical view of the data through a global window.

DHM included support for the first four of these. Other projects in the same department concentrated on the last two outside of DHM itself.

2.6.3 Hyper-G

Hyper-G was an ambitious system that came out of the Graz University of Technology in the early 1990s (Kappe *et al.*, 1993). It was an attempt to write an open hypermedia system for a particular distributed application, in this case a modern University Information System.

It was written to specifically deal with four areas:

1. *Research*. Giving individual scientists immediate access to the results of other researchers, accessing publications, libraries and databases of interest.
2. *Teaching*. Supporting the process of teaching and learning
3. *Administration*. Access and maintenance of relevant legal documents minutes of meetings, rules, records and timetables.
4. *Communication*. Communication and collaboration of groups and individuals (including inter-communication between individuals within the above groups)

The system was envisaged to import data in a variety of ways, from direct entry to allowing users to fax documents that would then be submitted to an optical character recognition (OCR) process. It was also a criterion that existing systems within the University would have to be incorporated. Unlike other distributed information systems being developed at the time including Wide Area Information Server (WAIS), the Gopher project and the World Wide Web (WWW) (described in Section 2.7), the Hyper-G server kept its links separately from the documents.

The designers envisaged that Hyper-G would be built upon a vast store of information (gigabytes of data on hard disk and CDROM) and would be accessed remotely over a Wide Area Network (WAN) using remote terminals.

The actual hypermedia objects stored in the server would be highly structured, belonging at the very least to a hierarchy of nodes. This way the database can ensure the integrity of all links by removing those that become broken (no node will become inaccessible as it belongs to at least one structure). They include ‘Tours’ and ‘Lessons’ as complicated structures created by a teacher for use by students.

Another interesting aspect of Hyper-G is its support for different user modes. A user can log onto the system in one of four modes:

1. *Identified*. The user gives their username and password and enters the system as themselves, thus others in the system can identify the user (i.e. when mail is sent). Users in this mode can also set up preferences and retrieve data previously saved (e.g. their trails).
2. *Semi-identified*. The user selects a pseudonym and a password. The system still knows who the user is but other users do not, although they will see the pseudonym.
3. *Anonymously Identified*. As semi-identified but the system does not record the true identity of the user. This avoids the big brother problem and still enables the user to maintain their own information on the system.
4. *Anonymous*. Used when any user identification is too time-consuming for the job (e.g. public terminals at museums) No user preferences can be stored and access rights are set up per terminal.

Hyper-G was also forward thinking in regards to language. The system was designed to contain primitive translations between languages and to convert queries using these translations to search a variety of documents in a variety of languages, whatever the language of the original query. It was imagined that these kinds of facilities could be added (experimentally) at a later date.

2.6.4 SEPIA

In 1990 researchers at GMD-IPSI Darmstadt, Germany were beginning work on a new Hypertext System designed to give extra support to authors. The system was known as SEPIA (Structured Elicitation and Processing of Ideas for Authoring). The group decided that the storage layer of the system was best built on top of an object oriented database and that they needed to define a data model that could be implemented in that database with software to interface with it. They called this type of software a Hypermedia Engine and named their implementation HyperBase (Schutt & Streitz, 1990).

The goal was to keep HyperBase application independent with the aim of making it open enough to cope with different data types in the future. Because of these application dependant semantics there are the following side effects:

- The applications are responsible for the interpretation of a nodes contents, they simply store a Binary Large Object (BLOB)
- Definition of point to point links must be done at the application layer, as only the application understands how to reference into the nodes.

- Although the hypermedia engine stores versioning information about the various hypermedia objects in the system, it is the application that must deal with versioning at the node level as only the application understands what actually constitutes a version of a particular document type.

Hyperbase was built with the intention of describing a hypertext data model that could be used for common storage and as a basis for the development of a Hypertext Query Language (HTQL). The Hyperbase system was built around a data model for Navigational Hypertext. This model was formed out of five different objects:

1. *HB_Nodes* : Represent some content (i.e. a document) and the history of that content
2. *HB_Links* : Connect two HB_Node objects
3. *HB_Composite_Objects*: Contains sub-objects and a history
4. *HB_Attributes* : Are attached to the other objects, they have a name unique to that object, an attribute value and a history.

The distinction between HB_Links and HB_Composite_Objects is made on two levels. Firstly HB_Links connect exactly two HB_Nodes while HB_Composite_Objects associate many, and secondly HB_Composite_Objects are designed to function as virtual objects as described by Halasz (see Section 2.4.3) as well as statically stored structure.

SEPIA was built on top of HyperBase with the aim of creating a hypermedia system that enabled authors to create hypertexts more easily (Streitz *et al.*, 1992). It acknowledges authoring as a design problem and provides four activity spaces to make authoring easier.

1. *The Content Space*. The design objects and operations of this space are meant to facilitate the development of a domain model. SEPIA uses the structuring facility of hypertext to support ‘idea dumping’. Grouping ideas by topic and linking them to relevant external sources.
2. *The Rhetorical Space*. It is in this space that the author creates the final, reader-oriented, article. This can either be a conventional linear text or a hypertext. The space provides a special ‘construction kit’ that ensures that authors create a text that appears as a coherent entity.
3. *The Planning Space*. Used by the author to externalise their writing plans and to construct an agenda for the authoring activity. It is a meta-space for coordinating the activity in the other three spaces and for controlling the progress of the document.

4. *The Argumentation Space*. This supports the development of an argumentative structure by providing the appropriate design objects and operations. This space was added as argumentation is a crucial cognitive activity that appears in a wide variety of document types and it was felt that it was important to support it directly.

The designer can switch design spaces seamlessly and also bring hypermedia objects across spaces. SEPIA allows automatic transfer of design objects, the reuse of specific design objects and also the indication and control of references across activity spaces.

SEPIA also supports multiple authors working on the same set of design spaces. In this case their browsers are loosely coupled and users are made aware of each other via a list of all concurrent users displayed on the screen, highlighting of objects locked by other users and also a relaxed WYSIWYG view.

It also supports a tightly coupled mode where all coupled browsers show a WYSIWYG view of the content. In addition to the functionality of the loosely coupled mode, browsers also receive scrolling and resizing events from other users.

To support meta-communication SEPIA implements a channel (audio/video) between users and allows gesturing using concurrent telepointing.

SEPIA was extended to include an advanced version server known as CoVer (The Contextual Version Server) (Haake, 1992). The group at GMD realised that for a version system to be useable the effort spent on version management should be outweighed by the benefits and this meant automating the system as much as possible.

Because the system versions everything silently in the background, the user is free to concentrate on the hypertext, while still having access to a full range of versioned hypertext objects. They identified five version creation requirements:

1. Versioning is an all or nothing approach, versioned and non-versioned documents have to be managed in the same system together.
2. It must be possible to maintain the states of the hypermedia objects. Tracking incremental changes, enabling the system to create many past versions.
3. It must be possible to maintain alternatives. E.g. alternative proposals, or alternative subdocuments.
4. Ability for an author to fix a part of a document and refer to it later.
5. A general versioning service should create versions automatically in the context of tasks (e.g. an original version of a document submitted for a review).

CoVer enhanced the basic version mechanism by adding context information in the form of tasks and annotations. This enables task tracking through the version history

providing meaningful, automatic version creation. For example a task is described and decomposed into several sub-tasks. Now when a document is created in the context of that task it can be ‘frozen’ by the version server at each of the points separating the sub-tasks.

2.6.5 Multicard

In 1992 researchers at Euroclid, France noticed that there were two distinct kinds of hypermedia system evolving, both easily describable using the Dexter model.

The first focused largely on the Runtime and Presentation layers of the Dexter model, concerning themselves with the management of hypertext objects and their storage. These systems usually offered basic document processing functionality with no regards to standards or sophistication.

The second breed of system concerned themselves with the Within Component layer of the Dexter model, developing hypermedia extensions to existing content-based applications (e.g. CAD systems). Typically however these provided only an elementary linking facility between two endpoints in the content data. What they decided was lacking was a clear specification of the interface between these two approaches; a specification and implementation of the Anchoring interface described in Dexter.

They presented Multicard, a system that unifies these two approaches by offering the basic set of hypermedia tools accessible via a standard interface known as the M2000 protocol (Rizk & Sauter, 1992).

Multicard presents a familiar set of hypermedia objects (nodes, anchors, groups and links) but links are treated rather differently. In Multicard links are viewed as event or message channels between objects. A variety of messages can be sent through the link including the standard activate request which causes the ‘following’ of a link and the opening of an associated object. The team felt that this approach allowed scripting languages to more easily reconfigure the behaviour of the system.

M2000 editors can conform to M2000 at a variety of different levels. The most basic support includes only two requests (Open and Close node) and returns an error message for all other requests. Multicard does not define any user interface features for the editors, it merely provides the mechanism for them to communicate with the Multicard system.

The protocol itself devolves into four basic areas:

1. Requests concerning nodes. E.g. open, close, print etc.
2. Requests concerning anchors. E.g. create, delete etc.

3. Requests concerning editable objects. E.g. create, cut/paste, select etc.
4. Requests concerning menu management. These allow scripts to modify the menus on the M2000 editors when in reader/author mode. E.g. add, delete, activate etc.

M2000 also provides custom messages by providing a basic operation that takes an application dependant string to be processed as a series of commands.

2.6.6 Chimera

In 1994 researchers at the University of California, Irvine, combined hypertext technology with software development environments (SDEs) and created the Chimera hypertext system (Anderson *et al.*, 1994). Chimera allows developers to freely associate different objects in the system regardless of type, with the aim of both capturing and visualising those relationships.

Heterogeneity was a major concern for the Chimera team as SDEs contain a wide variety of development tools and management software. In addition SDE viewers often supported multiple views of the same objects. Thus Chimera provides anchors that are specialised to particular views (rather than objects) and allows n-ary links across heterogeneous object managers.

The system itself was based on a client server model, where the components communicated via a shared API using a remote procedural call (RPC) mechanism. The server is responsible for maintaining and storing the collections of hypermedia objects (known as a Hyperweb) and serving that structure to the clients. The clients in turn were responsible for presenting the structure to the user, although there was no restriction on how this was done.

Chimera is a useful application of hypertext to an existing real-world problem. It also identifies the problems of integrating with existing management systems, some of which support limited linking technology in parallel to the hypertext system, and also the problem of providing a consistent interface over heterogeneous viewers.

2.7 The World Wide Web

Without doubt the most popular hypertext system is the World Wide Web (WWW) created by Tim Berners-Lee while he was working at CERN in Switzerland in 1989 (Berners-Lee *et al.*, 1994). The Web allows users to access distributed documents and navigate between them by following links, actually references embedded in the document currently being viewed.

The power of the Web lies in its simplicity. The Uniform Resource Locator (URL) allows any object on the Internet to be identified via a relatively simple text string. The first half of this string resolves via a Domain Name Service (DNS) lookup to the Internet Protocol (IP) number of a particular machine and the remaining text acts as a local identifier for the web server waiting on that machine (normally a path and file name).

Coupled with the Hypertext Markup Language (HTML) which allows URLs to be embedded in text and images, this creates a basic framework in which to provide hypertext. However, it has been argued that the Web is nothing more then a distributed file system (Nürnberg & Ashman, 1999) and that its hypermedia functionality does not match up with that specified by early pioneers as being fundamental to a hypermedia system (Bush, 1945; Engelbart, 1963; Nelson, 1987).

Common problems include a lack of support for hyperstructures (i.e. all HTML files hold their links in isolation and cannot be easily processed), a lack of support for navigation (only binary, one-way links are supported) and the problem of broken links, where pages are deleted or moved and their references become invalid.

However the fundamental simplicity of the Web has provided a springboard for a vast array of more sophisticated tools that add functionality to the Web as a whole. Search engines index web pages and provide easy, if non-exhaustive, starting points to seek information. Web crawlers trawl over thousands of pages, either compiling, or searching and indexing data. Web servers have been extended to provide for server side processing which enables HTML to be used as a delivery mechanism for non-hypertext applications, such as on line shopping and banking.

The Web has grown to become much more that a basic hypertext system, it is a cultural phenomenon that forms a cornerstone of information delivery over the Internet on a par socially with television and radio; one that promises to do much more in the future. It is perhaps ironic then, that it still does not fully achieve the goals set down by those early pioneers.

2.8 Movement within Information Spaces

Hypermedia systems are based on the notion that people wish to move through the information stored on their computer systems. However the methods of movement, orientation and travel (collectively referred to here as ‘Navigation’) have altered over the years. Transforming from early ideas on linking (as an analogy of associational thought) to more comprehensive structural provision.

In this section we shall look at the different mechanisms that researchers have provided to facilitate movement within their systems.

2.8.1 Linking

Throughout the development of hypermedia, linking has developed as a concept from the simple point to point linking of early systems to more complex, typed n-ary links of modern systems (Young, 1990). These allow more accurate representation of a variety of relationships within information space.

The separation of links from data that occurred in OHSs allowed systems like Microcosm to let users navigate in a variety of ways (see Section 2.6.1).

In addition the advent of hypermedia allowed for the possibility of dynamic linking across a whole range of different document types. This process depends on analyses of the appropriate media into some form of comparable metric, thus media-based navigation (Hirata *et al.*, 1993), is navigation where the query takes the same form as the desired results, e.g. ‘Find images similar to the one I am currently viewing’.

The analyses and indexing of media documents also opens up the possibility of generic media-based links (Lewis *et al.*, 1996). E.g. ‘Link anything that sounds like this song to this document’.

Any form of media-based navigation or content based retrieval depends on a powerful query mechanism. Powerful navigation of any sort depends on this type of mechanism to be available, to supplement the traditional link navigation model (as described by (Halasz, 1988), see Section 2.4.1).

Whilst data query is relatively simple in the case of text and is being explored in media-based navigation systems, Halasz also identified a second important query type, structural query. These queries are based on the structure of the meta-information itself, e.g. ‘Find all links containing three bi-directional anchors’. Several appropriate structure-based query languages exist including SQL and O2-Query (Christophides & Rizk, 1994) but these are difficult for users to use and often require knowledge of the storage back end of a system.

2.8.2 Spaces

Visual display of hypertext structure has always been popular and was present in systems such as Notecards (Halasz, 1988) and Neptune (Delisle & Schwartz, 1986) in the form of graphical browsers. However another form of hypertext can be achieved by promoting

spatial layout to primacy, such as with the VIKI spatial hypertext system (Marshall *et al.*, 1994).

In these systems the actual structure of the information is created spatially. Objects placed near each other are automatically grouped by the system by a process known as spatial parsing. These groups may themselves have structure depending on their layout, e.g. lists, heaps or matrixes.

Spatial hypertext systems not only provide their users with an easy overview of an information space they also assist in the creation of that space, modelling the way in which human beings naturally collate information into associative groups (Marshall & Shipman, 1997). A more detailed description of spatial systems appears in Section 6.2.2.

2.8.3 Context

One of the more interesting concepts in hypermedia, aimed at reducing the complexity of navigation, is the idea of a context. Unfortunately what exactly constitutes context has been the subject of much debate (Dervin, 1997).

In hypermedia research there have been three main views:

1. Source and Destination Context for a Link (Hardman *et al.*, 1993). This is the view that the source and destination of a link should not be a node but rather have context within a node (linking to a specific point in a file rather then just the file itself). E.g. linking to the third minute of a sound file.
2. Contexts as Workspaces (Delisle & Schwartz, 1987). This uses the idea of authoring hypermedia structure in a particular context that cannot be seen from other contexts, providing a user specific workspace. When the changes made are committed the contexts are merged into the true information space.
3. Contexts Altering the View of Information Space (Casanova & Tucherman, 1991). This envisages context as filtering which information the user can currently see. This is used to assist users to find specific information they need by removing irrelevant data from the information presented to them. E.g. Looking at information on an air-conditioning system, either in the context of a user trying to use the unit or in the context of an electrician attempting to repair it.

In many ways the first two views of context have been addressed in modern hypermedia systems. Anchors provide for linking context (point one) while spatial systems or composites provide the functionality associated with workspaces (point two). However

the third notion of context is neglected in many systems, although some work on alternative open hypermedia linkbases (collections of links) has been conducted (Crowder *et al.*, 1997).

Some systems have exploited context successfully, although often with some user cognitive overhead. In particular Computer Integrated Documentation (CID) (Boy, 1991) an adaptive hypermedia system. In CID users try and describe their objective using descriptors. E.g. you are a designer, you are interested in the air conditioner system, and you have little knowledge of the circuitry involved. When the system produces results it allows the user to specify which results they considered a success and which were failures. The system adjusts its internal metrics as a result and as a consequence if the same query is made with a similar context (set of descriptors) then the system is better able to prioritise its results (or even exclude some of them altogether).

Such a system could be said to dynamically adapt the view of the information according to the context of the current user.

2.8.4 Composites

Composites themselves afford us another way of reasoning about the information in a system. If we regard composites as particular types of links (relationships between nodes represented as sets) then we can navigate from node to node by moving between intersecting sets; this is known as Taxonomic Reasoning (Dyke, 1991).

This categoristic treatment of composites also allows for intelligent set-based querying, e.g. give me the set of all the company reports written in Italy.

Together all of these form a rich collection of navigation methods that attempt to make it as easy as possible for a user to move through an information space and home in on the data that they are particularly interested in.

2.9 Interoperability Efforts

Interoperability has been an important goal throughout hypertext history. The creation of standards was mentioned in Frank Halasz's Closing Plenary at Hypertext '91 where it formed one of several technical issues for future systems (Halasz, 1991). Unfortunately as the support for different navigational methods varies from system to system it has proved very difficult to turn into a reality. Despite these difficulties interoperability has long been identified as a requirement for industrial strength hypermedia (Malcolm *et al.*, 1991), and several attempts at rationalising the differences between systems have been made.

2.9.1 Dexter Revisited

The Dexter Hypermedia Reference model (Halasz & Schwartz, 1994) (discussed in Section 2.5.1) represents a firm basis for interoperability but has been criticised for failing to address some of the requirements of large-scale distributed hypermedia (Malcolm *et al.*, 1991). In particular by assuming that the applications making up the within-component layer are known to the system and that they all model their data on the same Dexter storage layer.

In addition experiments with Dexter implementations (Grønbæk & Trigg, 1994) have highlighted problems with the model itself, most noticeably the lack of support for either embedded links (such as within the WWW) or dynamic links (such as Microcosm's generic links). In 1996 Trigg and Grønbæk extended the Dexter model to cope with these (Grønbæk & Trigg, 1996). However other problems remain, including the lack of support for dangling links, incomplete specification of composites and the lack of a notion of context (Malcolm *et al.*, 1991).

2.9.2 Standard Protocols

In 1989 Sun's Link Protocol developed by Amy Pearl (Pearl, 1989) presented a standard protocol for desktop applications to access hypertext functionality. Pearl concludes that 'We hope to see linking, and attendant hypertext capabilities, as much a standard part of the computer desktop as the cutting and pasting of text are today.'

Multicard, developed by Antoine Rizk and Louis Sauter (Rizk & Sauter, 1992), is an OHS that connects to its applications through a published open protocol known as M2000 (discussed in Section 2.6.5).

Unfortunately neither the research community nor commercial organisations took up either of these interfaces. This may have been due to the immaturity of the field at the times they were published (1989 and 1992 respectively). Halasz makes the point in his revisited speech that any standards need to be based on well-articulated models due to the changing nature of the hypertext field which necessitates change within any standard. Perhaps this change overtook these particular protocols.

Work on the requirements of interoperability has been much more recent. In 1996 Østerbye and Wiil presented the Flag Taxonomy of Open Hypermedia Systems (Østerbye & Wiil, 1996). The flag taxonomy (named because of its similarity to the Danish national flag) was a framework within which to compare and classify OHSs. It described each system in terms of the Storage Manager, the Data Model Manager, the Session Manager, the Viewer and the interfaces between them. In 1998 they extended the model to describe

the interactions between separate OHSs and presented the T3 protocol (Østerbye & Wiil, 1998), which enabled intra-application integration (for instance, one application could ask another to display a particular file or node), to augment traditional application/server integration.

2.9.3 Open Hypermedia Protocol

In 1996 Davis and Rizk proposed a basic protocol to enable linking, known as the Open Hypermedia Protocol (OHP) (Davis *et al.*, 1996). The authors had acknowledged a problem with system development within the research community. Because each OHS required its own set of proprietary clients it was difficult to implement either a new OHS (because the researchers were forced to spend a great deal of time writing clients) or a new type of client (as they were then required to write an underlying hypermedia system).

OHP answered this problem by presenting a standard interface between clients and servers. The vision was for existing clients to be reusable across many hypermedia platforms, therefore reducing development times. It was envisaged that software could be written to 'shim' communications of the server format to OHP and then back to the client format and vice versa.

However the protocol was criticised, both for inconsistencies within the protocol definition itself (Anderson *et al.*, 1997) and also for its lack of an underlying data model and architecture (Anderson, 1998). In addition other areas of interoperability have been identified and remained to be addressed, such as the interface between a server and its storage back-end (Goose *et al.*, 1997).

2.10 Summary

In this chapter I have presented a broad range of hypermedia systems, the issues associated with these systems and described how different models and protocols have attempted to achieve interoperability between them.

The most recent of these efforts has been the evolving OHP effort, in which I have played a major role. In the following chapters I will explain how the work involved in the development of OHP has led to a greater understanding of inter and intra system communication and also the information worlds that we are attempting to create and model.

Interoperability has been identified as an important part of a larger research effort into Open Hypermedia (Nürnberg *et al.*, 1998). Along with emerging component-based

systems and the examination of structure and navigation within such systems, it forms a cornerstone of future hypermedia research.

Chapter 3

Development of OHP-Nav

3.1 Introduction

At the first workshop on Open Hypermedia Systems (OHS 1) in Edinburgh at ECHT'94, Antoine Rizk proposed the standardization of a lightweight protocol that would allow a client program to talk to a link server. The benefit that was expected was that the Open Hypermedia Systems Working Group (OHSWG) would be able to start experimenting with interoperability between link servers and reduce the overhead of re-implementing viewers. It was intended that this protocol would be simple enough that a third party application such as Word for Windows could be simply adapted using the built in macro programming facilities. A first draft of the Open Hypertext Protocol (OHP) (Davis *et al.*, 1996) was presented at OHS 2 at Hypertext '96, two subsequent meetings of the OHSWG refined and altered the protocol significantly (Davis *et al.*, 1997).

The original protocol had attempted to capture that common set of navigational 'point and click' actions that are embodied in most traditional hypertext systems. The protocol was intended to represent a common subset of that functionality. At OHS 3.5 in September 1997 the working group decided that the protocol should attempt to become a superset that could express the actions that occur in all structure processing systems. It was thought that we would develop a set of 'hypertext objects' which are stored on structure servers, and a set of interfaces that allow us to manipulate these objects.

It was also envisaged that less well-understood models of hypertext should also be supported, such as Spatial Hypertext (Marshall & Shipman, 1995; Reinert *et al.*, 1999) and Taxonomic Hypertext (Nürnberg *et al.*, 1996; Dyke, 1991; Dyke, 1993) (these other 'domains' are explained more fully in Section 6.2). In this world the original OHP becomes the 'navigational hypertext interface' (a subset of the OHP protocol) referred to from this point onwards as *OHP-Nav* and the objects in the data model (links, nodes etc),

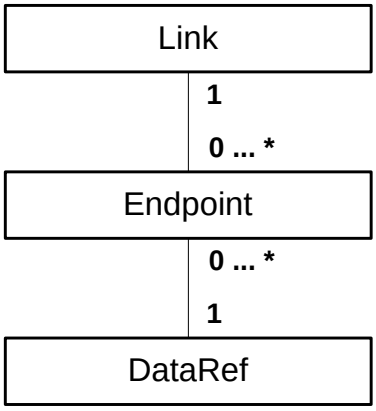

Figure 3.1: Hypertext Data Model

are ‘navigational hypertext objects’ which are a subclass of the set of all objects (although it was envisaged that some of these objects, such as nodes, might be re-useable across other domains).

In this chapter I shall describe the OHP-Nav protocol developed and in particular focus on the data model, architecture and message syntax. I was heavily involved in an early demonstration of OHP-Nav at the Hypertext ’98 conference in Pittsburgh, PA, where several OHP-Nav speaking components showed interoperation between hypermedia systems for the first time. I will describe this demonstration, the system that we developed and finally explore some of the protocol related issues that were raised.

3.1.1 A Common Data Model

The hypertext community has invested much time and effort in attempting to define hypermedia, and probably the most successful result is the Dexter model (Halasz & Schwartz, 1994; Grønbæk & Trigg, 1996) described in section 2.5.1. In this section we define the terminology and model that is assumed by the OHS working group.

This description attempts to be as inclusive as possible in the sense that the model is capable of representing the link models assumed by most existing hypertext systems. However, this model does not attempt to model the systems that have particular features such as transclusions in Xanadu (Nelson, 1987) or other domains such as spatial hypertext (see Section 2.8.2). These systems were expected to design their own interfaces.

Figure 3.1 depicts the basic entities of the hypertext data model, and their relationships.

The model of hypertext assumed consists of servers which hold links, endpoints, datarefs and nodes. Each of these objects have unique identifiers within the system. Servers which manage these objects are known as link servers.

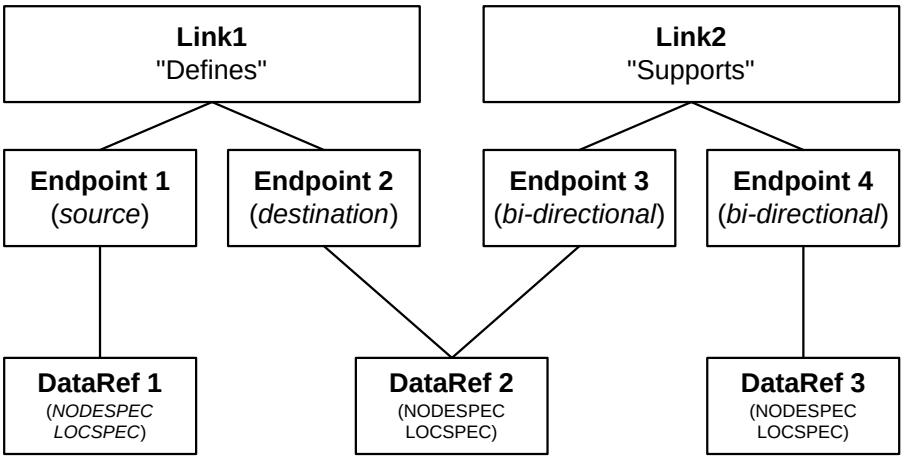

Figure 3.2: Example of Data Model

A link is an object which represents a connection between zero or many endpoints. A link may, or may not, have a type (e.g. ‘defines’). In many systems traversing a link may cause some process to run, as well as, or instead of, causing the focus to move to some point at the end of the link.

An endpoint is an object which holds the attributes of the end of a link. Typically an endpoint will hold a traversable direction which might be source, destination or bi-directional. For example Figure 3.2 shows two links, one of which has source and destination endpoints and the type ‘defines’, and the other which has two bi-directional endpoints and the type ‘supports’. Thus the link ‘supports’ may be traversed from Endpoint4 to Endpoint3 and vice versa, while the link ‘defines’ can only be traversed from Endpoint1 to Endpoint2.

A dataref references a node and defines the point within that node at which the application which shows the data should indicate some kind of persistent selection which will be a hypertext ‘hotspot’. The words ‘anchor’ and ‘button’ were deliberately avoided as those terms have been somewhat overloaded in the past. A dataref therefore consists of a node identifier and a location specifier or LOCSPEC. A dataref may be associated with zero or more endpoints. For example, a user may make a number of datarefs before associating any of them with any endpoint of any link. Also, one dataref might be shared amongst more than one endpoint, and these endpoints might belong to different links.

A node identifier (nodeID) uniquely identifies one node.

A node is a wrapper object which holds the meta-data about some content data including the information about where to find the file or files which make up that content data. This content data is defined by a content specification or CONTENTSPEC.

A location specifier (LOCSPEC) holds the information that defines the position at

which the persistent selection should be displayed as well as any presentation information.

A presentation specifier (or PSPEC) may be associated with any hypertext object and defines how that object should be displayed at run time.

Sometimes a link or dataref may have an associated script specifier (SCRIPTSPEC) which is executed when the link or reference is used.

In the above definitions, the terms CONTENTSPEC, LOCSPEC, SCRIPTSPEC and PSPEC have all been shown in uppercase. This is to signify that their definition is outside the scope of the navigational hypertext protocol and that they are therefore opaque, whereas all terms shown in lowercase are fully defined within the data model. However, some proposals for possible realizations of these opaque objects were created and are detailed below in Section 3.1.5.

If a link server stores the content data as well as the hypertext objects, then it may be referred to as a hyperbase, since it is a database storing the entire set of objects involved in the hypertext.

A link server is expected to provide a core set of link services which involve the creation, retrieval and use of links, endpoints, datarefs, nodes, scripts and presentations. It will also be expected to provide some level of document management, for example, informing the client the name of the file it must load and display in order to complete a link traversal.

Many link servers provide extra services, which assist the user in navigation, such as Microcosm's integrated search engine which creates dynamic links (as explained in Section 2.6.1).

3.1.2 The Assumed Architecture

While developing OHP-Nav it was thought that addressing interoperability not only required some common understanding of the data model but also the underlying architecture should be investigated and agreed upon. This was because a data model usually makes assumptions about architecture and because functionality and behaviour are to a large extent based on the architecture. In many ways later work on OHP dropped this assumption, or at least became far more flexible, preferring a vision in which information systems can communicate regardless of architecture.

In this section I will discuss general assumptions about the underlying architecture of open hypermedia systems made during the development of OHP-Nav. This was never

an attempt to define a reference architecture, although promising work was happening in parallel (Goose *et al.*, 1997; Grønbæk & Wiil, 1997; Wiil & Whitehead, 1997), instead it was thought of as the minimum that was necessary to allow an implementation of the data model described above.

There is a problem that impedes the simplicity of a model which has the application program communicating directly with the link server. In practice all those who have implemented link servers have identified the need for some component on the client side which is present throughout the hypertext session.This has previously been called the 'Runtime' (Goose *et al.*, 1997). the 'Tool Integrator' (Wiil & Leggett, 1996) and in the original version of OHP it was called the 'Communication Protocol Shim' (Davis *et al.*, 1996). In order to prevent any further naming problems the OHSWG agreed to call it the CSF. (This acronym might be said to stand for 'Client Side Functions'; actually it originally stood for 'Client Side Foo', which was the 'stub' name adopted by the group in order to prevent any further argument about the name). Figure 3.3 gives a graphical representation.

The responsibilities of the CSF will always include starting an application (with its data) when required to by the link server (e.g. as the result of traversing a link to a document which must be handled by some application which is not currently running). This function is necessary as in heterogeneous systems it is not always possible for a server application to start a client on a remote machine.

There are many other tasks that might need to be carried out on the client side:

- communicating with a document management system for lightweight clients
- providing client side tools for such activities as joining endpoints to links and datarefs to endpoints
- providing facilities for establishing and maintaining connections to servers
- caching of documents
- user information and client side security
- managing default scripting facilities

For the above reasons we believed that it is necessary that some of the communications between client side applications and link servers were routed through the CSF. The exact routing of the messages is irrelevant from the point of view of the link server, which simply receives the messages and sends replies. However, if we want to allow third party viewers to plug in to a CSF, there will need to be some standardization so that the viewer can know which services to expect the CSF to provide. For example some systems may want the CSF to intercept and handle all requests to produce content so that they can proxy for the document management system.

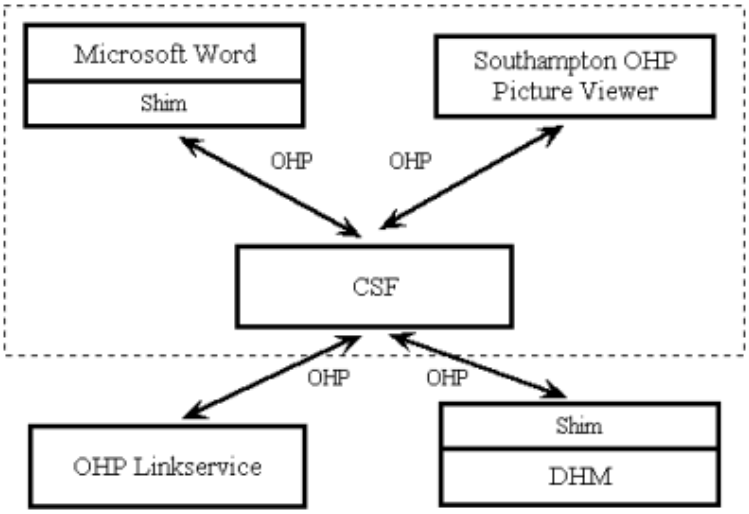

Figure 3.3: Assumed Architecture

Figure 3.3 gives an example of the kind of architecture we were assuming. In this figure there are two client side applications. One is Word, which has been adapted as a OHP aware viewer using its macro language, and one is a special purpose picture viewer which was written to work with OHP. Both these communicate with the link servers via the CSF using the OHP protocol. One of the servers is a native OHP server, whereas the other might be a DHM (DeVise Hypermedia) server, and a shim has been inserted in order to convert the OHP protocol to the DHM native protocol.

Besides the components application program, link server and CSF there is a fourth component, the Document Management System (DMS). We always thought that an additional protocol - a document management protocol (DMP) - should allow standardized communication to document management systems. The Open Document Management API (odm, 1997) could serve as a good example and some of the concepts in the hypertext data model described here, e.g. the definition of a node id as a unique document identifier, follow this proposal.

Figure 3.3 shows the client communicating with multiple servers. At this stage we had not defined whether this is on a ‘one server per session’ basis, or whether we expected the client to be able to send requests to distributed servers, as a web browser talks to multiple servers.

3.1.3 Message Definitions (EBNF vs. XML)

OHP messages consist of text strings. In the following examples the messages are shown formatted on multiple lines for ease of presentation. However, the messages themselves are one continuous stream of ASCII text, unbroken by line feeds.

Originally the protocol was loosely based on the Microcosm message passing format and was defined using an Extended Backus-Naur Form (EBNF), with messages consisting of tags, which were proceeded by a backslash (‘\’) and succeeded by a white space. The characters that followed, up to the next tag or the end of the message were the tag contents. A tags content could be empty. In addition some tags denoted a block of the message with some particular relevance. E.g. \LocSpec and \EndLocSpec. The complete description of OHP-Nav in this form can be found in Appendix A

However, early implementations of the protocol as described above raised several issues regarding the parsing of the tagged ASCII protocol definition. These issues included some inconsistencies in the definition. Among them the naming of tags and the handling of objects. Also, the message header contained insufficient information about the content of the message to allow efficient routing. Most importantly, the message definition was non deterministic which made message parsing difficult.

For these reasons we decided to define an XML conformant document type definition (DTD). As well as updating the protocol definition, XML had the advantage of being a well supported standard which allows us to re-use a variety of existing tools and packages. The complete description of OHP-Nav as a DTD can be found in Appendix B.

3.1.4 Message Header

As opposed to the earlier draft of OHP (Davis *et al.*, 1996), which defines a channel as an identifier for messages, here we defined a dedicated message header. As already mentioned above, OHP-Nav abstracts from the network layer so the header does not include any host, port or other communication specific information.

Every message will have a message header. A message header will contain information about the transaction that both sides of the communication channel may from time to time need to examine. It is not immediately clear that all of this information is essential, but there is a general consensus that parts of this information will, at times, be necessary.

MID a unique identifier for the particular message.
RMID the ID of the message this message is a reply to.
VID current version of the protocol.
SID a system dependant string which uniquely identifies some ‘transaction’ or ‘session’.
UID a system dependant string uniquely identifying the user.
FID the name of the message of which this is a header.

The MID would probably be realised by a time stamp. The RMID allows applications and link servers to keep track of sent and received messages and by that keep state. The

field can be left empty in case a message is sent that is not a reply to a previous message. As OHP was at that stage often being revised and amended a VID field was thought essential. E.g. '2.1.Southampton' would indicate version 2.1 of the OHP protocol with Southampton extensions.

At the time this definition was written the OHSWG had not yet defined what a session was but we envisaged that such a notion would become significant when we attempted to deal with collaboration issues. It was thought important for the protocol to provide basic support and thus we provide the SID and UID.

When early prototypes were being produced it was also discovered that in some cases the type of the message also needed to be provided in the message header. E.g. in the case where the body of a message is encrypted but that message needs to be routed based on its type. For this reason the XML version contains an additional 'function id', or FID, that was always set to the type of the message.

3.1.5 Opaques

There are certain entities within OHP that are treated as opaque (these appear within the protocol in uppercase). This means that their definition is unimportant for the specification of OHP itself although they will need to be understood by the applications using OHP. There are four opaques within the OHP definition, LOCSPEC, PSPEC, CONTENTSPEC and SCRIPTSPEC). We expect a situation where a general definition of an opaque is needed (to allow interoperability) but where the protocol must be flexible enough to allow specialist applications to define their own standards for these opaques and to insert an appropriate byte stream. In order for such a byte stream to be placed within in ASCII protocol, it would be necessary to MIME encode the bytes.

So that an application (or the CSF) can parse all opaques, each opaque is accompanied by a version ID and enclosed within a begin and end tag. Thus a parser can check the version and skip the opaque if it is not one that is understood.

A complete definition for these opaques can be found in Appendix A.

Location Specifications (LOCSPEC)

Locations Specifiers (LOCSPECS) describe a position in a document or piece of media. The definition of the location specifications was quite a controversial issue during the development of the proposal. The two main camps being the 'opaque' people on one side and the 'HyTime' people on the other. The argument towards opaque location specification is that for the link server location specifications should be opaque because the link

server does not actually deal with them but rather wants to handle them as uninterpreted byte blocks. The HyTime (Newcomb *et al.*, 1991) followers on the other hand argue that at some point you actually have to have a specification so that an application is able to locate a specific position within a document's multimedia content. If you have to define it use an existing standard, in order to deal with location specifications in an abstract, platform independent way.

We tried to learn from both viewpoints and concluded that both had valid points. Therefore location specifications are treated as opaque, but a definition was made that should be used within the opaque block if possible.

A number of additional criteria influenced our decision for the definition of location specifiers:

- A full HyTime implementation was considered to put too much onus on the application and also would be rather difficult to implement in a macro-like language. We thus agreed on a subset of location specifications as defined in HyTime;
- The protocol is assumed to develop, thus in the first instance the focus was on establishing the protocol and later on improving and enhancing it;

It should be stressed that the standardisation of location specifications was not directly a part of OHP. This can be compared to the situation with the document management protocol; strictly speaking this should not be part of the definition of OHP. However, if OHP was going to work there had to be some common assumptions about both location specifications and document management functionality. For OHP-Nav we therefore provided these features so that OHP-Nav as a protocol could be implemented.

In actual fact we realised that there were several different kinds of LOCSPEC and although they shared some common attributes they differed in the way in which they specified a location. We named these different specifications LOCS and came up with several possibilities.

Name Space Locations These are used to reference an object by its name. It was envisaged that this would be useful in systems were parts of the data were already named (i.e. a CAD system). In these cases simply recording the name would be sufficient.

DataLoc These are co-ordinate locations, i.e. objects are addressed by their relative position to another object. Data locations are heavily used by OHP as most systems define datarefs by relative positioning, for instance by keeping an offset from the beginning of a text file. In order to know how this dimension specification should

be interpreted, i.e. to get its data type, a quantum tag is defined. The DataLoc follows the HyTime semantics of defining a dimension list. In particular, two markers define beginning and end of a node on an axis thus resulting in (number of axes * 2) markers.

A reverse counter is kept in order to do cross checking of validity of data references. In case the expected object still can not be found an overrun behaviour can be specified; the error can simply be ignored, the markers can be truncated to the last unit on the axis, or an error can be raised.

TreeLoc These can be used for addressing a single object within a tree. The addressing is done such that on each level of the tree an object is selected by its position.

E.g. a list of '1 2 3' applied to an object referencing a book would get the third section of the second chapter of the book (assuming that a book has a structure with chapters and sections).

ScriptLoc OHP does not define a specific query mechanism for addressing locations (as does HyTime with the standard DSSSL Query Language, SDQ). However, by allowing script locations we define a way for those hypertext systems that use scripts to identify and address locations at the client side. Scripts are also treated as opaque to OHP and are dealt with below in Section 3.1.5.

NALoc Another idea borrowed from HyTime is to allow the addressing of data that is currently not accessible. This could be a printed book which is not available on-line, a live radio broadcast or even a person or an organisation. The point is that it might in some cases make sense to reference these nodes without having a application program for them. Instead, a client could display a notification string, stating for example where to find the book on a shelf or in a library. We call this location specification NALoc for 'Not Accessible' location.

Presentation Specifications (PSPEC)

Presentation Specifiers (PSPECS) were always considered an important feature of the protocol, but became increasingly difficult to define. Eventually the OHSWG agreed on a simple PSPEC with the understanding that it might be replaced at a later date. PSPECS were designed to store, and thus re-apply a required presentation for any hypertext object that a client may wish to display. A client might store a PSPEC as a byte stream, which would then later be retrieved and interpreted by the same client. We were aware that strictly speaking presentation styles should not be defined within OHP (Anderson, 1997). However, if we hope to achieve interchangeable client programs, then there must be some standard vocabulary for describing these presentation attributes, perhaps in a similar fashion to style sheets in the web community.

Name the name of this style.

Colour one of several basic assumed colours.
Style one of several basic assumed styles.
Visibility ‘true’ or ‘false’.

Originally the PSPEC was intended only for anchors and therefore a triplet of colour, style and visibility were used, each of which was optional. A more advanced PSPEC was planned that would include appropriate presentation information for the other data objects as well, but it was never finalised.

Content Specifications (CONTENTSPEC)

Within the protocol there was also a need to reference external documents. The specification of their position was called a CONTENTSPEC and, similarly to PSPEC, was defined with the understanding that it would probably be improved at a later date. At the very least it was understood that the CONTENTSPEC would need to reference both local files and distributed resources.

Name the content’s name, typically a file name, a URL or a DMS handle.
Location ‘FileSystem’, ‘Internet’ or ‘DMS’.
Attributes list of name\value pairs.

As far as the link server was concerned the CONTENTSPEC information was opaque. It stores the string, and when the node is required it sends this back. The client side is expected to provide some component which will be able to interpret this string in the cases where the client is asked to display a document. It was envisaged that typically this would be the CSF, which would then have the responsibility of actually getting the document from the file system, the Internet or the document management system that it is using (which may actually be the link server itself if the link server is a hyperbase).

Script Specifications (SCRIPTSPEC)

OHP was hoped to be able to cope with many possible scripting languages, but the OHSWG had no particular ones in mind. Because of this the SCRIPTSPEC definition remains extremely general and is little more than a wrapper for a script.

Language name of the script language, e.g. JavaScript, etc.
Data the actual script.
Attributes list of name\value pairs.

This is a simple definition, designed to allow scripting to be added to an application with little extra complication.

	Create	*Get*	*Update*	*Delete*	*Execute*	*Notification*
Endpoint	X	X	X	X	X	
DataRef	X	X	X	X		
Link	X	X	X	X		
Node	X	X	X	X	X	ClosingNode
Script	X	X	X	X	X	
PSpec	X	X	X	X		
Services						

Table 3.1: OHP Client Message Table

	Definition	*Display*	*Execute*	*Close*
Endpoint	X	X		
DataRef	X	X		
Link	X			
Node	X	X		X
Script	X		X	
PSpec	X			
Services	X			

Table 3.2: OHP Server Message Table

3.1.6 Messages

There are two main classes of messages that we should considered; those sent by the application program (the client) and those sent by the link server. The messages sent by the link server are always answers to messages that have been sent by a client. However, in a co-operative working environment we cannot rule out the possibility that the server sends a message to client B in response to a message from client A (an update message for example).

Tables 3.1 and 3.2 give an overview of the basic messages that form the OHP protocol. The very left-hand column defines the basic entities we have identified, i.e. link, endpoint, dataref, node, script, presentation specification (PSPEC) and services; the top line, i.e. the columns' labels, define the basic functions that can be processed on these entities. The functions are create, get, update and delete. Lists are introduced as a means to allow aggregation of entities. An endpoint table for example would be dealt with in OHP as a list of endpoints.

In addition an 'Error' message might be sent by any component being involved in an OHP session.

I will now present the messages in detail and also give examples or additional comments where necessary to understand the semantics of the protocol.

```
MESSHEADER
"\Subject          CreateNode"
"\NodeName "       <A title that the application may choose to display>
"\MimeType "       <MimeType>
"\PreferredApp "   <The name of the application which we would prefer
                    to use with this data>
"\ContentSpecVID " <ContentSpecVersionID>
"\ContentSpec "
CONTENTSPEC
"\EndContentSpec "
"\Attributes "
{"\Name "          <the attribute's name>
 "\Value "         <its value>}*
"\EndAttributes "
```

Figure 3.4: EBNF Definition for a CreateNode Message

Note that some of the messages might be rarely used, some even never. However, for reasons of consistency and symmetry of the protocol they were defined and kept as part of the standard.

An Example Message

It can be seen from Tables 3.1 and 3.2 that the definition of OHP is rather large and contains many different kinds of message. These are all detailed fully in Appendix A and Appendix B in the respective notations used for the definitions. Below is as example definition and resulting message for the operation 'CreateNode' expressed firstly in EBNF and then the later XML form.

This message will create a new node in the link server. It is perhaps the one message related to nodes that might be used by an application program, in the case where a client side program had just loaded or created some new data content and wished to register it with the link server.

Figure 3.4 and figure 3.5 show a CreateNode using the EBNF notation. Notice the opaque CONTENTSPEC that has been expanded using the recommendations above.

This might result in the following example message, attempting to create a new node based on a local file. Notice the first five tags, comprising the message header.

When the protocol moved into the prototyping stage and XML became the syntax some minor changes were introduced. As well as the FID mentioned above that appears in the header, the CONTENTSPEC has been simplified to just a URL. Figure 3.6 and figure 3.7 show the same message defined and written using the XML notation.

```
\MID              70873498573
\RMID
\VID              OHP1.2
\SID              23
\UID              1
\Subject          CreateNode
\NodeName         Thesis Document
\MimeType         text/html
\PreferredApp     Mozilla
\ContentSpecVID   CS1.0
\ContentSpec
   \Name       H:\THESIS\THESIS.HTM
   \Location   FileSystem
   \Attributes
      \Name    owner
      \Value   dem
   \EndAttributes
\EndContentSpec "
\Attributes
   \Name          timecreated
   \Value         2000-07-06 04:12:15.0000
\EndAttributes
```

Figure 3.5: EBNF Example for a CreateNode Message

```
<!-- CREATENODE -->
<!ELEMENT createnode   (node)>

<!-- NODE -->
<!ELEMENT node     (nodeid, nodename, mimetype, preferredapp,
                        contentspec, attributes)>

<!-- CONTENTSPEC -->
<!ELEMENT contentspec   (version, url, attributes)>

<!-- FIELDS -->
<!ELEMENT version   (#PCDATA)>
<!ELEMENT url   (#PCDATA)>
<!ELEMENT nodeid    (#PCDATA)>
<!ELEMENT nodename   (#PCDATA)>
<!ELEMENT mimetype   (#PCDATA)>
<!ELEMENT preferredapp   (#PCDATA)>

<!-- ATTRIBUTES -->
<!ELEMENT attributes  (attribute*)>
<!ELEMENT attribute   (name, value)>
```

Figure 3.6: XML DTD Extract for a CreateNode Message

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE OHPNav SYSTEM "ohpnavxmlJune10.dtd">
<OHPNav>
   <messageheader>
      <mid>70873498573</mid>
      <rid></rid>
      <vid>OHP1.2</vid>
      <sid>23</sid>
      <uid>1</uid>
      <fid>createnode</fid>
   </messageheader>
   <createnode>
      <node>
         <nodeid></nodeid>
         <nodename>Thesis Document</nodename>
         <mimetype>text/html</mimetype>
         <preferredapp>Mozilla</preferredapp>
         <contentspec>
            <version>cs1.0</version>
            <url>file://H:/THESIS/THESIS.HTM</url>
            <attributes>
               <attribute>
                  <name>owner</name>
                  <value>dem</value>
               </attribute>
            </attributes>
         </contentspec>
         <attributes>
            <attribute>
               <name>timecreated</name>
               <value>2000-07-06 04:12:15.0000</value>
            </attribute>
         </attributes>
      </node>
   </createnode>
</OHPNav>
```

Figure 3.7: XML Example of CreateNode Message

3.2 Demonstration at HT’98

At OHS 3.5 in September 1997 the working group also decided that a demonstration of the OHP protocol would be a good goal for the Hypertext ’98 conference, to be held in Pittsburgh, PA the following year.

Two systems were developed for the demonstration. One at the University of Aarhus, Denmark, and the other developed by the author, Jon Griffiths and Dave Chandler at the Multimedia Research Group (MMRG) at the University of Southampton.

The Danish system was based on an Emacs implementation. A wrapper was written for Emacs that communicated directly with the Danish OHP server. This wrapper handled all aspects of hypermedia, including link creation across applications and also application launching.

The Southampton system in contrast used a CSF to encapsulate hypermedia functionality common to all client side components. The CSF acted as a nexus for all client traffic and was responsible for launching applications and maintaining a link editor that could create links.

3.2.1 The Southampton CSF

The system developed in Southampton comprised of an OHP-Nav aware image viewer, a link editor, the CSF itself and the link server, the relationship between these is shown in Figure 3.8. All communication is done through TCP/IP sockets with the XML syntax described in Appendix B, using a 19 byte leading integer to indicate the length of the message stream.

The image viewer is an application that supports file system and Internet retrieval of documents, the following of links and the creation of new endpoints. It does not have a cache or any editing facilities. The link server uses JDBC into an Access Database, storing OHP-Nav objects (endpoints, nodes etc).

The CSF implemented uses a toast-rack architecture of slot in communication mechanisms to talk a variety of network protocols, either to client or server sides. For the demonstration at Hypertext ’98 a TCP/IP sockets mechanism was implemented. These mechanisms were independent of the OHP-Nav and thus could be reused for other protocols such as the CSF protocol described in Section 3.2.2.

3.2.2 Findings and Issues

In completing the specifications and prototype implementations the OHSWG encountered a number of issues that needed to be resolved.

OHP-Nav Traffic

We found that using the last OHP specification (Davis *et al.*, 1997) the number of messages being sent was disproportionately high in comparison with the task being carried out. This was because the protocol had been written to handle ID values (so as to mimic the ‘pass by reference’ structure of component frameworks such as CORBA (OMG, 1991)). This means that ID values were continually having to be resolved and re-resolved.

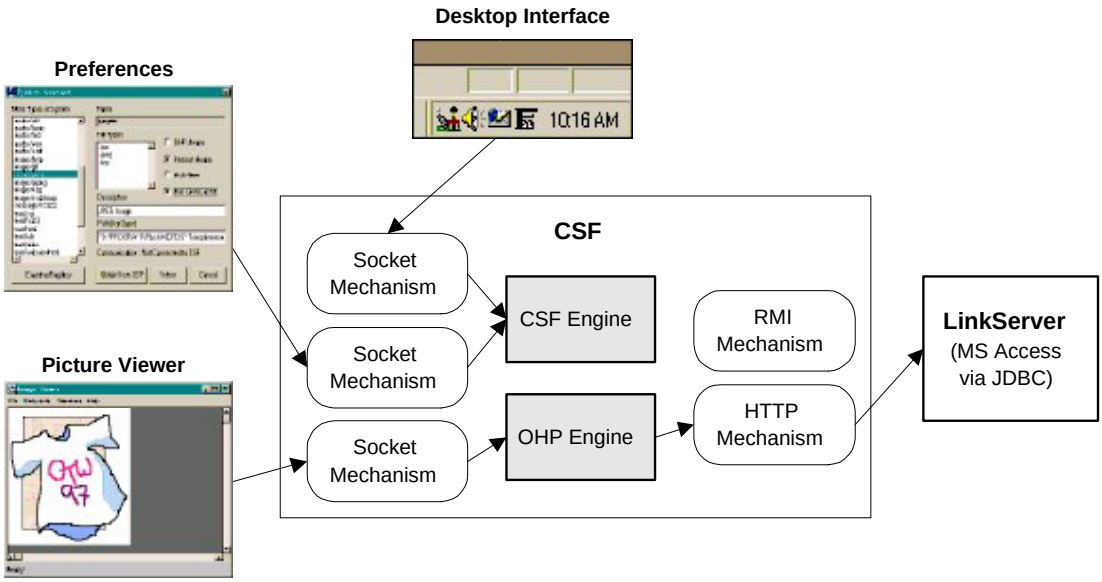

Figure 3.8: The Southampton Foo Configuration

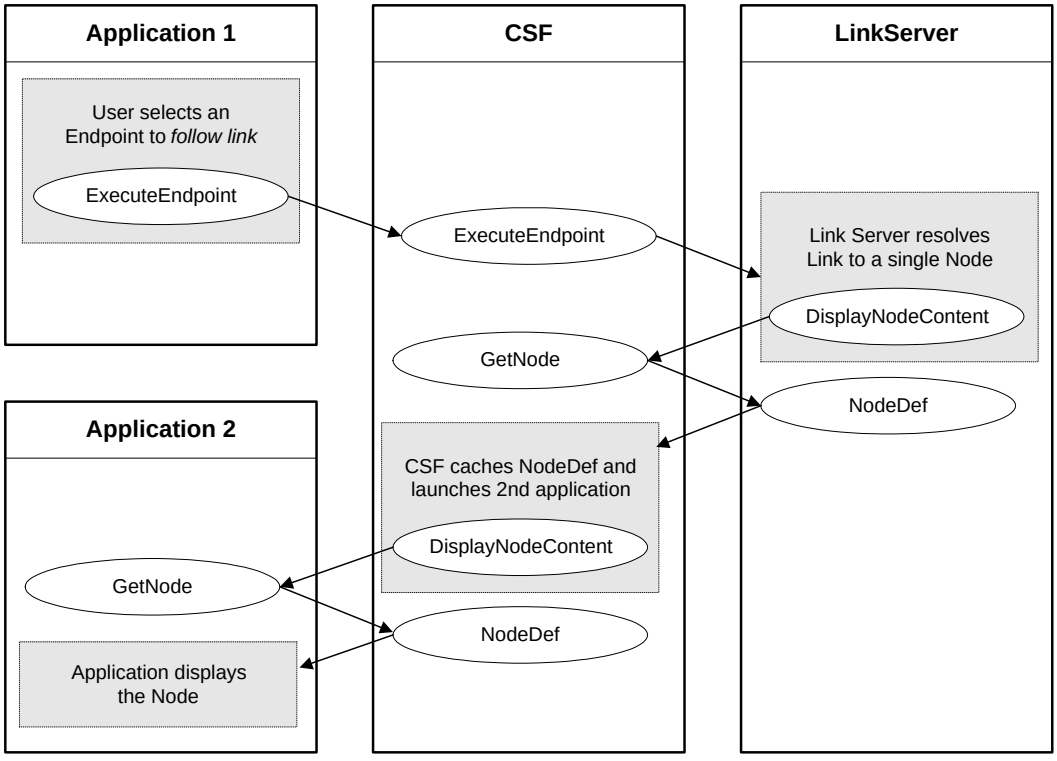

Figure 3.9: OHP-Nav Traffic

Figure 3.9 shows the messages sent during a simple following link process. In this case the link is only binary. Notice that whenever a DisplayNodeContent is sent it must be immediately resolved by the recipient sending a GetNode message. When the CSF receives the first DisplayNodeContent message it does not know the mimetype of the Node and must resolve it before it can choose an appropriate application to launch. It then remembers the Node details so that when it sends the DisplayNodeContent on, and the new application subsequently requests the Node details, it does not have to query the server again.

One possible solution might be to enable the link server to pass entire objects to the client (which would then not need to be resolved) while allowing the client to send object IDs to the server (which already knows all the objects and their IDs). In this way the overhead of sending messages for ID resolution could be avoided.

Routing

Another issue we encountered was within the routing system of the CSF. We found that the CSF needed to record which applications sent which message IDs, so that when a message was answered the RID value of the return message could be used to find the original application. As OHP messages no longer carry any channel information (as defined in the original proposal (Davis *et al.*, 1996)) we needed to 'hijack' the message ID on the way out so that it included the necessary information to identify it on its return as an RID. This avoided the problem of having to maintain possibly lengthy tables with time outs for all the values.

Although this was successful I felt at the time that the inclusion of an Origin field within the header of a message would make things simpler.

The CSF Protocol

One of the things that became apparent when designing the CSF was that there was an enormous amount of information about the system, the users and the client applications that the CSF would need to know about. The user (or administrator) will not be able to enter all this information in by hand, there is simply too much of it.

To cope with this, a new set of messages was needed, so that an application could communicate information about itself and its users to the CSF. This would not be a part of OHP, rather it would be a parallel protocol that dealt in users and systems. This is called the CSF Protocol (CSFP). The protocol also builds towards a more modular client system, using CSFP to communicate between components.

The CSF must know the following things per system:

- Security information
- Application to Mime type map

The CSF must know the following things per application:

- Security information
- Caching settings
- Presentation settings
- Mime types supported
- Internet awareness
- Supported scripting
- Multiple or single document view

The CSF must know the following things per user:

- Security information
- Presentation settings
- Application to mime type map

The exact nature of the Security information was never defined. It was accepted however that some security information would need to be stored and so it is included above.

It was important that the CSF knew if each application supported single documents (e.g. Netscape Navigator) or multiple documents (e.g. Microsoft Word) so that it knew if it needed to launch a separate instance of that application should a second document come through to it. Also, although an application may be Internet aware, the CSF may choose never to allow it to retrieve its own documents (e.g. for caching reasons).

One of the problems that we faced while developing the CSF was the correct launching of applications with documents. The process involved launching the required program and then waiting for it to connect back to the CSF. Once it had connected the DisplayNodeContent message could then be sent to cause it to load the document.

The problem is how does the CSF know which connecting application is the one it just launched. In OHP it is possible to follow n-ary links, creating the need to launch multiple documents at once. Thus it is not sufficient to expect the next connecting application to be the one just launched. Short of adopting a peer to peer architecture the best solution is for the launched application to send a CSF protocol message to the CSF, letting the

CSF know that it has been launched. This message includes the mime types that the application supports. This should be enough information for the CSF to then send the DisplayNodeContent message although errors could still arise.

In the system developed for the Hypertext ’98 demonstration the CSFP developed supported only a limited subset of the functionality listed above. There were three messages:

SystemMapDef: This is designed to be sent by a system program to inform the CSF about the entire set of Application to MimeType mappings.

GetSystemMapDef: This is sent by an application when it needs to know the current settings of the CSF, it is answered by a SystemMapDef message

Ready: The Ready message is sent by every application when it is launched. It allows the CSF to know which of the applications that may have contacted it are which. It does this based on the mimetype sent.

Naming

Although we had not adopted any particular naming policy for objects in the Hypertext ’98 demonstration (relying on an arbitrarily generated unique identifier, usually based on the time of object creation) it was always thought that a common naming system should be supported by the OHP suite of protocols.

The adoption of a global naming system would produce many benefits. In the same way that a URL can uniquely identify any ‘document’ on the Web, an OHP unique identifier could identify uniquely any of the hypertext objects used in OHP, e.g. nodes, endpoints, pspecs, etc. The identifier would be used as the OHP ID for the object. Thus we would have the situation where an OHP unique ID within a global distributed hypertext system could be sent via e-mail and still be valid.

Unfortunately a URL is far from being the perfect mechanism for such a task. Section 4.2.3 looks in more detail at the requirements for a naming scheme.

3.3 Architectural Assumptions (the CSF)

When we began to work on the OHP proposal we believed that addressing interoperability not only required some common understanding of a data model but that there must also be some underlying architecture that is jointly understood by all involved. This is because the data model usually makes assumptions about the architecture and because functionality and behaviour are to a large extent based on the architecture. Promising

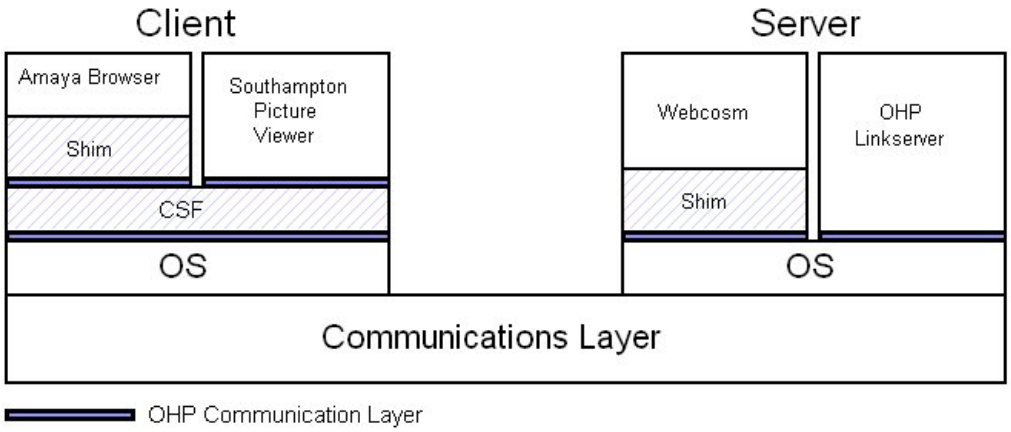

Figure 3.10: OHP-Nav Architecture

work in this area already existed (Goose *et al.*, 1997; Grønbæk & Wiil, 1997), however an assumed architecture grew alongside OHP-Nav. Figure 3.10 shows this current architecture.

From the diagram it is obvious that the architecture is not balanced. Why do we have a Client Side Foo (CSF) and not a Server Side Foo (SSF)? Many scenarios generated by the OHSWG have shown the CSF communicating with a single link server. In this case there is no need for a nexus similar to the CSF on the server side as no routing is needed. However in the situation where several link servers are all residing on a single machine or network then the role of a potential SSF becomes clearer, many of the functions it should perform are analogous to those on the client side.

- Message Routing
- Support of (Multiple) Protocols
- Synchronization and Compilation of Results
- Transaction Handling
- Concurrency Control
- Notification Control

This list could be optionally extended to include scripting engines, versioning control, configuration tools and security and user components. It can be reasonably assumed that since a protocol or interface is required on the client side (i.e. the CSFP) to organise these modules, one will also be required on the server side (e.g. an SSF Protocol, SSFP).

The problem with making these architectural assumptions is that they bring a great deal of baggage to every system that intends to implement OHP-Nav. The Danish system brought to Hypertext ’98 showed that it was possible to implement an OHP compliant

client without having to use a CSF. Instead it redistributed the functionality around other client side components. The same process could be applied to the SSF proposed above.

A more balanced role for a CSF (or CSF functionality) was summarised well by Anderson (Anderson, 1998). He identified the following important client side roles:

1. Process Invocation. Invoking clients as they are required (e.g. by cross application linking).
2. Session Management. Keeping track of users and their preferences. The CSF is also in a good position to keep track of how compliant each application actually is (e.g. does it only understand launch requests or does it know the entire OHP suite).
3. Protocol Shim. The original OHP draft included the notion of a shim that converted proprietary protocols to OHP and vice versa. Rather then implementing this for each client this functionality could be provided by the CSF once and used by all applications that spoke that particular protocol.
4. Service Provision. Providing tools for link creation and displaying link endpoints for user selection. Again useful to prevent each client from having to provide that functionality itself.
5. Proxy Support. The CSF is in an ideal position to provide caching to the system, both of files and also of meta-data.

The result of all this work with the CSF was to regard it not as a single application but as a collection of components that may or may not be present. Therefore it would be possible to provide all the services required using a single CSF process (as in the Southampton system), or to distribute that functionality amongst the clients themselves. Either implicitly (as in the Danish system) or by using some agreed client to client protocol for launching and caching (such as the proposed T3 protocol (Østerbye & Wiil, 1998)).

3.4 The Definition of OHP-Nav in IDL

One of the things that became clear from the OHSWGs experiences with tagged messages and later XML was that there was a need for an independent definition of OHP-Nav, one that could be translated into any particular syntactic language. With respect to this idea, the OHSWG also worked on an Interface Definition Language (IDL) definition which maps the existing OHP-Nav XML DTD to a CORBA compliant IDL. The full definition can be found in Appendix C.

There are basically three different ways of implementing a mapping of the asynchronous ASCII based navigational protocol to a CORBA interface definition.

1. OHP-Nav messages passed as ASCII strings: ASCII messages could be passed between communication objects that would support the navigational interface. The advantage of this approach is a precise mapping of messages between the XML definition and the IDL definition. Only the communication mechanism would be different. The disadvantages of this approach include the need for message parsing and the additional overhead for naming. Additionally, we lose many of CORBA's benefits such as typed objects and regular flow of execution.
2. OHP-Nav messages as typed method calls on communication objects: This approach requires that OHP-Nav messages are mapped onto IDL method definitions. The advantage of this approach is that the IDL mapping is still very close to the XML specification and that at the same time we receive the benefits of CORBA such as typed objects and a high level of communication abstraction. The disadvantage is that dedicated communication objects are still needed for communicating messages. These objects have to be 'exported' and 'known' by all the participating components.
3. Pure CORBA implementation: This approach takes full advantage of CORBA. It is characterised by defining all hypertext objects and their behaviour as interfaces and by implementing a set of components that publish and support those interfaces. Therefore OHP-Nav clients, link servers, CSFs and SSFs (if present) will all support a different set of interfaces. For example, a server would support all hypertext object interfaces, the notificational interface, etc. While a client would support only the notificational interface.

In this definition of the OHP-Nav IDL we mapped OHP-Nav messages to IDL methods. However, in CORBA we can return objects from method calls. This means, for many of the methods, that we do not have to rely on a return message. For example, a 'createEndPoint()' call will return an endpoint object directly to the caller. This is different to the purely asynchronous way ASCII messages are being communicated over sockets. In this case the 'createEndPoint()' message would be sent from a client to a server and the server would answer it with an 'endPointDef()' message.

By using this technique we are somewhere in between the second and third approaches. This enables us to imitate the XML definition without sacrificing all of CORBA's benefits. Before we can follow the third approach entirely we, as a community, would have to agree on further issues such as the naming of objects and the structure of the object model.

3.5 Summary

The development of OHP into OHP-Nav and the subsequent demonstration at Hypertext ’98 provided a focus of work for the OHSWG with the objective of showing interoperability between systems and to help eventually refine the specification to a releasable standard.

In this chapter I have explored this evolution, seen the definitions that were made and examined some of the problems that became evident during the development of the early prototype. With this work completed the OHSWG began to concentrate on particular areas of the protocol and the research effort became much less focused.

In the next chapter I will look at some of the missing areas of OHP-Nav, including Naming and Communication Infrastructure, and describe my work on the evolution of the SCRIPTSPEC opaque into a first class Computation object, deserving of its own OHP subset.

Chapter 4

Development of OHP-Service

4.1 Introduction

The Open Hypermedia Protocol (OHP) was originally designed to encompass the area of hypermedia navigation as well as several others that were deemed too important to ignore, these included collaboration and services. As work on the protocol progressed it became apparent that these sub-areas were more complex then one might expect. In addition, other domains had been identified e.g. Spatial and Taxonomic hypermedia (Nürnberg, 1997). To include this complexity in a general OHP protocol would give people a strong disincentive for using it, and yet the subjects were to important to exclude.

One possible approach would be to divide OHP into a suite of similar, related protocols, all using the same meta-information and delivery mechanism. Thus OHP Navigational (OHP-Nav), OHP Taxonomic (OHP-Tax) and OHP Space were born (although OHP-Nav is currently the only mature definition, containing most of the functionality in the original OHP definition). It was also decided to remove dynamic services from the Navigational protocol and place them into a package of their own that could be applied to any domain (The Service Package).

In this chapter we shall look at some of the work that evolved from the OHP effort and in particular examine the Service Package that was created by Sigi Reich and I. The software that I developed to test these definitions is presented and the new issues that the process of development uncovered are explored.

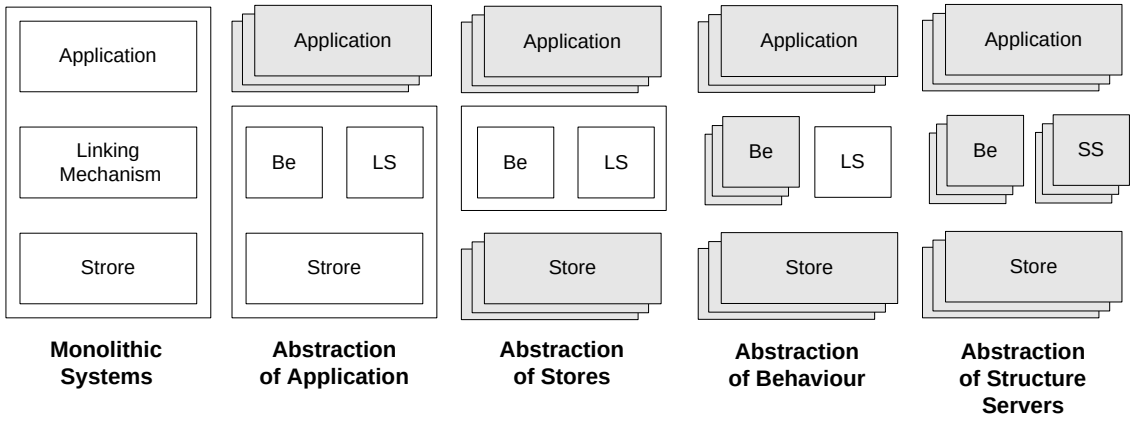

Figure 4.1: Evolution of Hypermedia Systems

4.2 The Evolving Interoperability Effort

After the Hypertext ’98 demonstration (described in Section 3.2) the OHSWG’s interoperability effort evolved in two complimentary directions. Firstly, the notion of multiple domains and a set of protocols to support them helped to spawn a new systems approach known as Structural Computing. This approach treats hypertext as a special case of a general philosophy of computing in which structure is more important than data.

Secondly, the OHSWG continued to develop OHP-Nav, but with a more formal and focused approach.

4.2.1 Structural Computing

During the OHP development period a new approach to hypermedia research started to emerge. Structural Computing ‘asserts the primacy of structure over data’ (Nürnberg *et al.*, 1997), as such it is concerned with looking at how structure can be discussed and managed at all levels of computing. In this world view, OHP-Nav is simply one protocol of many that facilitates different processes operating across structure, in this case the processing of navigational structure.

Figure 4.1 shows the evolution of hypermedia systems over time (Nürnberg *et al.*, 1997) from left to right, where *Be* stands for *Behaviour*, *LS* stands for *Link Server* and *SS* stands for *Structure Server*. Early hypermedia systems were monolithic but gradually parts of the system were abstracted away into separate processes. The third iteration, the abstraction of stores, represents many of the OHS systems developed to date (as described in Section 2.6).

The two further abstractions represent the evolution of these systems into Component-Based Hypermedia Systems (CB-OHSs). These systems further abstract their middleware. The abstraction of behaviour deals with the idea that arbitrary computations should be possible over the structure served from a Link Server while the last abstraction allows for arbitrary computations over the structure served from arbitrary Structure Servers. It is this last stage that finally provides a framework that supports Structural Computing.

In OHP terms, the different Structure Servers would serve the different structures from the various hypermedia domains (Nürnberg *et al.*, 1998). Thus a Navigational Structure Server would be developed that provided an OHP-Nav interface to clients. Other OHSWG developed protocols, such as OHP-Space, would be supported via separate Structure Servers.

4.2.2 The Data Model

One of the concerns about a multi-domain suite of protocols was that, where necessary, they should share the same data model. For example, nodes from the Navigational Domain might be reusable elsewhere. This was a problem as up to that point there had been no separate data model definition; all the structures were defined implicitly within the protocol (see Section 3.1.6).

It had also been recognised that as well as run-time interoperability a useful outcome of the work of the OHSWG would be a standard interchange format and that this would also require a standard data model that was separate from the protocols (Grønbæk, 1998).

For both these reasons the hierarchical data model shown in Figure 4.2 was defined (Grønbæk & Sloth, 1999). The model shows Presentation Specifiers (PSpecs), Service Objects (Computations) and Contexts, even though at that point they were not well understood.

4.2.3 Naming

Another facet of the protocol that had not been fully explored was the problem of naming. It has been said that naming is a critical feature of a hypermedia system (Tzagarakis *et al.*, 1999) and that to crack the naming problem is to move significantly closer to solving the distributed information systems problem as well.

Naming as an issue in Open Hypermedia Systems, in particular those using the OHP suite of protocols, has been raised many times during the development of the suite (Millard *et al.*, 1998; Nürnberg & Leggett, 1997; Nürnberg *et al.*, 1998). However, from an OHP point of view it is not so much an issue to develop our own naming system as it is

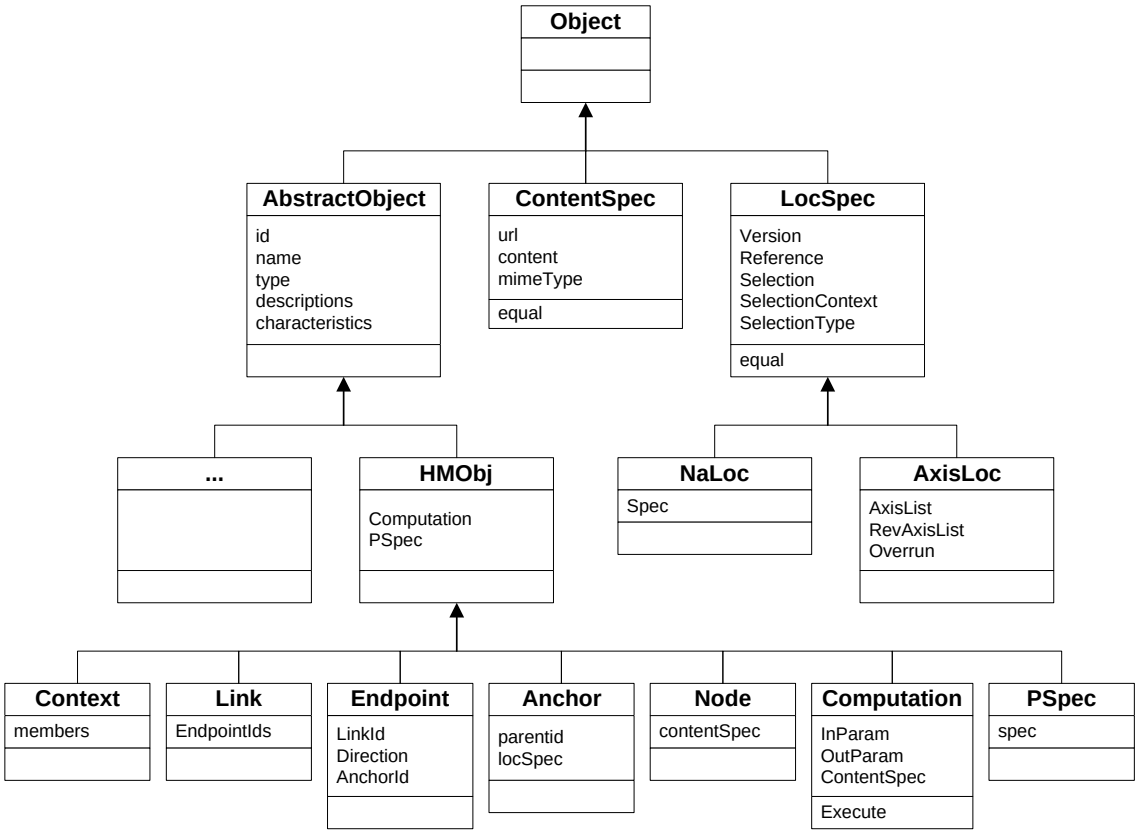

Figure 4.2: OHP Suite Data Model

to rely on existing systems and agree on naming conventions. Despite this it is entirely possible that some aspects of a proposed naming scheme might influence the architecture and/or operation of OHP. For instance, we might need a standardised 'lookup service'.

4.2.4 Hypertext Requirements

All the objects derived from AbstractObject (shown in Figure 4.2) are subject to naming. OHP makes no assumption about the naming system used to create it's IDs and has always defined the IDs to be opaque strings. As with naming in any distributed system there are a number of issues to be addressed by OHSs. Sigi Reich and I did some initial work in this area and identified the following ones (Reich *et al.*, 1999a):

- *Scope*: names should be globally valid, their meaning should not depend on the location (Sollins & Masinter, 1994), i.e. a resource's name should not give any hint as to where the resource is (physically) located (which provides location transparency (Tanenbaum, 1995)).
- *Uniqueness:* names should be globally unique. I.e. the same name will never be assigned to two or more different resources. Furthermore, the lifetime of a resource is potentially unlimited so that names have to be persistent (Sollins & Masinter, 1994).

- *Readability:* names should be readable and interpretable by humans and they should be easily transcribable (Berners-Lee, 1994; Engelbart, 1990).
- *Scalability:* there are different aspects to scalability such as number of accesses, number of resources, etc. With respect to the number of accesses it can be said that a central broker dealing with name resolution will scale badly even when replicated; with respect to the number of resources it has to be said that resources might conceivably be available on the network forever, i.e. the number will be very high.
- *Mobility:* resources might be subject to re-location at other places (Guenther, 1999). For instance, a set of links might be imported from a different linkbase (and naming clashes will have to be resolved) or also services might be mobile and have different locations over time. Naming should be location independent, i.e. resources should be relocatable without affecting their names (Tanenbaum, 1995).
- *Querying:* often users will not know the exact name of a resource so that it can be looked up. However, they will know some attributes such as author, type, etc. In this case it should be possible to provide the user with a mechanism to look up a name given a set of attributes.

At the same time Tzagarakis et al. were investigating naming as a mechanism of distribution at the University of Patras, Greece, using their system Callimachus (Tzagarakis *et al.*, 1999). Along with Sigi Reich they have since applied their naming philosophy to OHSs and produced some promising first drafts of a naming standard for Open Hypermedia (Tzagarakis *et al.*, 2000).

4.2.5 Collaboration

Human collaboration on a hypertext has always been an important issue in hypermedia research, it was a part of Bush's early vision (Bush, 1945) and formed one of Halasz's seven issues (discussed in Section 2.4.6). The idea of cooperative hypertext has also been explored in later generation systems such as SEPIA (described in Section 2.6.4).

The interoperability work of the OHSWG has always been based around the notion of system components working together and thus might be regarded as a different kind of co-operation to that described above. However the difference between a component and a user can be ambiguous in a sophisticated system, furthermore the ability to control simultaneous human access to a hypertext is always desirable. For this reason support for collaboration was thought of as an important part of the OHSWG's work.

Haake and Wang identify five characteristics of a cooperative environment (Haake & Wang, 1998):

1. Management of shared objects
2. Management of shared user interfaces
3. Support for group awareness
4. Support for coordination
5. Support for communication

Thus to provide collaboration on an OHS they believe it is necessary to make the OHS ‘group aware’ by adding the notion of sessions to the system and providing access to shared hyperstructures through group aware applications. A session manager is responsible for managing the access to the system in one of several cooperation modes.

Wiil and Nürnberg point out that with a CB-OHS, sharing and interaction can take place at both the hyperstore and structure service levels (Wiil & Nürnberg, 1998). While sharing at the hyperstore level allows multiple users to access the information, sharing at the structure server level enables more powerful collaboration modes and requires session management. It has been proposed that this provision of session management and coordination should form a Workflow protocol for the OHP suite (OHP-Wf) (Wang & Haake, 1999).

While other members of the OHSWG concentrated on collaboration, I began looking at how the Service definition from OHP-Nav could be expanded into a protocol in its own right.

4.3 Services

Services first appeared in the original OHP definition (Davis *et al.*, 1996), although they were never completely defined. A Service is a ‘black box’ of functionality, known only by name to a client, that can be invoked and its results understood, even though its workings are completely opaque. A Server can supply a description of this ‘black box’ to its clients and in this way a generalised client gains access to complex functionality that would otherwise be unavailable.

I spent some time developing the OHP-Service protocol and model with the help of Sigi Reich. We had the initial objective of re-examining the old OHP-Nav view of Services and producing a more sophisticated definition (Millard *et al.*, 1999). Although we achieved this we also came to view the entire OHP-Nav approach in a new light, questioning not only if it was appropriate to define a dynamic service mechanism within a hypertext suite, but also whether or not the hypertext suite was not itself merely another type of service.

We began by considering the different ways in which it is possible to examine the data in an OHP-Nav system. Effectively there are three:

1. By using the operations defined in OHP-Nav, i.e. Create, Delete etc;
2. By utilizing a separate query facility based on the data model, i.e. ‘get all Nodes where Attribute “Author” equals “D.Millard” ’;
3. By using a Service Package to dynamically discover what services a server has to offer and then dynamically invoking them. This has the advantage that it allows us to use functionality stored on the server as well as data (for example, an effective search algorithm that depends on system knowledge to operate);

The confusion over these three possibilities goes right back to the older definitions of OHP-Nav (Davis *et al.*, 1997) where there were four standard services that had defined semantics:

1. *Follow Link* - returns all the other endpoints (not the original input endpoint) which are associated with the input endpoint via any link structure;
2. *Show Endpoints* - returns all the endpoints within the input node;
3. *Show Links* - returns all the links that connect to the input node;
4. *Get Relevant Nodes* - returns a list of nodes managed by the link server that may act as starting points for the user;

In the subsequent revisions of OHP-Nav (1.2 and 1.3) services were removed from OHP-Nav as they were believed to belong in a sub-protocol of their own. As a result these standard services have become operations. This has a few advantageous side effects. It means that any server that is OHP-Nav compliant must understand and implement all four operations, as opposed to standard services which a server may or may not offer. Also it means that the messages that deal with this functionality can be much more specifically tailored for the task (rather than being generalised to cope with any service).

We concluded that anything beyond these four standard services should be represented as Service Objects within the system and I spent some time examining the requirements for such services and defining a mechanism through which they could be provided.

4.3.1 Services as Objects - The Computation

In networking the word ‘service’ means a facility that is available on the network (Tanenbaum, 1995). To avoid overloading names the objects that represent services in the OHP data model are known as Computations. A Computation stores the information about

a service in the same way that a Node represents a document, it is a meta-information object.

Although Computations are defined in the Service Packages it was always intended that they may be referenced from other domains. For example in OHP-Nav all hypermedia objects have a field that represents the ID of an attached Computation. The semantics of this attachment are not defined, i.e. if a Computation is attached to a Node it may be executed when that Node is opened, activated, played, closed or on any other event. Without a complete event model this is impossible to implement in a generalised way. However Computations can still be associated with objects, it is merely up to the client to decide when to invoke the service.

Computations are complete objects in the system and can be referenced by ID and even linked to, although the actual behaviour of following a link to a Computation is undefined. A Computation contains information for the user (name and explanation of the functionality) and also the input and output parameters of the service. In this way a generalised client can understand how to call the Computation and what to expect in return.

4.3.2 Progress and Feedback

In experimenting with services it became apparent that one of the problems is that a service may actually take a long time to complete. This becomes a problem if once the service has been initiated the only related message a client can expect is the completion message.

Thus our major conclusion from experimenting with services was that some mechanism was needed to either let the user/viewer know the time that a service takes to complete or to send progress messages to give some progressive feedback. We considered two options, adding an ‘expected time to completion’ field to the Computation definition or adding progress messages to the message set.

The problem with adding an approximate time to complete to a service’s definition is that the time actually taken is very dependent on the load on the processor at the time of execution and also varies with other users activity and processor load on the machine. On the other hand the problem with progress messages is they do not convey to the user at the time of invocation the actual speed of the service. Thus a user could select a slow service believing it to return instantaneously.

As a compromise we decided to combine both approaches. The service definition contains (as an XML attribute) the Expected Time to Completion (ETC) which is given

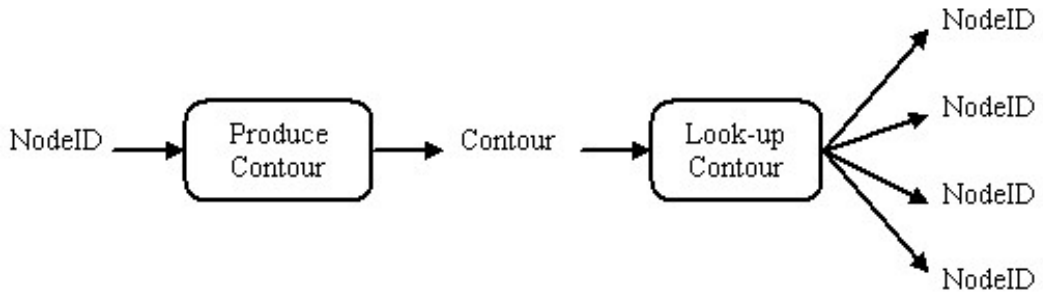

Figure 4.3: Combining Services

the value of one of the five recommended bands. In addition a service progress message provides the necessary information for a client to track progress once it has started. However there is no assumption about the frequency of progress messages or even that a server will send any at all.

4.3.3 Composite Services

The simple black-box system of Computations described above handles many different cases of Service (including all four found in the original definition). However there are cases when the opaque nature of the approach becomes a problem.

Take the example of the service ‘Find Similar Music’ (FSM), that analyses a section of music and produces a contour (Blackburn & De Roure, 1998). It then checks that contour against the ones in a database to produce a list of nodes representing musical scores similar to the selection. This is an ideal candidate for a service; a complicated operation with well defined input and output parameters that could be used by a generalised client.

There is a problem. Because the service needs to examine the music file to produce the contour, it must either move the music file to the service (which could potentially take a very long time) or it has to move the service to the music file. The first option becomes unrealistic with files larger then a few tens of Kbytes, the second has to deal with the problems associated with mobile code.

We could still provide the service to a specialised client that knew how to produce a contour, if we broke the FSM service into two. The first part, ‘Produce Contour’ (PC), takes a selection and produces a contour. The second, ‘Look-up Contour’ (LuC), takes a contour and checks it against the database. If the client supported the PC service and the server supported the LuC service, then by combining the two, we get the original FSM service (see Figure 4.3. In this way the idea for a Composite Service was born.

Composite Services contain similar information to computations with the addition

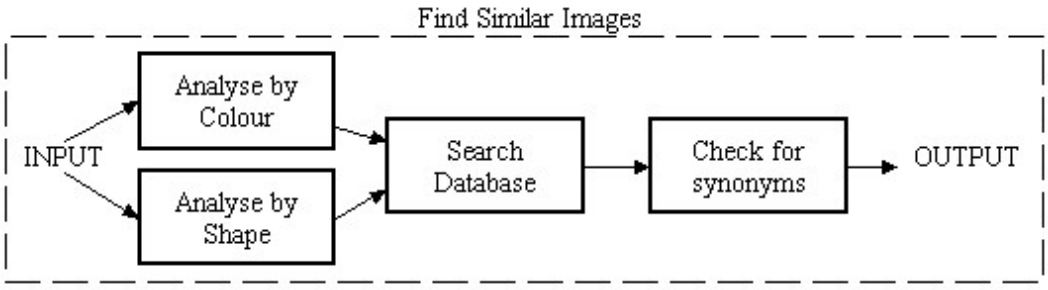

Figure 4.4: The ‘Find Similar Images’ Composite Service

of a computation graph. The computation graph describes how computations can be combined. It allows computations to be called in serial or in parallel (see Figure 4.4). By having a non-cyclic graph we avoid a lot of the problems with managing the flow of data (such as getting stuck in loops) while retaining most of the power of the concept.

A Client that wishes to use a Composite Service has the option of invoking it with a single execute message as long as the Composite Service has its own identifier (this would put the onus on the server to combine the services). If there is no identifier then the server expects that the client must know where to find each of the sub computations (either from the server, itself or a third system component) the client may then call an execute message in turn on each service, passing the output of each into the next, until it has finished.

Composite computations represents a way of a component ‘imparting knowledge’ to the rest of the system about how computations can be combined. This leaves plenty of scope for the development of more sophisticated systems which learn about particular combinations of computations and also provides a framework for the exploration of mobile code within a CB-OHS environment.

4.3.4 The OHP-Service Package

There is a complete set of operations for manipulating Services, most of which reflect those already defined in OHP-Nav (Create, Delete etc.) However there are several operations unique to the Service Package. These are presented in Table 4.1:

EXECUTESERVICE provides a means to invoke a service with a set of parameters resulting in a SERVICEEXECUTED message containing ranked results.

The full DTD for OHP-Service can be found in Appendix D.

Operation	*Description*
RETRIEVESERVICES	Requests a list of all appropriate services from the server
SERVICESRETRIEVED	Gives information about all relevant Computations to the recipient
RETRIEVECSERVICES	Requests a list of all appropriate Composite Services from the server
CSERVICESRETRIEVED	Gives information about all relevant Composite Services to the recipient
SERVICEPROGRESS	Gives information about how near completion an invoked service actually is
EXECUTESERVICE	Invokes a given Service with a set of parameters
SERVICEEXECUTED	Returns the result of a given service via a set of results

Table 4.1: Computation Operations

Service Definition

For each service object (Computation), we are interested in its definition, how to call it and its result. The DTD extract shown in Figure 4.5 describes how the Computation is defined. A Computation is comprised of a globally unique service id, a specification id (SPECID) which uniquely identifies the functionality of the service (i.e. 'Follow Link'). The name and function name (NAME and FUNCTIONNAME) give a brief description suitable for display to a user and a more comprehensive definition of what the service actually does. The following three element types describe in turn what parameters the service takes, what results it produces and which mime types it is applicable to. The codespec field was never used but was intended to be a location for mobile code (i.e. a client side script) and the expected time to arrival field (ETC) appears as discussed in Section 4.3.2

Each template contains a type entry that defines what that parameter is. Any OHP component can define its own types which can be anything that can be represented as a string. However we do assume some standard types, including the OHP objects themselves (ohpObject from the OHP-Nav DTD) and some basic types (INTEGER (4-byte), DOUBLE (8-byte), BOOLEAN (TRUE — FALSE), STRING and LOCALFILE (the local representation of a file)). Although it is much more specific then the others the LOCALFILE is necessary and is used a great deal. For example, multimedia objects are often very large and ideally should be processed at their current location (to reduce file transfer overheads). In these cases a node is not sufficient as it does not inform the client that the file used should be a local one. Without this knowledge the client might try and invoke the Service on a remote file which would then have to be downloaded before

```
<!ELEMENT COMPUTATION ((ID, SPECID, NAME?,
FUNCTIONNAME?, INTEMPLATESET?,
OUTTEMPLATESET?, MIMETYPESET?, CODESPEC?, ETC?)>

<!ELEMENT MIMETYPESET (MIMETYPE, MIMETYPE) >
<!ELEMENT MIMETYPE (#PCDATA)>

<!ELEMENT INTEMPLATESET  (INTEMPLATE, INTEMPLATE*)>
<!ELEMENT OUTTEMPLATESET (OUTTEMPLATE, OUTTEMPLATE*)>

<!ELEMENT INTEMPLATE   (((HRANGE, LRANGE) | POSSIBLEVALSET?),
DEFAULT, NAME, TYPE)>

<!ELEMENT POSSIBLEVALSET (VALUE, VALUE, VALUE*)>

<!ELEMENT OUTTEMPLATE (NAME, TYPE) >

<!ELEMENT HRANGE  (#PCDATA)>
<!ELEMENT LRANGE  (#PCDATA)>
<!ELEMENT DEFAULT  (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT VALUE (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT FUNCTIONNAME (#PCDATA)>
<!ELEMENT ETC (#PCDATA)>
<!ELEMENT SPECID (#PCDATA)>
<!ELEMENT CODESPEC (#PCDATA)>
<!ELEMENT TYPE (#PCDATA)>
```

Figure 4.5: XML DTD definition of a Service

processing could take place.

This definition does not support arrays and structures as these made defining and sending parameters overly complex and we didn't want to re-invent RPC with XML (although attempts to do exactly that are being made (xml, 2000)). Instead we assume that a service that wishes to produce a list of results will simply produce multiple outparams with the same name. Alternatively the component can use its own container types.

Composite Services

As discussed in Section 4.3.3, services can be composed of other services. For instance a 'Look Up By Contour' service that can be applied to a piece of music consists of a 'Feature Extraction' service and a 'Contour Matching' service. Each composite service has its own description and name. It may also have an ID, in which case components

```
<!ELEMENT CC  (ID, COMPUTATIONGRAPH, FUNCTIONNAME?,
NAME, SPECID)>

<!ELEMENT COMPUTATIONGRAPH  ((PARALLEL | SPECID), (PARALLEL | SPECID)*)>
<!ELEMENT PARALLEL  (SPECID, SPECID, SPECID*)>

<!ELEMENT ID   (#PCDATA)>
<!ELEMENT FUNCTIONNAME  (#PCDATA)>
<!ELEMENT NAME   (#PCDATA)>
<!ELEMENT SPECID (#PCDATA)>
```

Figure 4.6: XML DTD definition of a Composite Service

may execute the composite service directly by giving it the input parameters of the first service and expecting the output parameters of the last service.

The DTD extract in Figure 4.6 shows the definition of a Composite Service. This contains a unique id, specification id, user name and description like a normal service. It also contains a list of service ids that represent which services are to called and in which order (the computation graph). If the service ids are surrounded by the parallel tag then they must have the same input and output parameters and are executed in parallel. This means that the output results of the services before them are sent to all services in the parallel segment, then the output results of all of those services are combined and sent to the next service in the list. This is useful in situations such as those where different feature extracting algorithms need to be applied at the same stage and the results compiled later.

There are cases where the component offering composite services does not actually offer one of the sub services. These composite services do not include an ID and therefore cannot be called directly. It is the responsibility of the calling component to call each sub service directly and chain them together. To do this it obviously must know of other components that do offer the sub services, or offer them itself!

This is especially useful in cases where the actual location of where a service is to be performed can play a crucial role (e.g. services applied to large multimedia documents). For instance, a Feature Extraction service on a huge .wav file should at best happen at the file's location so that the file does not have to be copied (see definition of LOCALFILE in Section 4.3.4). In this case the component offering the composite service may not actually offer the Feature Extraction. Instead it has to be offered by a client side component, maybe even the calling application itself.

4.3.5 A Descending Order of Complexity

I also realized that it could sometimes be the case that a service designer would be faced with a decision. Either they use their own parameters and offer a more powerful service or they use standard parameters and offer a reduced service. For example the FSI service described in Figure 4.4 returns a set of ranked nodes. This ranking value is only one of many outputs from a FSI algorithm, many of which also offer standard deviation from the input node and comparison to the mean values of all the images processed (Rui *et al.*, 1999).

We could have created a more powerful service by defining our own output parameter, of type FSI_RESULT, that would have included all this extra information. Unfortunately this would have meant that only viewers that understood the FSI_RESULT could use the service. To rectify this it would have been possible to implement a second service, identical to the first other then the return type. In this case a viewer that did not understand the FSI_RESULT would only display the simpler service. But this would have meant that a viewer that understood both would be offering two identical services to its users, which could lead to confusion.

In Section 4.3.4 we described the Spec ID a field that is universal to a particular piece of functionality, whatever the implementation. In the case of our multiple FSI services, both would have identical Spec ID so a simplistic approach would be to allow the viewer to pick one to offer to the user. It may well have chosen the FSI_RESULT as this is the more complex object and therefore probably the more powerful service. But it is important at this stage to differentiate between knowing what a service is (e.g. the Spec ID) and knowing what the parameter types are.

Any viewer which understands what the FSI service is could be written to offer and use a particular implementation of that service, throwing away all the other offered services with the same Spec ID. However if the viewer understands what the FSI_RESULT is but not the service then it may become confused, what if three services are offered all with the same Spec ID but all with different parameters, which one should it use?

In this case a possible solution would be to amend the service definition to include a descending order of services. These are all services with the same Spec ID but different object IDs that are ranked in the order in which the viewer should consider them. If the viewer cannot understand the first it moves on to the second and then the third and so on, until it finds a service that it does understand. In this way a link server may offer a service in several stages of complexity without worrying about how the viewers will interpret them.

The problem with this approach is that it adds complexity, both to the messages themselves and the interpretation that must take place at the client end. For this reason the descending order of services has never been added to the definition, although it remains a possible future addition.

4.4 Demonstration at HT'99 : The Solent System

At the OHSWG's meeting at Southampton (OHS 4.5) it was decided that as the Hypertext '98 demonstration (see Section 3.2) had formed such a positive focus point for the group the same should be attempted for Hypertext '99 in Darmstadt, Germany. It was also decided that since we had demonstrated interoperability at Hypertext '98 we should concentrate on showing some of the features of the protocol.

Some of the more successful parts of the Hypertext '98 demonstration were the collaboration aspects. It was thus decided that the Danish contribution to the Hypertext '99 demonstration would be to extend this simple support into a more advanced system and define a collaboration extension to OHP (as discussed in Section 4.2.5). The Southampton contribution would be to finalise the computation definition (as described in Section 4.3) and create a more complete working example.

This software, developed by myself and Jon Griffiths at the Multimedia Research Group, was a Component-Based Open Hypermedia System (CB-OHS) which we refer to as the 'Solent' system (Reich *et al.*, 1999b), named after the waters around Southampton. The system was designed to address the many issues posed to hypermedia middleware and to serve as a platform for the development of the OHP-Services interface, in particular in its support of advanced multimedia applications such as content-based retrieval (Christodoulakis & Triantafillou, 1995), content-based navigation (Lewis *et al.*, 1996) and navigation in audio (Blackburn & De Roure, 1998).

4.4.1 Description of the Architecture

The architecture of the *Solent* system builds on the experiences of the IAM research group at Southampton, gained in developing the open hypermedia systems Microcosm (Davis *et al.*, 1992), Microcosm TNG (Goose *et al.*, 1996), MEMOIR (De Roure *et al.*, 1998) and others. It also builds on the CSF components that I developed as part of the earlier Hypertext'98 demonstration (described in Section 3.2). Figure 4.7 depicts the conceptual architecture.

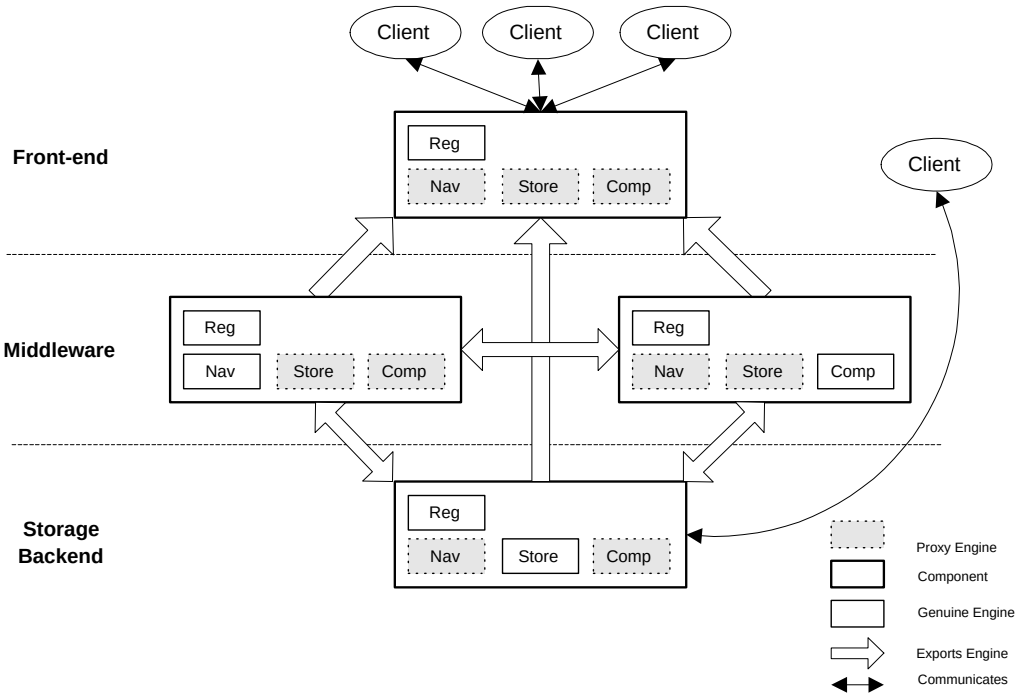

Figure 4.7: The Component Architecture of the *Solent* System

The Figure shows the separation of components into front-end (i.e. the applications), middleware services, and the storage back end. This is in similar to other CB-OHSs such as Microcosm TNG (Goose *et al.*, 1996) or Construct (Wiil & Nürnberg, 1999).

Key to the system is the notion of an 'engine', which is a software process managing a certain subset of functionality within the system. The most essential engine, present in every component, is the *registration engine* ('Reg' in Figure 4.7). The purpose of this engine can be compared to that of an information broker in CORBA. This engine is tightly coupled to the hosting component. Its job is to receive registration requests from other components in the system. When one component registers with any other it sends its connection details (the protocol and version it speaks as well as the host and other location information that it resides on). The registration engine then creates a *proxy engine* (depicted as grey boxes in Figure 4.7) in its own component and sends its own registration information back. Registration and de-registration can thus be truly dynamic.

A proxy engine appears to the outside world as would any other engine of the component but in actual fact it proxies all requests to a real engine in a separate component. When a component registers it sends details of all its engines, including any current proxy engines. Because the proxy engine retains the connection information of the genuine engine, when it registers it sends this information rather then that of the proxy. This means that messages are only ever re-directed once (client to proxy to engine). Any component

can be treated as if it contained an appropriate engine as long as it knows the location of such an engine in another component.

The communication objects are currently configured to work using messages represented in XML over plain TCP/IP sockets. Other communication mechanisms such as CORBA's IIOP or Java's RMI have been investigated as well; however, due to reasons of platform independence and standardisation within the OHSWG, XML over plain Sockets has remained the communicating mechanism of choice. Each incoming message is offered to each engine in the component, it is up to the engine to decide if it wishes to deal with a particular request. Each engine does this based on the protocol and version of the message being sent (if it can differentiate between them) and also the name of the engine that the message was intended for. As a result, concurrent support of multiple versions of a particular interface is possible.

This architecture results in a group of communicating components that can be dynamically configured and distributed over multiple platforms and at the same time act together cohesively as a whole. Even though currently only two core hypertext interfaces are supported, namely the navigational and the service interfaces, the architecture allows for an easy integration of interfaces to other hypertext domains such as spatial hypertext (Reinert *et al.*, 1999) or workflow applications (Wang & Haake, 1999).

4.4.2 Sample Applications: Generic Media Player and Car Stereo

A number of hypermedia applications have been built using the *Solent* system. In this section I will briefly describe two examples, a software Car Stereo and a generic Media Player. The Car Stereo Viewer replicates a car CD system which can play files encoded in the MP3 standard, it was built specially for use within the *Solent* system. The second application, a generic media player, is an adapted client of the windows media player which has been developed for content based navigation in audio (Blackburn & De Roure, 1998); it demonstrates how services can be dynamically discovered using the computational interface. Figure 4.8 shows the setup.

As can be seen in Figure 4.8, the adapted generic Media Player communicates using both the navigational and computational interfaces, whereas the purpose-built Car Stereo client only uses the computational interface. The Car Stereo has a built-in set of requests it understands and these can be invoked by users by pressing a button. E.g. users might want to ask for 'similar songs' to the one that they are currently listening to.

The Media Player on the other hand, is a generic application that supports the computational interface. As a result this application is able to negotiate with a computational engine about the set of services it is offered. The Media Player understands the basic

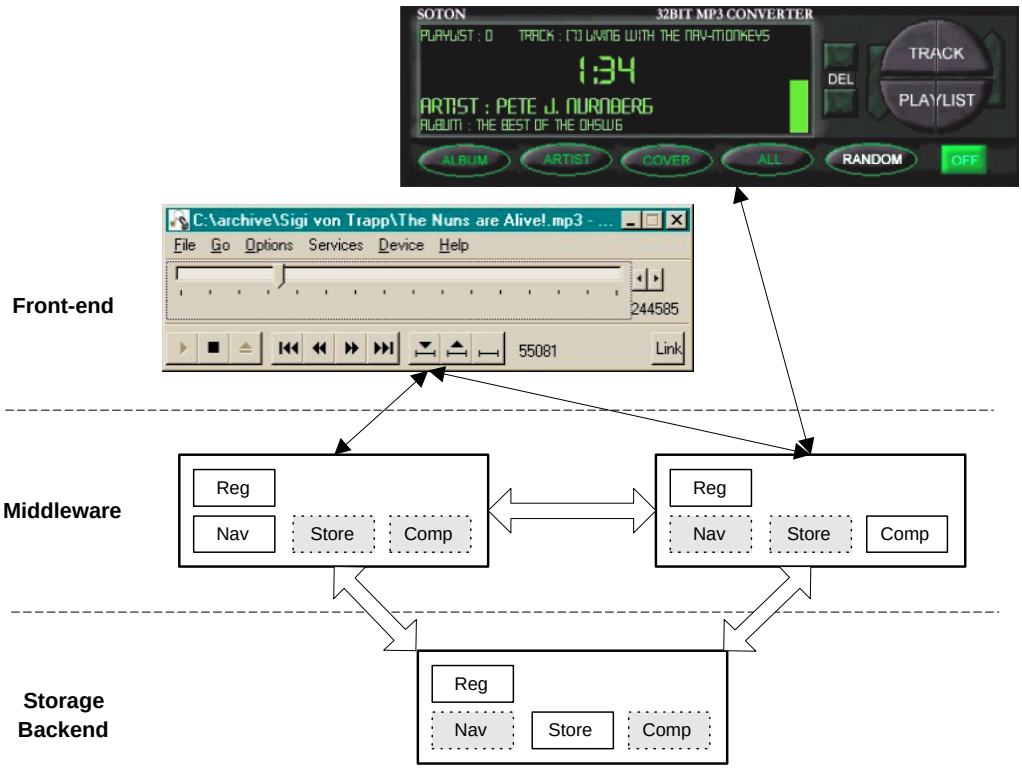

Figure 4.8: Media Player and Car Stereo as Sample Applications of *Solent*

mechanisms on how to call a computation and how to parse its results, i.e. it knows how to deal with input and output parameters. In doing so it can dynamically discover the set of computations available, offer them to the user, perhaps ask the user to provide it with some additional parameters and finally execute the computation. By supporting the navigational interface the Media Player not only allows users to retrieve tracks by executing a computation, but also to navigate from a track to some other destination, e.g. a description in the form of a HTML file on the Web.

4.4.3 Lessons learned from Solent

The Solent system successfully demonstrated not only the point of dynamic computations but that the OHP-Service protocol described in Section 4.3 is capable of implementing them. The experiences we gained during the design, implementation and prototyping were manifold.

- The modular design clearly helped in restructuring the system to the different application areas and allowed us to better select the components to support various functionalities (Shackelford *et al.*, 1993);
- Furthermore, this flexibility helped to address the problems encountered in serving different versions of protocols and interfaces that we needed to support during

the development of standardised interfaces within the OHSWG's interoperability effort;

- The modularity of the system allowed us to run different interfaces simultaneously assisting interoperability, in particular at the front-end (Wiil & Nürnberg, 1999);

However it was the storage layer and our arbitrary XML-based storage component that provided some of the most important lessons about communicating structure between components.

4.5 Experiences with XML

During the development of the Solent system we discovered that one of the problems with working with an evolving protocol is that it is very difficult to build software around it. The development time for the software is generally greater then the version time of the protocol. To combat this problem in Solent a separate component is responsible for all the storage and retrieval of hypermedia objects, represented in XML. The advantage of XML is that there is a discernible structure, even though the storage component does not understand the hypermedia object that the structure describes. Because of this the component is able to store arbitrary objects, as long as they conform to an XML hierarchy.

Because of this, we could change the structure of the hypermedia objects without having to change either the parsers or the storage back-end of the system. Unfortunately it also meant that we have had to develop algorithms that can search an XML hierarchy quickly and efficiently.

4.5.1 Storing Arbitrary XML Structured Objects

The Solent System stored the XML structures within a relational database, with the Solent storage component acting as an interface. This storage component was capable of storing any XML hierarchy and searching that hierarchy using pattern matching.

Figure 4.9 shows an OHP-Nav node, defined in XML that might have been stored in the database and Figure 4.10 shows the same object represented graphically.

Figure 4.11 shows the way in which it would have been stored in the Solent database. The SETS table stores the element (branches), the VALUES tables stores the values (leaves) and the HIERARCHY table records the order of the elements and leaves within a parent element.

```
<NODE>
   <ID> id12345 </ID>
   <CONTENTSPEC>
      <URL> C:\\archive\\All Saints\\1 - Never Ever.mp3 </URL>
      <VERSION> OHPNav-1.3 </VERSION>
   </CONTENTSPEC>
</NODE>
```

Figure 4.9: XML Object to be Stored

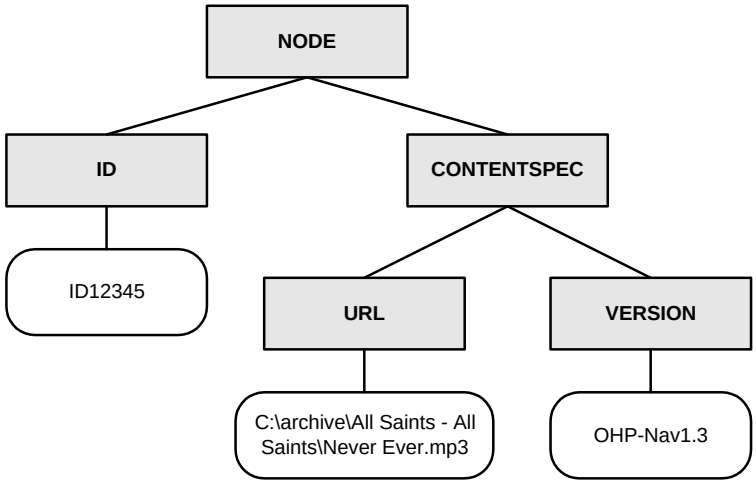

Figure 4.10: XML Object Hierarchy

4.5.2 Pattern Matching Algorithms

A consequence of storing all (structured) objects as flat relations in the database is that the retrieval of objects involves building XML elements composed of smaller entities: sub-elements and values. Given a particular node of the tree, it is relatively trivial to write code that constructs the XML model in memory. A more difficult task is searching the XML hierarchy for patterns while it is still stored in the database. We considered several approaches to address this problem (shown in Figure 4.12):

Breadth-First Retrieval

Firstly, data was retrieved using a *breadth-first* approach. This method begins at the element root and works its way down the tree. The problem with this approach is that when storing similarly structured objects in the database the difference between objects meant for retrieval and ones that are not may only be perceivable when examining the values of the leaves of the retrieved XML structure. Therefore there may be a lot of retrieval activity within the database, which is retrieving structure that is not associated with the database objects meant for retrieval and this is not discovered until late into the retrieval process. This is an inherent flaw when retrieving XML structure using a

SETS Table

ID	Name	Belongs To ID
S1	NODE	-
S2	CONTENTSPEC	S1

VALUES Table

ID	Name	Belongs To ID	Version	Value
V1	ID	S1	OHPNav-1.3	id12345
V2	URL	S2	OHPNav-1.3	C:\ archive\ All Saints\ 1 - Never Ever.mp3
V3	VERSION	S2	OHPNav-1.3	OHPNav-1.3

HIERARCHY Table

ID	Order	Belongs To ID
S1	1	-
V1	1	S1
S2	2	S1
V2	1	S2
V3	2	S2

Figure 4.11: The XML Object Stored in the Solent database

top-down approach, where the discerning information is located at the bottom of the structure.

Depth-First Retrieval

In a next step, data was retrieved using a *depth-first* approach. This involved retrieving the value at the bottom of the branch for each unexplored sub-element of an element. A major problem concerned with this approach occurs when there are lots of objects in the database which match the object meant for retrieval. By retrieving objects from the bottom-up, the retrieval algorithm may begin constructing objects which comprise the same structure as one of those objects meant for retrieval, but may actually contain the values and/or sub-elements which belong to different objects. This is an inherent flaw when retrieving XML structure using a bottom-up approach, where the discerning information is located at the top of the structure.

Combining Depth-First and Breadth-First Retrieval

Thirdly, we investigated a *combination* of depth-first and breadth-first retrieval methods. It follows the same steps as the depth-first retrieval technique until the first branch is encountered within the template object. Then instead of searching the database for the value

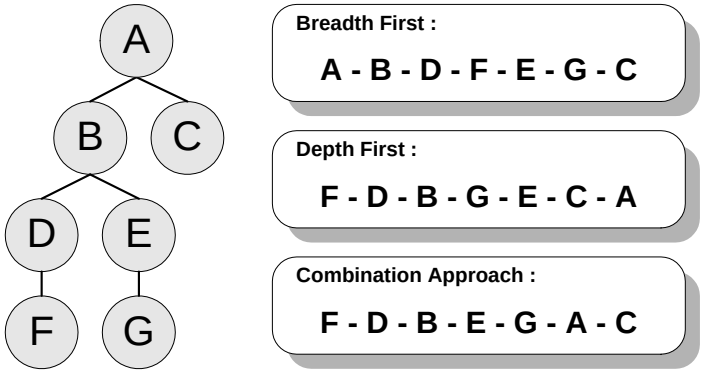

Figure 4.12: Retrieval Algorithms at Work

at the end of that branch, it performs the breadth-first technique for retrieving database entries down that branch.

Utilising Semantic Knowledge

We also investigated query optimisation at the application level. This involved modifying the client application that was sending the database retrieval requests. It was modified to change the order of the elements in the pattern (XML structure) sent to the storage component in such a way that the most unique values would be seen by the algorithm first, this meant that the initial set from which objects were retrieved from the database was minimised.

The interesting thing about this optimisation is that, unlike the algorithms, it requires knowledge about the structure beyond the simple XML hierarchy. Knowledge that, strictly speaking, should not be available in an arbitrary storage component.

4.5.3 Lessons Learned

In actual fact all of these retrieval algorithms were disappointingly slow. With the breadth first approach being particularly unusable as it took five times as long as either of the other two. Adding some semantic knowledge to the retrieval process did improve matters, but not enough to prevent us from abandoning the generic database retrieval of XML for practical reasons and relying on pre-caching of structures in memory instead. The exact results and a more complete discussion of the Solent System can be found in (Reich *et al.*, 1999b).

I believe that this experience with XML shows the power of a generic approach but also demonstrates the need to carefully examine a particular problem domain and engineer a solution that, while still as generic as possible, encompasses only those structures

that are needed and thus preserves the level of performance required in real systems.

4.6 Summary

In this chapter I have described the broadening coverage of the OHP suite of protocols and presented the work undertaken to define OHP-Service, a protocol for the dynamic discovery and invocation of information services.

This work raised some interesting questions with regard to the hypermedia functionality normally associated with a hypermedia system, i.e. the functionality enshrined in the OHP-Nav protocol. Consider the Follow Link operation from OHP-Nav, this could be implemented in the suite of protocols in one of three ways:

1. *An Operation* : Currently the Follow Link is implemented as an explicit operation on the data structures defined for OHP-Nav, as such it forms part of that protocol;
2. *A Query* : An alternative view states that since a Follow Link is effectively a process across structure then it should simply be implemented as a specific case of a general query language. Either such a language would be part of the OHP-Nav definition, or contained in a separate protocol for the generic querying of structure;
3. *A Service* : Arguably the Follow Link is opaque and thus should not be formed by such an open query mechanism. In this case an alternative to the operation approach is to encode a Follow Link as a Service and export it using the OHP-Service protocol;

In the OHP-Nav definition the Follow Link was included as an operation. In this way we left open the other two possibilities, such that a server could export a Follow Link Service, or a client could ignore any service or operation and undertake the Follow Link itself using a query mechanism.

Clearly however, there are systems based issues here that are separate from purely hypermedia concerns, such as whether we regard hypermedia as a delivery mechanism for Services as opposed to a target application. In addition since this work began, other, more general service discovery technologies have begun to mature. In particular, research undertaken in agent systems (Wooldridge & Jennings, 1995) points at a component based future where components undertake ‘meta-communication’ on some standard level entirely separate from application level concerns. For example Sun’s Jini project (Waldo, 1999) is very well placed to become the de facto standard for service discovery frameworks. It is in this general layer that such things as Progress and Feedback (discussed in Section 4.3.2) actually belong.

Placing the OHP suite into this larger picture of general services poses a problem that is only made worse by the fact that the scope of the original proposal has increased out of all proportion compared to its original intent. The OHP has grown from a simple protocol that would allow standardized clients to talk to any OHS into a mammoth undertaking that involves all of the components of a system and which includes multiple domains of hypertext and many levels of functionality. The resulting problems would seem to indicate that the scope of the protocol should be dramatically reduced.

The goals of the protocol had been moving since its inception and in my opinion it was time that they were re-examined. When OHP was conceived the OHS architecture considered was a client/server one, ideal for intra-LAN systems. As technology has moved on, and we move into an age of distributed information and ‘intelligent’ agents, it is possible that by placing the OHP functionality more accurately into a more general dynamic service discovery, or agent, framework, we may actually offload many of the problems.

In other words, by accepting that the scope of OHP is *specifically* the presentation and navigation of information, we no longer have to deal with any of the communication infrastructure issues that have caused this ‘feature creep’. Instead we have to build OHP on top of existing networks and prototype frameworks that support the dynamic exchange of knowledge between distributed components.

Chapter 5

Infrastructure and Communication

5.1 Introduction

In Chapter 3 and Chapter 4 I have described the progression of OHP into OHP-Nav and OHP-Service. This was somewhat of a natural evolution with new ideas incorporated in an ad-hoc fashion as they were considered, the best of which have remained in the formal definition. However, certain things have become evident as the protocol has been implemented and further considered.

Firstly there has been the continuous issue of the definition of OHP. This has moved from the purely message based document described in Appendix A, through a more well defined standard (XML) described in Appendix B to the implementation independent IDL described in Appendix C. However, this IDL still reflects a message passing idiom and does not utilise the distributed object advantages of the component frameworks that IDL is typically used with (such as CORBA, JavaBeans etc.)

In this chapter I shall examine the issues of syntax and infrastructure in an effort to explore *how* components might communicate. In particular I shall look at how the order of communication, discourse, is managed and examine how principles from linguistics, human conversation and agent-based technology might influence this development.

5.2 The Languages of Communication

It has already been mentioned that the purpose of OHP has changed a great deal since the protocol first appeared. Initially a basic client-server communication protocol it has grown to reflect the concerns of a large community of researchers. Even given that we understand what functionality we are going to standardize there is still the question of

what actually will become standard in the system. I.e. how will components actually talk to one another? There are two approaches (Millard *et al.*, n.d.):

1. *A programming API* : The OHSWG could decide on a standardised API that applications could use to communicate. This way source code compatibility is preserved and changes on-the-wire do not require applications to be re-written. This requires specifying:
   (a) the system calls,
   (b) the callbacks to be used,
   (c) the data to be exchanged.

   Examples of available systems include CORBA (and it's interface definition language IDL), Microsoft DCOM or Java component technologies (Java Beans). One disadvantage of the API approach is that the definition is dependant on the binding (i.e. a particular definition language and implementation of the communication module).
2. *An on-the-wire communication model* : The alternative approach is to define a message language in which components can converse. This involves defining:
   (a) the syntax of the messages (e.g. an XML hierarchy),
   (b) a set of requests, associated responses and their syntax,
   (c) the data and its syntax,
   (d) how to setup the transport medium (e.g. opening a socket on a port, etc.).

At one time or another both approaches have been argued for. However, the need to produce communicating systems that actually work has resulted in the group returning to the on-the-wire approach, even though the API approach seems cleaner and would allow us to concentrate on the hypertext issues rather than the networking ones. The API approach also preserves source code compatibility. The implementers just need to implement two APIs, one for the client and another for the server. If they then find that a new on-the-wire protocol should be used, they can change their implementation without altering the source code of the components involved.

However it is still not a perfect solution as it may require recompiling applications when a different medium is required. This is indeed the case with CORBA where binary applications are ORB dependent. This does not give a lot of freedom to the final user as a binary is typically compiled for a fixed communication medium.

As the on-the-wire communication model is typically adopted for Internet protocols (normally ASCII and socket based), there is a simple argument that says that since it works for the World Wide Web and the Internet it can work for OHP to. Unfortunately this approach has several disadvantages:

- Writing efficient socket communications is a very difficult task, involving threads, polling, etc. It is very easy to produce inefficient communication systems.
- Such libraries have to be rewritten for every application. This results in the risk of a bad implementation, where data is not properly formatted or parsed. The CORBA approach with a stub compiler avoids this problem by generating code automatically.
- It becomes extremely difficult to deal with non-protocol data and requests, such as routing information for mobile agents, garbage collection or session management.

Both approaches to interoperability have their advantages and disadvantages. An on the wire protocol has the ‘taste’ of the Internet community and a simplicity that is very appealing, while the programming API allows further techniques to be transparently added (i.e. mobility, etc.). In both cases, a data model has to be adopted. The data model specifies the type of data and its associated meaning (in terms of primitives) exchanged during communications. The data model does not specify the syntax of data (this depends on the approach: e.g. XML over sockets).

Even given a data model and some communication medium, there remains the need for some type of infrastructure over which that model can be discussed by a variety of components. This infrastructure is different from the network and itself may run independently over different lower network protocols (sockets, rmi, etc.). It could be implemented either in a message passing form or as an API. In effect it is a framework in which components can discover each other and exchange data.

OHP-Nav, as described in Chapter 3, managed to achieve a framework effect via a standard message header and an understanding about the transport layer across which the messages flow (in this case the number of header bytes on a TCP/IP stream). But at the Hypertext ’98 demonstration it became apparent that this was not powerful enough to deal with the communication requirements between components. For example, there was no way to register your existence or notify others of events.

What was needed was a communications layer below OHP that would add some communicative context to messages. If this was defined separately from OHP-Nav then it may be shared by the entire OHP set and therefore allow components of many different domains to converse. E.g. an OHP-Nav client could talk to an OHP-Space server.

We basically envisaged a layered protocol architecture for OHP as shown in Figure 5.1.

This involves taking the current definition and dividing it into two parts (the upper two layers). The uppermost layer is the content of the message, an object from the data

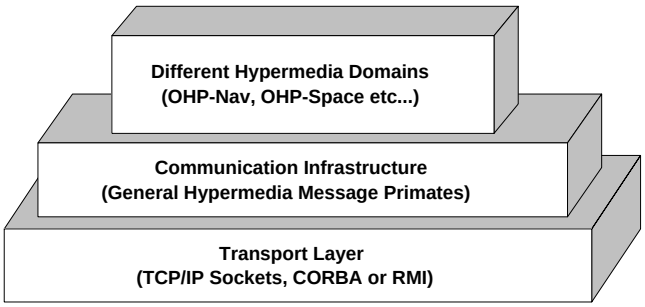

Figure 5.1: Layered Protocol Architecture

model and an operation to apply to it. The middle layer is the message header and cradle for the operations; this is also the place to define composite messages, where multiple operations are sent in a single communicative act.

I believe that the argument between the message based approach and the API based approach is concerned with this middle infrastructure layer. Since we have concluded that both approaches have positive and negative aspects, the choice between them seems a mute point. Instead we should look at what we can expect from this framework. For example, do we require it to be asynchronous? Most importantly, is it just a transport mechanism for managing 'conversations' or does it contain a set of semantics of its own, i.e. do different communication primitives convey any meaning about their content?

5.3 Communication and Meaning

The communication framework, or infrastructure, described above poses different problems then when we were concerned with a single protocol, as it has to be able to convey meaning about a wide range of topics, of which OHP-Nav is only one. Inspiration can be drawn from the world of human communication. We do this with the aim of rising above current idioms of message passing and function completion, to a more powerful view of sophisticated components 'discussing' data.

The first thing we must do is to examine how meaning is conveyed in a conversation.

5.3.1 Semiotics

Semiotics is the study of signs. More particularly it is the study of anything that stands for anything else. This includes words, images and sounds. It can be broken down into three further subjects:

- *Syntactics*. The analyses of the structure that exists between signs.
- *Semantics*. The meaning of signs (there relationship to what they represent).

- *Pragmatics*. The way in which signs are interpreted.

The relationship of semiotics to hypertext is only recently being explored (Neumüller, n.d.). In particular the fact that links can be seen as references or signs that represent their destinations. However, we are concerned here with the relationship semiotics holds with communication; conventionally communication between human beings but equally as applicable between components of a system. In this case we can regard syntax as being beyond the scope of our exploration, as it very much dependant on the communication system being used. However, we can look into the semantic and pragmatic considerations of OHP and try to learn from the semiotic field.

Signification

Signification is regarded as the process of creating and interpreting signals. This involves two parts:

1. SIGNIFIER. The sign itself, which may be one of three types:
   - *Icon*. There is similarity between the signifier and signified. E.g. a portrait of a person.
   - *Index*. The signifier is closely related to the signified. E.g. in a causal relationship such as smoke being the index of fire.
   - *Symbol*. A conventional link exists between the signifier and signified. E.g. insignia that denotes military rank.
2. SIGNIFIED. The object represented.

Words may be thought of as verbal symbols. In a similar vein the hypertext objects of component communication can be thought of as structural symbols, perhaps even icons, as they often reflect the informational make-up of the real or virtual objects that they signify. However it is important to remember that semantics, the meaning of signs, is not only about signification.

Meaning

The referential nature of language would seem to indicate that the meaning of a proposition is encompassed in the references it contains, the signifiers mentioned above. However, there are examples that show this to be incorrect (the following examples are based on (Saeed, 1997)).

Firstly, consider the more familiar case of human language. Here, there are definitely words that refer to real objects. Proper nouns denote individuals, verbs denote actions

etc. However, there are plenty of words that by our referential standards would have no meaning; words such as ‘and’ or ‘but’. However, it remains obvious to us that these words do have meaning.

The second problem is that it is possible in real language to refer to things that do not exist, something that is still an issue in hypertext. Consider a sentence such as:

‘The Dragon was slain by St. George.’

In this case there is no real world referent for *‘The Dragon’* only other signifiers. Despite this it is obvious that the sentence above makes sense and that people have a notion of ‘Dragon’ even though they have never seen one in reality.

The third problem is that several expressions share the same referent but have different meanings. For example ‘Polaris’ is also known as ‘The North Star’. However the two sentences below clearly have different meanings, the second being a tautology:

‘Polaris is the North Star’

‘Polaris is Polaris’

All this points to the fact that there is more to meaning than reference. A complete study of meaning is beyond the scope of this thesis. However, we can accept that an extra dimension of meaning does exist and we can refer to this as *sense*.

What is interesting is to examine the way in which we convert the concepts that humans hold as thoughts, or that software components store as data, into words that convey some sense.

As Saeed observes, “It has become clear that meaning is richer than language at both ends, so to speak, of the communication process. Speakers compress their thoughts, and hearers fill out their own version of the intended meaning from the language presented to them.”

To do this hearers must have some understanding of not only the communication process but also the context in which discourse occurs.

5.3.2 Contextual Discourse

To help understand a particular utterance, a listener often has to perform one of several contextual tasks (Saeed, 1997) :

1. Fill in deitic expressions (words or phrases whose meaning changes from one discourse to another)
2. Fix the reference of nominals
3. Access background knowledge
4. Make inferences

The question is, which of these is applicable to component communication? We will now examine each task in turn:

Fill in Deitic Expressions

Deixis are words or phrases whose meaning depends on the perspective of participants in a discourse, they must be evaluated in terms of the context of that discourse to make *sense*. For example:

- *Spatial Deixis*. This is the implicit division of space around a speaker. E.g. 'It's too hot *here* lets go over *there*'. In software components' conversations these references become less useful (unless we are considering mobility), however they might be employed either in a physical or virtual way in the future.
- *Person Deixis*. The roles of participants. E.g. '*I* was asleep while *you* were working'. In its current form OHP acts as a conversational tool between only two components at a time. However a conversation between multiple participants could well be valid (for example in the collaborative field described in Section 4.2.5) and in these cases shorthand references for other participants would become important.
- *Social Deixis*. The social identities and relationships of participants. E.g. Familiar versus polite pronouns such as *tu* and *vous* in French. Although the social roles of components is not explicitly explored in the OHP suite, there has always been a notion of client and server. In addition other work has identified roles that govern competitive verses cooperative behaviour in component systems (Gibney & Jennings, 1998; Lesser, 1991) and social deixis could provide cues in a discourse that help each component establish their role.
- *Textual Deixis*. Orientation within the discourse itself. E.g. '*At this point* we have to re-examine the evidence'. This is analogous to the management of messages being sent between components. For example the message identifiers that allow referencing of the discourse itself and thus provide a framework for session management.

Fix the Reference of Nominals

This essentially involves the resolution of references. In language, where metonymy (associated referencing) and synecdoche (parts representing wholes) are common this can be a complex problem. The software world offers unique identifiers as a solution, but this itself builds to form a complex naming problem (as discussed in Section 4.2.3).

Access background knowledge

Knowledge can act as a context for the understanding of utterances. In discourse there are three types of knowledge:

1. knowledge computable from physical context
2. knowledge available from what has been already said
3. knowledge available from background/common knowledge

Without a true virtual world, or interface to the real world, there can be no knowledge computable from physical context. However it is certainly true that software communications could include references to what had already been said (this is done in OHP via the message identifiers) and that communicating components share mutual background knowledge described in a common vocabulary.

Make Inferences

As described in Section 5.3.1 listeners actively participate in the construction of meaning by making inferences to preserve coherence in what they are told. Importantly, speakers are aware that their listeners will do this and take advantage of it to speak less explicitly than they otherwise might. E.g. 'I'm sorry I'm late, the train was delayed.' The listener can infer that the speaker was on the train and that is the reason why they were late.

It is perfectly feasible for components to make inferences based on what they are told. E.g. a component tries to create a node but is told that they do not have write permissions on this hyperbase. The component can infer from the reply that their creation request failed.

Inference is possible because of the 'co-operative principle', a tacit agreement by speakers and listeners to cooperate in the process of communication (Grice, 1975). This principle lays down four maxims:

1. *Maxim of Quality* speakers try to make their contributions true
2. *Maxim of Quantity* speakers try to make their contributions as informative as required for the current exchange. Not more or less so.

3. *Maxim of Relevance* speakers try and make their contributions relevant
4. *Maxim of Manor* speakers avoid ambiguity and obscurity and make their contributions brief and orderly

These maxims form a base-line for talking. Software components often make the same assumptions and we should embrace these as an opportunity to gain more expressive and flexible communication, where by not only allowing the listening components to make inferences but actually expecting it, we actually could impart more knowledge more elegantly than with more explicit but stilted communications.

5.3.3 Performatives

Performatives, or speech acts, are actually abstractions in the field of linguistics. The terminology was introduced by Austin (Austin, 1975) and defines the process by which people distinguish different types of communication (e.g. questions vs. statements), effectively a special class of actions that correspond to the basic building blocks of dialogue. The view that all utterances are in fact speech acts has led to the popular belief that there are two basic parts to meaning:

1. the conversational meaning (or proposition)
2. the speakers intended speech act

For example, take the following four sentences:

1. David is writing his thesis
2. Is David writing his thesis?
3. David, write your thesis!
4. If only David would write his thesis!

These sentences contain the same proposition but are altered by their speech acts, observe the following breakdown into their corresponding parts:

1. DAVID IS WRITING HIS THESIS + *declarative* = statement
2. DAVID IS WRITING HIS THESIS + *interrogative* = question
3. DAVID IS WRITING HIS THESIS + *imperative* = order
4. DAVID IS WRITING HIS THESIS + *operative* = wish

This is obviously of interest to computer scientists involved in computer communication as by applying different performatives to the same operations the meaning of those operations could be altered, creating a powerful vocabulary of actions. An added advantage is that if all components understand the performative operations then they could

draw some meaning from a message even if they did not understand the accompanying operation. For example a component might not understand a particular creation request, but it still understands that it is a request of some sort. This could be useful for returning sensible error messages to unknown operations and enabling components to forward messages intelligently whatever their contents.

OHP Performatives

Sigi Reich and I tried to apply this philosophy to the OHP suite. For inspiration we investigated KQML (Labrou & Finin, 1997), and FIPA's Agent Communication Language (FIPA, 1997), existing performative based standards. Both of these define a set of general primitives (or performatives) that define how communication takes place rather what is actually communicated.

Both the FIPA and KQML ACLs define a large set of possible performatives, these can be found listed in Appendix E. Taking these sets as a starting point we created a list of performatives that were relevant to the hypertext components that OHP-Nav was envisaged to serve. We implemented these performatives as a part of a general OHP message header that adds communicative context to the OHP-domain messages. This was the message header then used with OHP-Nav 1.3 and the Solent system (see Section 4.4). There were twelve performatives in all:

Registering:

register Informs another component of this component's existence. Message body is empty.

unregister Informs another component that this components is leaving or shutting down. Message body is empty.

Requesting Operations:

ask Return all results of this operation in one message.

ask-stream Return all results of this operation in a series of messages, ended by an *eos* message.

eos End of stream marker for ask-stream request. Message body is empty.

cancel Cancel the ask request referred to by the reply-id tag in the message header.

Subscription to Messages:

advertise Lets other components know that this component could inform them of all messages of the same type as the message body sent with this performative.

unadvertise Lets other components know that this component can no longer inform them of all messages of the same type as the message body (which must previously have been advertised)

subscribe Sets up a subscription where the recipient informs the sender of all messages of the same type as the message body (which must previously have been advertised).

unsubscribe Cancels a previous subscription set up with the 'subscribe' performative

We considered it important to include an advertisement mechanism. Without it any component can subscribe to the activities of any other component. This makes it very difficult to write light components. However, if you include the advertise performatives then any component that does not wish to deal in subscribe messages merely ensures that it advertises nothing.

Notification:

notify Notifies the recipient that the operation in the message body has occurred as a result of the message in the reply-id.

Error Handling:

error Notifies the recipient that an error occurred (reason and diagnostics is given in the domain specific message body)

There was a great deal of disagreement within the OHSWG as to whether performatives should be added to the OHP-Nav definition. While Sigi Reich and I were convinced that they added value, particularly on a message management level (such as subscription and streaming), other members of the group thought that they only added an unnecessary layer of complexity and were in many cases only re-stating semantics that were already present. In addition they believed that many combinations of messages and performatives were meaningless (such as an *ask* performative combined with an *Error* message).

In an effort to keep the OHP definitions as simple as possible performatives were never formally added to the OHSWGs description of OHP-Nav. However I have never lost the belief that they are useful. In fact I now believe that the disagreements within the group regarding performatives were a symptom of a much larger and fundamental problem. In the OHSWG definitions of OHP and OHP-Nav we have never separated the propositional content of the messages from the communication part. In other words, OHP does not assume a communicative infrastructure but attempts to implement one itself.

As a result of this confusion, many of the OHP-Nav messages already enshrined particular speech acts within them (for example, a *CreateNode* message is already implicitly an *ask* request). Without a 'pure' set of propositions, it is impractical to apply a performative set. Hence the group's problems with duplicate semantics and meaningless combinations.

5.4 Summary

In this chapter I have examined the different approaches to communication and turned to the fields of linguistics and agent technologies to provide some suggestions for future direction.

In particular I have concluded the following about interoperability and communication:

1. A Communication Infrastructure is needed to allow the OHSWG to concentrate on the contents of communications rather than their packaging.
2. This infrastructure might be implemented as an API or a message based protocol. Although each approach has positive and negative aspects these are separate to issues concerning the way in which conversations are conducted.
3. We could learn much from human conversational technique to help express meaning within this infrastructure, including the use of inference and contextual discourse.
4. Performatives allow content to be expressed as single propositions and help to form a concise, flexible and powerful infrastructure layer.

Having decided that it would be best to define OHP 'above' an existing performative based infrastructure it is now necessary to examine exactly what is to be the subject of communication; both to construct a common vocabulary of propositions but also to reconcile the various hypertext domains, or at least understand them in common terms.

Chapter 6

The Information Continuum

6.1 Introduction

In Chapter 5 I described how a proposition based communication infrastructure could form a basis for an interoperability protocol. Having addressed the issue of *how we talk* it is now time to turn to the issue of *what we say* and look at the semantic content of the propositions themselves.

This is necessary as over time there has been a gradual diversification of the OHP. Initially this was manifest via the inclusion of opaque 'Script' type objects that could be used to implement any system's particular peculiarity. This was extended to the notion of Services, and I spent some time exploring this idea and defining Service objects (as described in Section 4.3). However the idea that Hypermedia might exist in several different domains (OHP-Nav, OHP-Space etc.) has brought forward the notion that Navigational Hypertext is itself only one Service amongst many.

In this chapter I shall examine the issues of semantic content. In particular I shall define more precisely what is meant by Navigational, Spatial and Taxonomic Hypertext and examine how these different hypertext domains can be incorporated into a single information world view and what that means for our current ideas of linking.

6.2 Hypertext Domains

The realisation that a communications infrastructure releases OHP from tangential concerns about communication, leaves us free to concentrate on the propositions themselves. Effectively a semantic language in which to discuss hypertext. We already have a separate data model for OHP (see Section 4.2.2) and it could be argued that this data model

forms a common vocabulary for hypertext. But when one looks at the breadth of hypertext work it becomes clear that, although powerful, the OHP model is not expressive enough.

Hypertext is often seen as a way of navigating information spaces by selecting hotspots in documents which move the user through the information space, with the World Wide Web serving as a prominent example. However, there are many more hypertext domains, each with their own specific needs and requirements, which are not served by the data model. These domains include spatial hypertext, argumentation support, taxonomic hypertext, hypermedia art, hypermedia literature and others (for an overview see Nürnberg, 1997 (Nürnberg, 1997)).

These hypertext domains and the systems supporting them rely on different conceptual models (Dyke, 1991). However the ultimate overall objective common to all these conceptual models is to help users understand and engage with information via navigation (Thüring *et al.*, 1995). Different criteria for measuring this objective may apply, e.g. metrics borrowed from software engineering such as ‘cohesion’ and ‘coherence’ (Lowe & Hall, 1999), and the cognitive requirements of these domains differ, but common ground does exist and can be explored.

In the following sections we will briefly describe some of the most important domains and reflect on their specific properties. The domains described are Navigational Hypertext, Spatial Hypertext and Taxonomic Hypertext.

6.2.1 The Navigational Domain

Navigational hypertext is concerned with partitioning information spaces into nodes and establishing relationships between them such that users can move their applications’ ‘view’ between them. Links store connections between ‘hot spots’ in documents. By clicking on one hotspot the user navigates to the one at the other end of the link.

Navigational Hypertext is probably the oldest conceptual model of hypertext, envisaged as it was by the pioneers (Bush, 1945; Engelbart, 1962; Nelson, 1967). OHP was designed initially to operate within this domain, reflecting the functionality of many of the systems developed at that time. Chapter 2 gives a comprehensive overview of those systems.

The model of navigational hypertext presented here follows closely the data model specified by the OHSWG (Reich *et al.*, 2000) and detailed in Section 4.2.2. It is based firmly on the definition of several important objects that make up a link structure. Consider the two links shown in Figure 6.1.

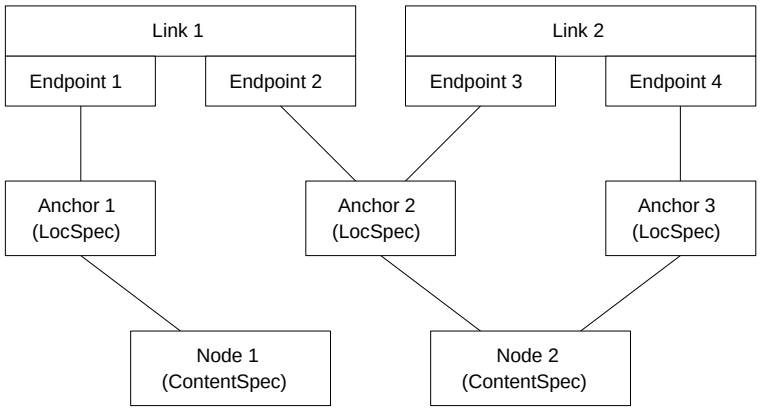

Figure 6.1: The OHP Node Link model

In this diagram there are two node objects. These represent the systems notion of a document or a file. Each node may have several anchors associated with it. These are objects that define a region inside the node to use as a hotspot (for example there could be an anchor from word twelve to word seventeen). The endpoint objects bind an anchor to a particular link. As an anchor can be bound to several links (see *Anchor 2* in the Figure) the endpoint contains all the information that is relevant to this anchor in the context of one particular link (such as its direction).

Operations applied to objects include creation and deletion primitives for all the objects in the system as well as a Follow Link function that returns a set of endpoints based on a single input endpoint parameter according to the underlying link structures. As described in Section 4.6 this may be more then merely a query on the structures stored.

Navigational Hypertext was thought by the pioneers to mimic the associational way in which human beings think and reason. As a result navigational systems tend to be ad hoc and unstructured. This leads to many of the common problems associated with hypertext, such as cognitive overload and disorientation (Conklin, 1987). There is even one school of thought that describes ad hoc linking as harmful to the comprehension of information (Young, 1990).

6.2.2 The Spatial Domain

The opposite approach to ad-hoc linking, Spatial Hypertext relies on the organisation of nodes into logical composites or spaces, in a process referred to as ‘Information Analysis’ (Lowe & Hall, 1999) or ‘Information Triage’ (Marshall & Shipman, 1997).

While Navigational Hypertext is concerned with links between nodes in an unstructured information space, Spatial Hypertext allows movement only by following the structure that makes up that information space. This restricts the movement but allows a user

to become familiar with areas within the space and organise themselves accordingly. The key characteristic is to leave structure implicit and informal (at least as presented to the user) (Marshall & Shipman, 1995).

Relationships between nodes are simply expressed by their visual characteristics such as spatial proximity, colour or shape. This results in some interesting properties. If for instance a node is slightly misaligned with other nodes then this might express an uncertainty about whether this node is actually part of this relationship. In other words it expresses classification within relationships, where some nodes are 'more' related then others. Spatial hypertext systems are therefore inherently flexible.

Examples of spatial hypertext systems include VIKI (Marshall & Shipman, 1995) and CAOS (Reinert *et al.*, 1999). The following conceptual model summarizes spatial hypertext:

1. Nodes are visual symbols serving as wrappers to documents, therefore they have characteristics such as colour, location and shape.
2. Nodes can be aggregated (visually) thus building collections of related objects.
3. Composites are the visual representations of these collections, they also have visual characteristics and therefore can also be aggregated into other composites, although this relationship may not be circular.
4. These collections are typed, i.e. they may form lists, matrixes, sets, stacks etc. This effects their visual presentation but also acts as an organizational aid, adding both order and internal structure to the collections (i.e. one node may be placed before another, or the proximity of one node to another could reflect the strength of their association).

Any spatial system has to make all of these relationships explicit in the system so that queries can be made of the information and that visual information can be stored economically. To this end many systems use spatial parsers to convert the implicit spatial relationships manipulated by the user into explicit associations within the system. It is this parser that recognizes the way in which nodes have been lain down and decides on an appropriate structure to store the information such as a list or a set.

Spatial systems are also unusual in that they hold the potential for fuzzy membership of a composite. For example take Figure 6.2. Here, documents describing different dangerous animals have been organised into a 'dangerous animals' space. Colour is used to represent exactly how dangerous each animal is, i.e. a rabbit is not very dangerous, while a human or a tiger are extremely dangerous. Interestingly, it is possible to have different

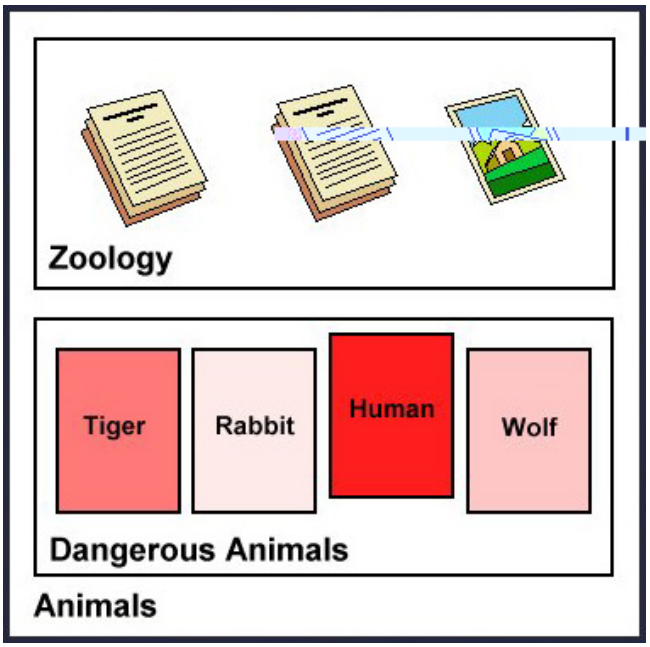

Figure 6.2: Fuzzy Membership of a Composite

degrees of fuzzy membership within the same composite. In this instance the slight misalignment of the human composite indicates that we are unsure as to its membership of the dangerous animals set.

Operations on objects include creation and deletion, adding and removing objects to composites. In addition to reduce the complexity of the users view, spatial systems often restrict the depth into the spacial hierarchy that a user can see. Thus there is also a 'zoom' function that allows a user to zoom into a composite's sub-components, unveiling new depths of structure.

6.2.3 The Taxonomic Domain

Taxonomic hypertext applies hypertext concepts such as non-linear information access and individualized views to the domain of reasoning about taxonomies, e.g. in biology, linguistics and other application areas (Nürnberg *et al.*, 1996; Dyke, 1991; Dyke, 1993).

For this kind of knowledge task a model based on set theory is used (Dyke, 1991). Users sort artifacts (the equivalent of nodes) into categories, based on their characteristics, forming a taxonomic hierarchy. Different users may have different views of how the artifacts are partitioned. Taxonomic reasoning is the process of moving around the hierarchies by crossing the boundaries between overlapping sets.

Figure 6.3 gives an example of a taxonomy. Here two people have categorized three artifacts, in this case documents about three species; lions, platypus' and ducks. They both agree that all three lie within the category of 'Animals' but they disagree on how

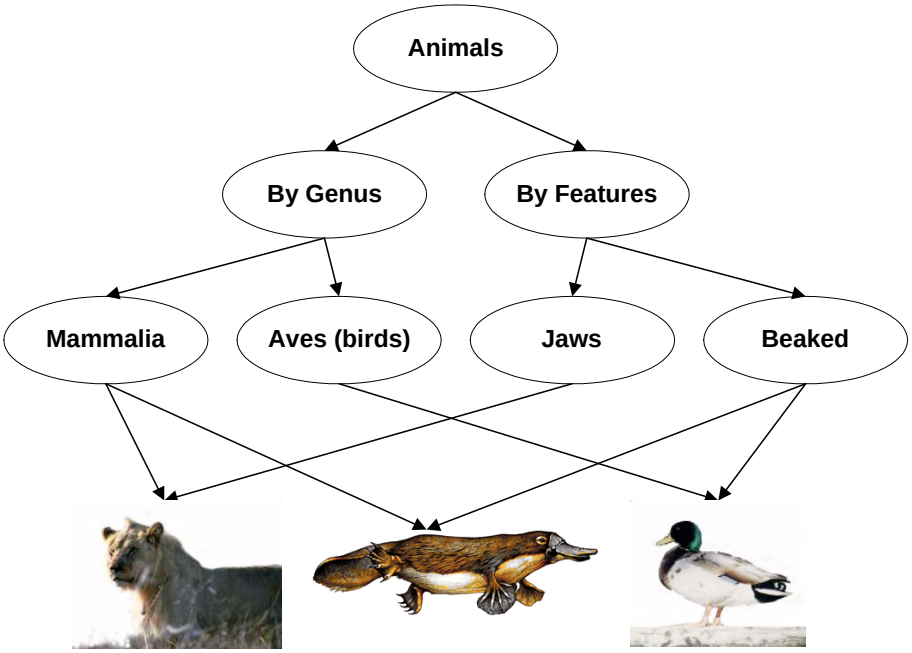

Figure 6.3: Example Taxonomy

they should be further sub-divided. The first view is that this should be done by Genus, grouping the lion and the platypus together as mammals and the duck on its own as a bird. The second view disagrees and believes instead that they should be categorised according to their features. So the platypus and the duck are grouped together, as they have beaks, and the lion on its own as it has jaws.

There are two important rules that govern the shape of taxonomies.

1. Whenever a taxonomy splits via perspectives the same artifacts can be reached down each branch, it is merely their categorization that changes.
2. All categories are single parented within a single taxonomy (i.e., an artifact may be in two categories only when each parent category is within a different taxonomy as demonstrated in Figure 6.4).

Artifacts can be added to categories and also removed. Categories are organized hierarchically so categories may be added and removed as well. Categories cannot be circular, as this is semantically meaningless (Dyke, 1991). In addition some set like operations are also required so as to reason about multiple taxonomies, such as set union and intersection.

6.3 Information Spaces

The partitioning of systems into hypertext domains, as described above, seems very natural, as does the division of OHP into a suite of similar protocols. In Chapter 5 I described

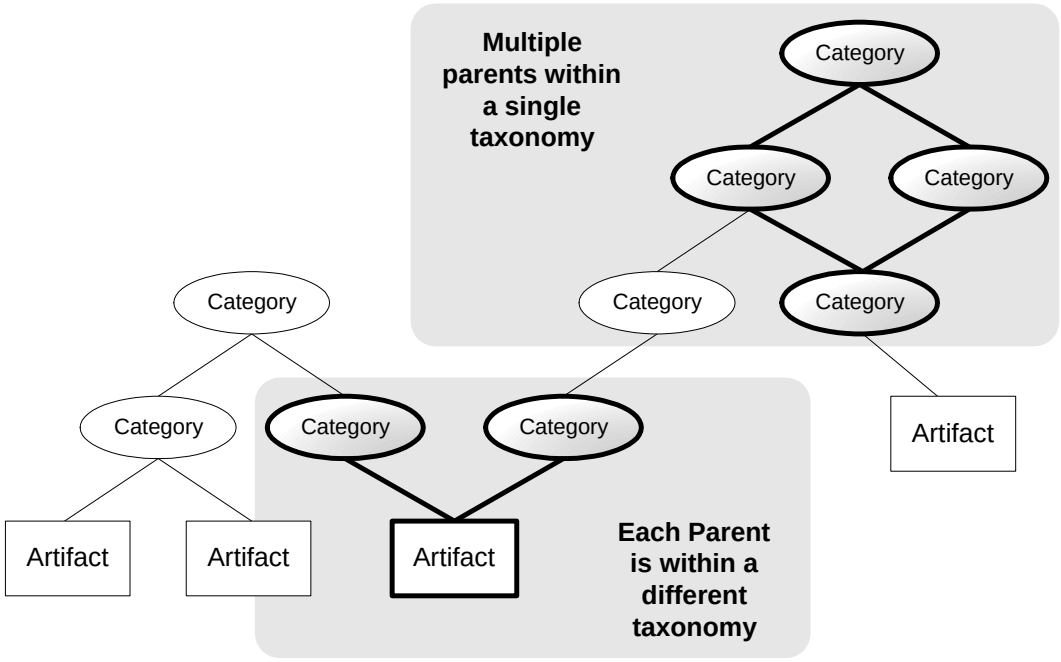

Figure 6.4: Rules for Taxonomy Parenting

how all the protocols within that suite should operate over a common infrastructure. The question I would like to tackle now is whether or not those protocols should in fact operate over the same information space, which itself calls into question whether or not the division of OHP into separate domains is necessary at all.

To begin answering these questions we need to consider to what degree these information spaces actually overlap? For example, an early observation is that the concept of a node and the resource it wraps is universal to all domains. But to what extent are the other objects in each domain compatible with each other?

Further questions arise when we consider the notion of context (see Section 2.8.3). If context partitions an information space how does this effect the browsing of that space. Do contexts themselves form part of the space and are they analogous to objects in the various domains?

To answer these questions it is important to begin to understand the information spaces that we are modelling and crucially look at how they relate to one other. This will help us discover if the boundaries we have drawn between the different OHP protocols, reflect the reality of the information spaces or if they are artificially imposed.

6.3.1 The 6D Model

To start understanding the information spaces it is helpful to visualise them. Figure 6.5 shows a three dimensional rendering of a single finite information space. This rendition

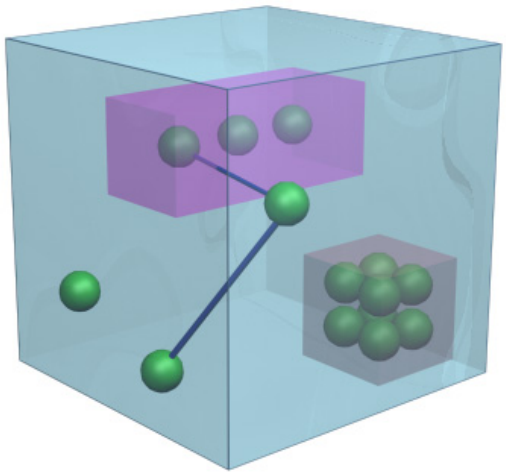

Figure 6.5: Visualising an Information Space

is analogous to that of a Spatial Hypertext system, but extended logically into the third dimension. Nodes are represented as spheres, grouped together in composites within the space (with colour representing the structure of the composite, i.e. lists, matrixes etc.). Links are shown as lines between the nodes. Notice that they are in addition to the composite structures and may pass seamlessly through them.

It is important to distinguish between our three basic rendering dimensions and a three dimensional reality. In the case of the rendering space it is possible for an object to exist in more than one location, i.e. a node may belong inside more than one composite. Indeed in many systems a node may exist several times within a single composite, for example a slide may appear several times within a single presentation.

With some thought it soon becomes apparent that such a representation is incomplete. What about situations where the structure around the nodes varies according to context? For example in adaptive hypermedia systems different links may become visible on a users second visit to a node than on their first. In addition how do we represent the change in the system over time and how do we represent the different domains? In fact we need to extend the dimensionality of our model to include these other aspects.

As our rendering space occupies three dimensions already we must resort to a hyper cube in order to represent the other axes. Figure 6.6 shows the six apparent dimensions of hypermedia by displaying them in a six dimensional hyper cube. In this hyper cube each of the inner dimensions, the three sides of cubes 1, 2, 3 and 4, represent the 3D rendering spaces. The three extra dimensions represent behaviour, time and context respectfully.

A user's view on a system depends not only on what area their viewers are currently

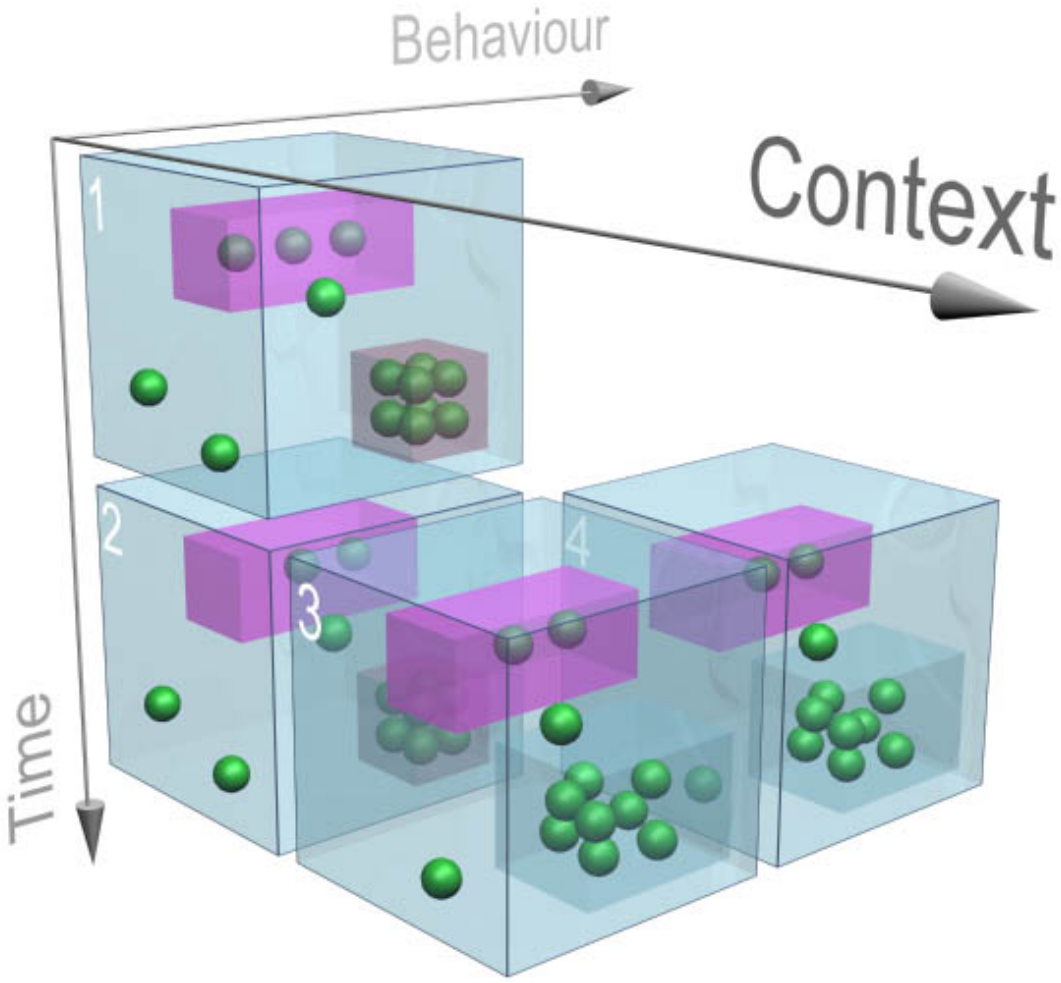

Figure 6.6: The 6D Model of Hypermedia

looking at (their position within a rendering space) but also on their position on the other three axes:

- *Time*. This represents how the space has changed over time. In Figure 6.6 we see that in the past (space 2) the list only contained two nodes rather then the present three (in space 1). This axis essentially represents versioning in hypermedia. The model is not concerned about how that versioning is done, or even at what granularity, only that it is there.
- *Behaviour*. The same space could exist elsewhere with different behaviour. This axis represents an information space spread across different types of systems, or domains of hypertext. For example, space 3 could exist within a Navigational system while space 4 could exist within a Spatial System.
- *Context*. This represents the different contexts that you could view the same space in. In the figure a user in one context would see space 2 which contains a list and

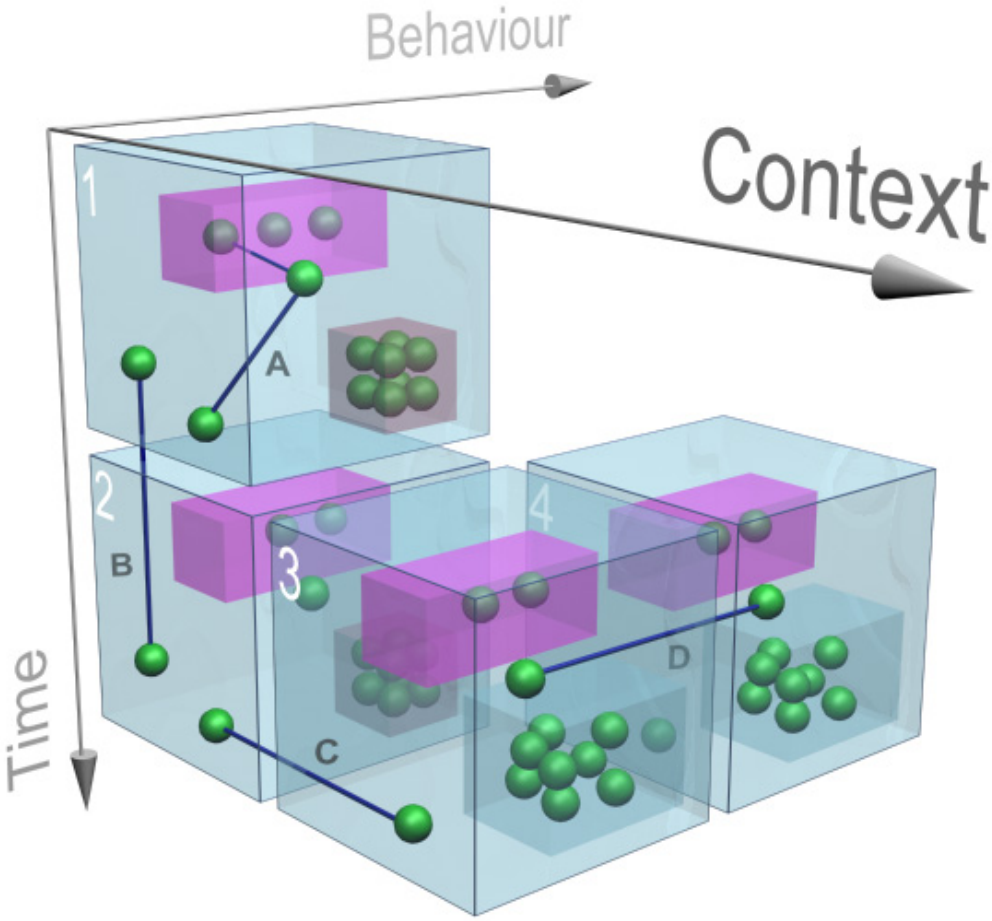

Figure 6.7: Cross Dimensional Links within the 6D Model

a matrix while in another context they would see space 3 which contains a list and set.

It is important to remember that these dimensions can be mutually exclusive for some spaces. Most existing navigational systems for example do not have a representation inside a spatial one. A more interesting example is the consideration of mutually exclusive contexts. In this case each context could represent a different linkbase (Crowder *et al.*, 1997), maybe on a different machine.

Figure 6.7 shows the same information spaces but this time also includes four different types of link, drawn as lines between nodes:

- *Link A - Across Space*. This is a normal link that takes a user from one point in a rendering space to another point in the same space.

- *Link B - Across Time*. This is a link that takes a user from a point in rendering space to the same point but in a past or future rendering space. It could be compared to moving up or down a version history.
- *Link C - Across Context*. This is a link that moves a user from a point in rendering space to the same point but in a different rendering space. It is similar to Parunak's moving between overlapping sets, where the set in question is a perspective.
- *Link D - Across Behaviour*. This is a link to a different type of system. E.g. following a link from an OHS to the WWW or from a Spatial to a Navigation system.

Combinations of these types of hyperspatial navigation are also possible. For example linking across time and space, or linking across context, space and behaviour. Strictly speaking a link causes the user to navigate to a node, which may subsequently exist in several places within a rendering space, but for purposes of clarity only a single line is shown on the diagram.

Unfortunately this six dimensional model is a simplification. In actual fact there are *n* context dimensions, which possibly include time and behaviour as well as *n* rendering dimensions (limited to three here for simplicity). As an example an employee's view on their companies information could change according to their technical ability, time with the company, current position, security clearance etc. Each of which forms a new context axis. To specify their view on the information a user has to specify their position on each axis.

In devising this model we have made some basic assumptions. We have assumed that information structure in one domain is mappable (although perhaps only in a limited fashion) to another. We have also assumed that it is possible to create links across all dimensions although not necessarily *within* all information spaces.

We have thus changed our notion of the information spaces. Rather than thinking of them as independent domains along the behavioural axes we now consider them part of a larger continuous structure represented by all the spaces in the 6D Model. Indeed, each contextual information space also forms part of that structure and allows us to tackle issues like versioning and context uniformly across all domains.

The union of all these information spaces is therefore both *exhaustive*, meaning it contains all information and domains, and *consistent*, meaning that all the spaces exist simultaneously. I call this super space of spaces the *Information Continuum*.

6.3.2 The Information Continuum

There are two important things that arise out of this consideration of a single continuum of information:

1. Navigation is common to all information spaces and hyperspatial linking (i.e. linking across the dimensions of the continuum) is possible.
2. Context is common across all domains (along the behavioural axis). As it is a common problem it should be possible to tackle it in a consistent way.

This means that we have to consider very carefully exactly where in our continuum the link and context objects exist. We will now consider several options:

Links: Are Links and Composites the Same?

This question results from the observation that the structure of a link and the structure of a composite are almost identical (i.e. a list of member objects). In Figure 6.6 links are shown separately from composites (lines rather than spaces). In this case they can be considered as 'wormholes' through the hyperspaces, connecting one part of the continuum with another. On the other hand if links are composites they form part of the information itself and can contain other links and themselves be the destination for navigational activity.

Although the 'wormhole' representation of these links may seem initially appealing, there is a danger in enshrining the existence of links purely in the properties of the system. For a start it removes the link from the information space itself, and makes referencing that link within the hyperstructure impossible (i.e. you could not create a link that anchored on another link).

Given that experience necessitates links to be first class (and thus referenceable objects) we are left with the question of whether a link object is the same thing as a composite. If it is not, then our navigational data model still lacks the notion of a composite. Halasz identified proper support for composites in hypermedia as one of his seven issues (Halasz, 1988) and it seems only proper that such support is provided in any model.

It helps us to consider the consequences of using links to represent composites. For example what happens to the added structure that a composite contains (i.e. what does it mean to follow a link that is actually a list as opposed to a set)?

Adding structure to links would effect navigation at two levels, at the point of *traversal* and also at the point of *arrival*. For example, traversing a link that is structured as a list may return only the next endpoint in the link rather then all endpoints in the link.

Perhaps this is a nice way to express Bush's trails (Bush, 1945). In addition, if the user arrives at an endpoint that referred to a composite then the destination reached may change according to the structure of that composite. Linking to a list, for example, may result in the first item in the list being returned, while linking to a set would result in all the items being returned.

Thinking of a link as a composite seems to make sense, after all, a link represents a relationship between several items, which is exactly what a composite is, and has the added benefit of being typed and navigable. Perhaps we should say that a link is a particular type of composite (i.e. one with direction).

We can now turn our attention to the idea of a context object and how that fits into the continuum.

Contexts: Are Contexts and Composites the Same?

The initial intention with OHP-Nav was to implement context as a list of node IDs; therefore a context could be represented as another type of composite or link object. As a result we can ask the question of where contexts exist in the hyperspace. Are they a part of the information space as well?

Thinking of context in this way seems intuitive, however it is probably more accurate to think of context as being implicit in the system. Users don't actually see a context; they are *within* a context. It is like a filter on their perceptions of the information space.

In which case it is actually not necessary to hold contexts explicitly as objects at all. Instead, what is required is for each object in the system to hold a number of contextual properties. When a user browses the system their contextual requirements will intersect with these properties in such a way as to appear to produce a data set of viewable objects.

This would work if context were merely a mechanism for the inclusion or exclusion of objects. However context is more than that, as a single object may have different properties according to the context in which it is viewed. For example, consider a system containing information on 'movie monsters'. In a context designed for children, objects might resolve into cartoon representations of the monsters, while in a context designed for adults those *same objects* resolve into movie clips of the 'real' monsters. This introduces the notion of a *concept*, the idea that several objects in the system actually are facets of the same object but in different contexts.

Since a concept is a set of objects where each object exists in a different context it can be thought of as a link across the contextual dimensions of the continuum. In terms

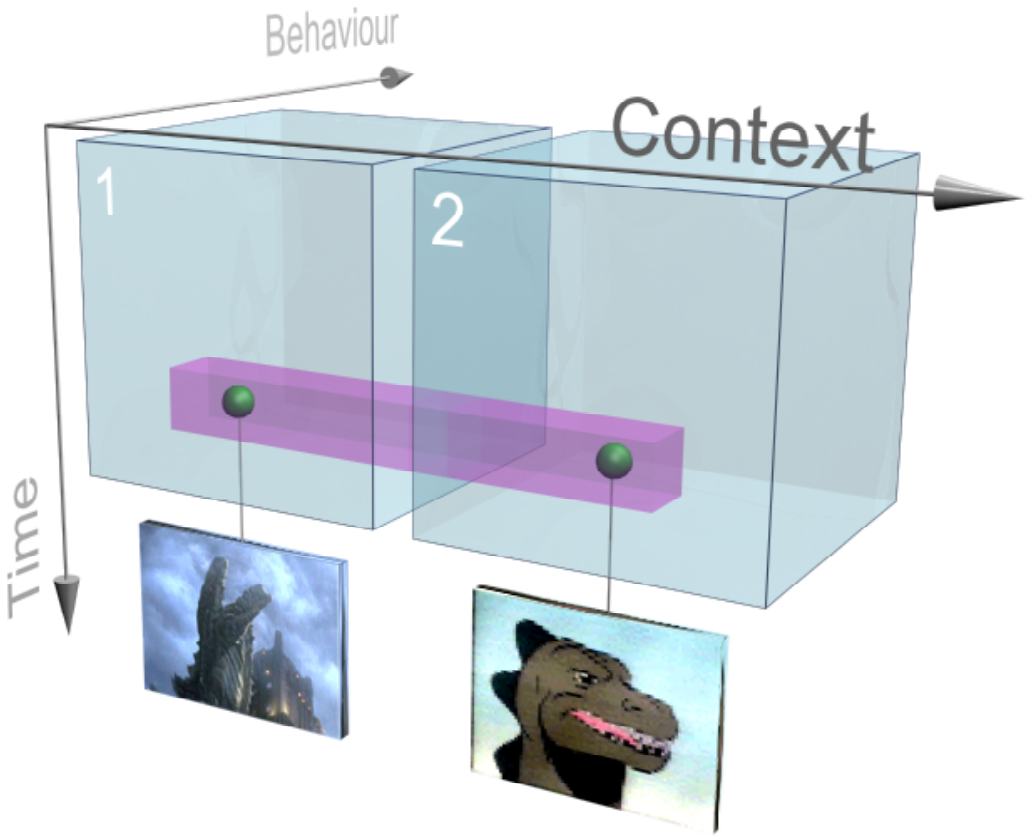

Figure 6.8: A Contextual Link (or Concept)

of our example this would be a link between each cartoon monster and its more terrifying video clip counterpart. This leads us to a third and final question.

Composites: Are Links and Contexts the Same?

Let us consider our previous notion of a link as a composite of zero or more items. It is clear that there are two types of link. Normal links across space (spatial links) and links that cross one or more dimensions of the continuum (contextual links) regardless whether these dimensions are time, behaviour or some other distinguishing factor.

Given the definition of a concept given above it seems clear that contextual links and concepts are the same thing. In other words, the contextual links are actually the structures that maintain an object's cohesion across the dimensions. The link represents a common concept.

A natural consequence of this is that there are two ways to link across space and context. Figure 6.8 shows two nodes, a cartoon of Godzilla and a movie clip of Godzilla, linked across context by the common concept of 'Godzilla'. Notice that links are now represented like composites in the system, so the concept is shown as a thin purple box.

Inside both spaces there are two visible objects; the clip (which is a movie in one space and a cartoon in the other) and the concept. As a result there are two ways to link across context:

1. Link directly to a particular instance of an object, for example by anchoring a link on the cartoon clip. By following this link a user would see the cartoon even if they were an adult.
2. Link to a concept, for example linking to the concept of ‘Godzilla’ rather than a particular clip (anchoring on the purple box). In effect within a single space we link to the concept, which is then resolved across context to be a either the cartoon or the movie clip depending on whether the context of the user is that of a child or an adult.

This gives us a concrete notion of what links and contexts are within the system. Now we can say that the spatial links are analogous to composites and belong in a single space, while the contextual links (which form part of the mechanism of context) are shared amongst all participating spaces.

6.3.3 Time as a Contextual Dimension

One of the dimensions identified in the 6D Model and later incorporated into our notion of context was that of time. As mentioned this dimension represents the development of the hyperstructure over time, although we do not assume any particular granularity.

As time is a contextual dimension, our notions of concept and contextual linking (described above) still apply. Effectively a contextual link across time represents a version history for a particular object. Otherwise the time dimension appears to have little bearing on the way in which we interact with the hyperstructures. However consider the case where we have nodes that represent temporal media.

Normally we can consider the temporal nature of this media as orthogonal and separate to the notion of time across the hyperstructure. However, this view is brought into question when the media in question is live (for example a streamed digital television signal).

Imagine the case where we are querying a node to find out about any anchors attached to it. In the orthogonal time case it seems natural that we receive all the anchors and display them as required, this is because the stream in question is finite. However, in the non-orthogonal case the stream may be infinite in length (consider the live television example) and it seems more natural to rely on context (in particular our position in the time dimension) to return to us only those anchors which are available at the current time.

This is interesting as we are regarding such anchors as objects that are anchored on the entire document but have been designed to exist only for a fixed period of time, after which they will no longer be available. Presumably the locspecs of such anchors would still fix the anchor at a particular time stamp within the stream so that they could be displayed as accurately as the non-orthogonal case.

This is a powerful argument for contexts as a necessity in a hypermedia system, rather than an additional feature, and it indicates that current thinking about structure and temporal media might be incomplete without the notion of context.

6.4 Summary

In this chapter we have explored the information spaces that hypermedia components discuss in terms of hypermedia domains and contextual divisions. In particular we have learnt the following things:

1. There are many hypertext domains, each of which presents information to the user in a particular way and has its own methods of navigation. Spatial Hypertext allows users to zoom in and out of a hierarchy of composites, Navigational Hypertext allows users to follow ad hoc associations between (sections) of documents and Taxonomic Hypertext allows users to navigate up and down a taxonomic hierarchy and also move between overlapping categories.
2. These domains represent part of a Behavioural axis of an *n* dimensional model of information space. Where a users view of the space is altered by their context.
3. Thus rather than thinking of these domains as independent of one another we now consider them part of a larger continuous information space with a common structure. I call this super space of spaces the Information Continuum.
4. Within this continuum it is possible to link between items in a single space (spatial links) or across the dimensions of the continuum, between items in different spaces (contextual links).
5. Spacial links (or intra-space links) are analogous to composites (and may therefore form a Spatial Hypertext hierarchy of composites, or Taxonomic Hypertext hierarchy of categories).
6. Contextual links (or inter-space links) are analogous to concepts, identifying several objects in different contexts as aspects of the same thing. As a user we could create a navigational link to an item and then navigate directly to it, or we could anchor that link on the concept object which is then resolved across context when we follow it.

Perhaps the most important result that arises from our consideration of the Information Continuum is the idea that all the domains of hypertext are related in some fundamental way, such that navigation across those behavioural dimensions is possible. In Section 5.2 I stated that separation of a communication infrastructure from the rest of OHP would allow us to concentrate on the semantic content of conversations. This semantic language is essentially an agreement that components will discuss the world in a particular way, using particular terms.

For example, when a member of the public takes their car to a mechanic, there is a certain vocabulary of terms used to discuss the situation (concepts such as ‘car’, ‘engine’ etc.). When information components discuss hypertext structure they need a similar common lexicon of terms and associated meanings. If hypertext domains are all reflections of a common world then the best semantic language would describe these fundamental structures and enable components to interoperate across domains and users to navigate the information spaces in new and innovative ways.

Chapter 7

The Development of FOHM

7.1 Introduction

In the previous two chapters I divided the problems faced by the OHP effort into two distinct categories. Firstly I tackled the communication problems by describing how a proposition and performative based approach results in a more powerful communication language. Secondly I discussed the integration of the various hypermedia domains into a single information continuum and described how our notions of navigation and context fit into that view.

In the following two chapters I will bring these two threads back together to build a prototype implementation of a cross-domain system.

To do this I firstly describe the two approaches to building such a system, and explain why I believe that the common language approach chosen offers benefits in flexibility and simplicity.

I then introduce FOHM, the Fundamental Open Hypermedia Model, designed to express all the structures found in the three domains of Navigational, Spatial and Taxonomic Hypermedia. These domains are compared and contrasted and their mappings to the FOHM model explored, including a description of what unique structures each domain contributes and how the other domains cope and benefit from them.

7.2 Inter-Domain Interoperability

There are two distinct sorts of inter-domain interoperability that one might imagine. Let us imagine that there are two workspaces, the first of which is a traditional navigational hypertext workspace, and the second of which is a spatial hypertext. If there were a link

from a node in the first workspace to a node in the second workspace, what would be the semantics of following this link? The first and perhaps most obvious outcome would be that as the link was followed into the spatial workspace, a spatial browser would open from where the user could continue to browse. However an interesting second possibility could be that the navigational browser could remain open, and could continue to attempt to interpret the spatial workspace *as if it were* a navigational workspace (Nürnberg *et al.*, 1999).

There are two options:

1. *Translation*. For each domain it should be possible to provide a translation mechanism for each other domain. This translation functionality could be captured in a middleware layer, making it reusable across many other components.
2. *Common Language*. Alternatively we could attempt to define a *common* model which would be sufficiently generic that it could express all the structures required by all current domains, and hopefully all future domains. This model would in effect be a common semantic language.

These approaches differ over two features:

1. *Extensibility*. This concerns the ability to include new domains into the system in the future. The translation approach allows new translations to be added but exhibits quadratic growth with the number of domains that intend to interoperate, thus in practice the number of interoperating domains may be quite limited. The common language approach relies on the ability to express any domain in the common language, something that cannot be tested in advance but which does guarantee that all domains that can be expressed may interoperate.
2. *Distinction between Domains*. The translation approach makes a clear distinction between domains (and thus provides unambiguous semantics). However it is not known whether a clear distinction is appropriate, in other words, why can't a hyperstructure belong to multiple domains? The common language approach makes no attempt to define boundaries between different domains, however as a result it looses the advantage of distinct domains as the semantics concerning the structures become blurred.

To summarise, although there are concerns over the extensibility of a common language, it offers benefits in flexibility, both to the information structure and the software implementations and should be investigated.

The creation of a common language depends on examining existing domains and drawing parallels between them. If objects that exist in one domain can have meaningful representations in another, then a common language is possible.

7.3 FOHM

In order to explore the similarities between different domains I began to construct a formal model of the three existing domains with the help of Dr. Luc Moreau and Dr. Sigi Reich (Millard *et al.*, 2000). In this section I shall briefly present the formal definition of the model which we shall then use to explore the common ground between the different domains and confront their differences.

7.3.1 Comparison of Domains

There are a number of similarities between the different hypertext domains that we can use to build a common semantic language. In all domains, we find notions of *data* (nodes in navigational, visual icons in spatial, or artifacts in taxonomic hypertext) and *associations* (links in navigational, composites in spatial or categories in taxonomic hypertext). On the other hand, there are features that only appear in a single specific hypertext domain:

- The navigational domain contains anchors that allow links to point into documents rather then be anchored on entire documents.
- The spatial domain introduces typed structures (e.g. lists, sets, etc.) to the relationships.
- The taxonomic domain contains perspective objects that allow the relationship structures to diverge according to different views.

Other differences concern restricting the possible composite structures for various domains. For example circularity is allowed in navigational hypertext but not in taxonomic or spatial.

7.3.2 A Fundamental Hypertext Layer

Given the similarities between the different domains we defined a *fundamental layer* composed of data structures that are general enough to ‘encode’ any data structure of each hypertext domain. These operations and data form a Fundamental Open Hypermedia Model, which we call FOHM. We also define general operations on these data

$$
\begin{array}{rcll}
I^a & = & \{\alpha_0^a, \alpha_1^a, \ldots\} & \text{(Association IDs)} \\
I^d & = & \{\alpha_0^d, \alpha_1^d, \ldots\} & \text{(Data IDs)} \\
I^r & = & \{\alpha_0^r, \alpha_1^r, \ldots\} & \text{(Refs IDs)} \\
I & = & I^a + I^d + I^r & \text{(IDs)} \\
\mathcal{D} & = & \{d_0, d_1, \ldots\} & \text{(Data)} \\
\mathcal{B} & = & I^r \times \vec{\mathcal{F}} & \text{(Bindings)} \\
\vec{\mathcal{B}} & = & \mathbb{N} \rightarrow \mathcal{B} & \text{(Binding Vectors)} \\
\mathcal{A} & = & \vec{\mathcal{B}} \times \mathcal{T} \times \mathcal{S} & \text{(Associations)} \\
\mathcal{T} & = & Name \times \vec{\mathcal{N}} & \text{(Relation Types)} \\
\mathcal{S} & = & \{\mathsf{heap}, \mathsf{list}, \mathsf{stack}, \mathsf{perspective}, \ldots\} & \text{(Structural Types)} \\
\mathcal{R} & = & \mathsf{whole} : I \rightarrow \mathcal{R} \mid \mathsf{inside} : I \times \mathcal{L} \rightarrow \mathcal{R} & \text{(Data Refs)} \\
\mathcal{L} & = & \{l_0, l_1, \ldots\} & \text{(Loc Specs)} \\
\mathcal{F} & = & Dir + Shape + \ldots & \text{(Features)} \\
\vec{\mathcal{F}} & = & \mathbb{N} \rightarrow \mathcal{F} & \text{(Feature Vectors)} \\
Dir & = & \{\mathsf{src}, \mathsf{dst}, \mathsf{bi}\} & \text{(Direction Features)} \\
Shape & = & \{\mathsf{square}, \mathsf{circle}, \ldots\} & \text{(Shape Features)} \\
\mathcal{N} & = & \{\mathsf{direction}, \mathsf{shape}, \ldots\} & \text{(Features Spaces)} \\
\vec{\mathcal{N}} & = & \mathbb{N} \rightarrow \mathcal{N} & \text{(Feature Space Vectors)} \\
Name & = & String & \\
Object & = & \mathcal{D} + \mathcal{A} + \mathcal{R} & \text{(Objects)} \\
Store & = & I \rightarrow Object & \text{(Store)}
\end{array}
$$

Figure 7.1: The Fundamental Data Model

structures. Figure 7.1 contains a formal representation of the data model, whereas Figure 7.2 displays some example operations, defined as transitions of an abstract machine representing a server and client.

FOHM makes no assumptions about the protocol it is running over or the systems that are using it. It is a semantic language that requires an implementation in a syntactic language before it can be used. Although other structural languages exist they tend to contain very little semantic information and suffer performance penalties as a result of their generality. Section 4.5 describes our experiences of this problem with XML.

7.3.3 A FOHM Example

In FOHM I describe four objects that are analogous to objects in the OHP-Nav data model. I attempted to give these objects unique names to avoid clashes with models of individual domains. An *association* object represents a relationship between other objects, it contains a feature space; a list of features that all the objects in the association must map to. It also contains a set of *bindings*, these attach *references* (usually datarefs) to the association via a feature vector that describes how the reference maps to the feature space. Finally FOHM has a notion of a *data* object, this is a wrapper for some piece of

Functions:

$$
\begin{array}{rcl}
createAssociation & : & \mathcal{A} \times Store \rightarrow (I^a \times Store)_\perp \\
& = & \lambda\, a\, \sigma.\ \textbf{let}\ i\ =\ newId(\sigma) \\
& & \qquad \textbf{in}\ (i, \sigma[i \rightarrow a]) \\
& & \qquad \textbf{end} \\
deleteAssociation & : & I^a \times Store \rightarrow Store_\perp \\
& = & \lambda\, i\, \sigma.\sigma[i \rightarrow \perp] \\
\\
retrieve & : & I \times Store \rightarrow Object_\perp \\
& = & \lambda\, i\, \sigma.\sigma(i) \\
\\
addBindingToAssociation & : & I^a \times I^r \times \vec{\mathcal{F}} \times Store \rightarrow Store_\perp \\
& = & \lambda\, i^a\, i^r\ \vec{f}\ \sigma.\ \textbf{let}\ \ (\vec{b}, t, s)\ =\ retrieve(i^a, \sigma) \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \vec{b'}\ =\ \vec{b}[size(DOM(\vec{b})) \rightarrow (i^r, \vec{f})] \\
& & \qquad\qquad\qquad \textbf{in}\ \sigma[i^a \rightarrow (\vec{b'}, t, s)] \\
& & \qquad\qquad\qquad \textbf{end}
\end{array}
$$

Queries:

$$
\begin{array}{rcl}
getBindingForData & : & I \times \vec{\mathcal{F}} \times Store \rightarrow (SetOf(\mathcal{B} \times I^a))_\perp \\
(i, \vec{f}, \sigma) & \rightarrow & \{(b_1, i_1^a)\ :\ (\mathcal{B} \times I^a)\ |\ \ b_1 = (i_1^r, \vec{f}_1) \\
& & \qquad\qquad\qquad\qquad\quad b_1 \in retrieve(i_1^a, \sigma).\vec{B} \\
& & \qquad\qquad\qquad\qquad\quad \vec{f}_1 = \vec{f} \\
& & \qquad\qquad\qquad\qquad\quad retrieve(i_1^r, \sigma).I = i\ \ \}
\end{array}
$$

Figure 7.2: Some Fundamental Functions and Queries

data that lies outside the scope of the model, normally a document although it could represent any file or stream.

Figure 7.3 shows a possible FOHM structure. Bindings map References (DataRefs in this case) to the Navigational Link on the left by defining their direction and to the Spatial List on the right by defining their position. The DataRefs either reference a whole item of Data or point into that Data (e.g. to reference a particular region).

7.3.4 Levels of Structure in FOHM

Between them the domains modelled support three levels of structure:

1. *Explicit External Structure.* By creating typed associations between data, FOHM allows explicit relationships to be expressed. This is analogous to taxonomic categorisation, where we say that an object (or data) belongs in one category or another.
2. *Implicit External Structure.* This is classification within a relationship, defined by the feature space of an association and the corresponding feature vectors. E.g. in

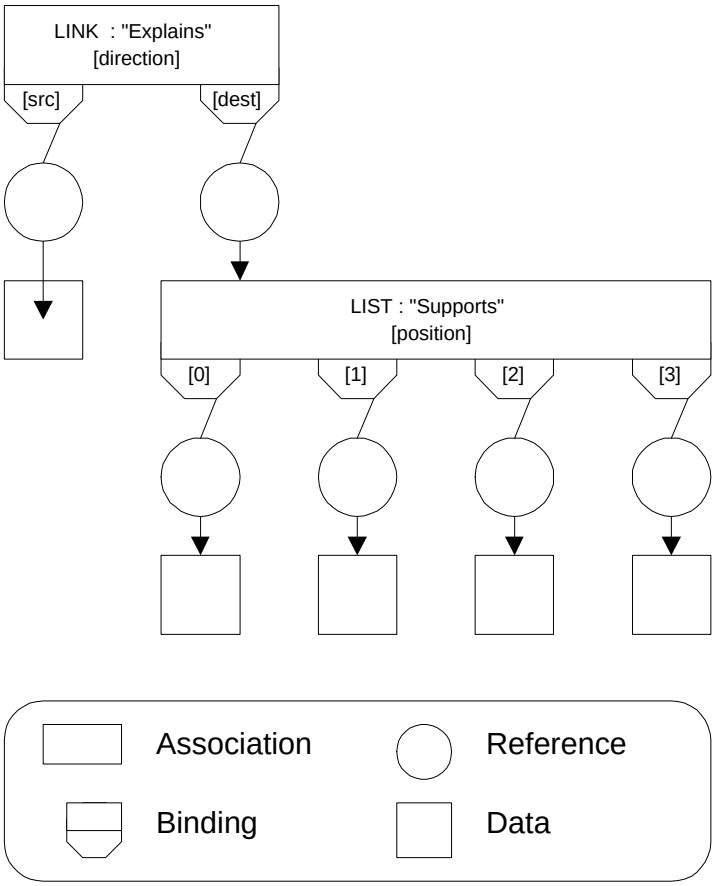

Figure 7.3: FOHM Structures: A Navigational Link to a Spatial List

Spatial Hypertext an association may contain a set of data objects, each one of which is given a Red/Green/Blue (RGB) value. Objects which are similar in some way will be more alike in colour.

3. *Internal Structure* is the structure of the information actually inside individual data objects. For example a film can be viewed as a collection of many scenes, a user viewing the film follows a path or trail through those scenes. Typically this is not the way a film is stored due to file size and performance restrictions. Thus at some level we no longer handle structure externally and instead handle it internally via a proprietary data format. References allow FOHM's external structures to reference the internal structure of data (e.g. link to the seventh scene of a film).

One of the most important aspects of FOHM is that if a client does not totally understand the semantics of the structure it is given it may still understand a portion of it. For example imagine an association that represents a company. This association may have a feature space with a single feature 'role'. All data that binds to this association must have a vector that maps to that feature, in effect stating what the role of that data object is in the company.

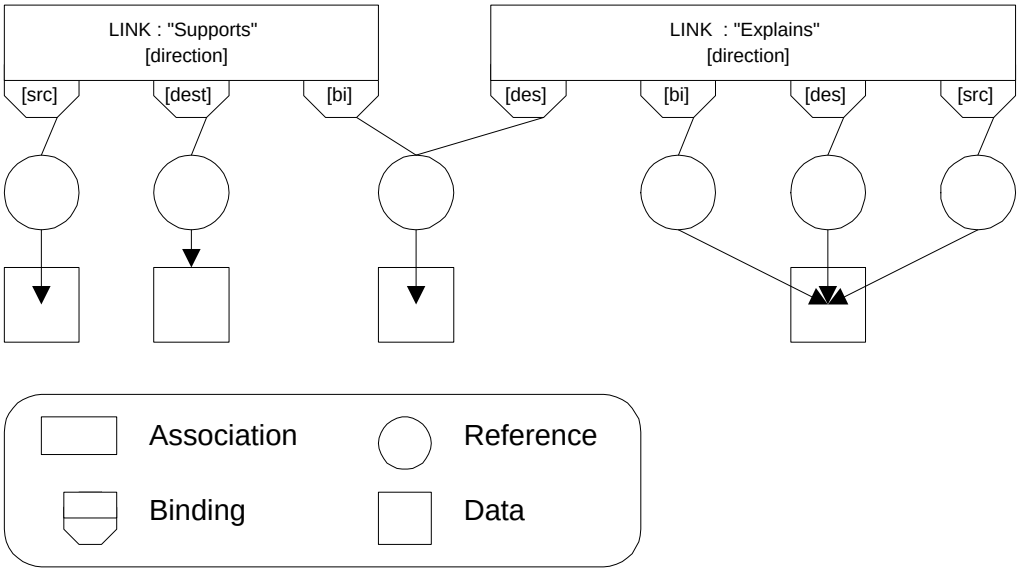

Figure 7.4: FOHM Structures: Two Navigational Links

Should this structure be served to a client that understands the feature space ‘role’ then that client will appreciate all of the meaning of that structure. However, if it was served to a client that didn’t understand the feature then that client would at least still understand that a relationship exists, could display the relationship to the user, and otherwise manipulate the association as normal.

7.4 The Contribution of the Domains

Each domain brings something unique to the FOHM model (Millard & Davis, n.d.). In this section we will look at what is brought to FOHM from each domain and discuss how each of the domains are extended to handle the extra structure.

7.4.1 Navigational Hypertext: Introducing the Anchor

In many ways Navigational Hypertext is the simplest of the three domains. Its notion of directed links can be easily modeled in FOHM by a single feature ‘direction’ to which nodes are bound with either a ‘source’, ‘destination’ or ‘bi-directional’ value. The anchor object allows external linking structure to point into otherwise opaque data, it is this mechanism that allows the referencing of *internal structure* as mentioned above. However neither Spatial or Taxonomic Hypertext has such a mechanism.

In FOHM the anchor is replaced by a reference object (a DataRef). This object is also accessible by Spatial and Taxonomic clients enabling those domains to reference internal structure as well. This is useful to both domains and allows spaces and categories to refer to parts of a data object as well the object in its entirety.

Figure 7.4 shows a Navigational structure described within the FOHM model. In this case two links. The first is a link across three different data objects (one of which is referenced in its entirety), the second is a link across one area of one document and three different areas within a second document. Notice that Associations can share References and that References can share Data objects.

7.4.2 Spatial Hypertext: Classification within Relationships

At a superficial level it is possible to view Spatial Hypertext as a presentation layer on top of Navigational Links. For example a link may have some attributes that determine that when viewed spatially it appears as a red square. However this is to miss the important notion of *implicit external structure* as described above. In a Spatial Hypertext System the visual attributes of the various objects actually form extra structural information about those objects, one of the applications of which is to allow fuzzy membership of an association.

Spatial Hypertext Systems rely on users to understand the visual clues supplied. E.g. they can express that one object is redder than another but no semantic reason is given, the user must interpret that information themselves. In FOHM we use the feature space to contain all the spatial features that datarefs may bind to, e.g. colour, shape, size, etc. A binding maps values to those features, therefore describing a references position in the space. An interesting consequence of this approach is that as FOHM has no restriction on the feature spaces used it is possible to replace spatial mappings with semantic ones. In effect allowing the system, as well as the user, to appreciate fuzzy membership.

As an example consider a set of nodes that represents dangerous animals. In a Spatial Hypertext system we may colour the animals such that red indicates danger. When rendered the user can see what the nodes represent and they can also see that some are redder than others, but the understanding of what red represents is lost. However in FOHM we could replace the colour with a different feature that explicitly enshrined that meaning, e.g. ‘danger’, which has defined values ranging from ‘harmless’ to ‘deadly’. A system that understands the meaning of the ‘danger’ feature has a true understanding of how the animals relate to one another. The disadvantage of this is that it is probable that more systems understand colour than a specific feature like ‘danger’.

Figure 7.5 shows a simple Spatial structure described within the FOHM model. In this case there is a set of three objects that ‘Support’ each other. This set contains two data objects and a list. This list actually represents a ‘Tour’ that supports the documents in the set. Notice that the third position in the tour revisits the first data object.

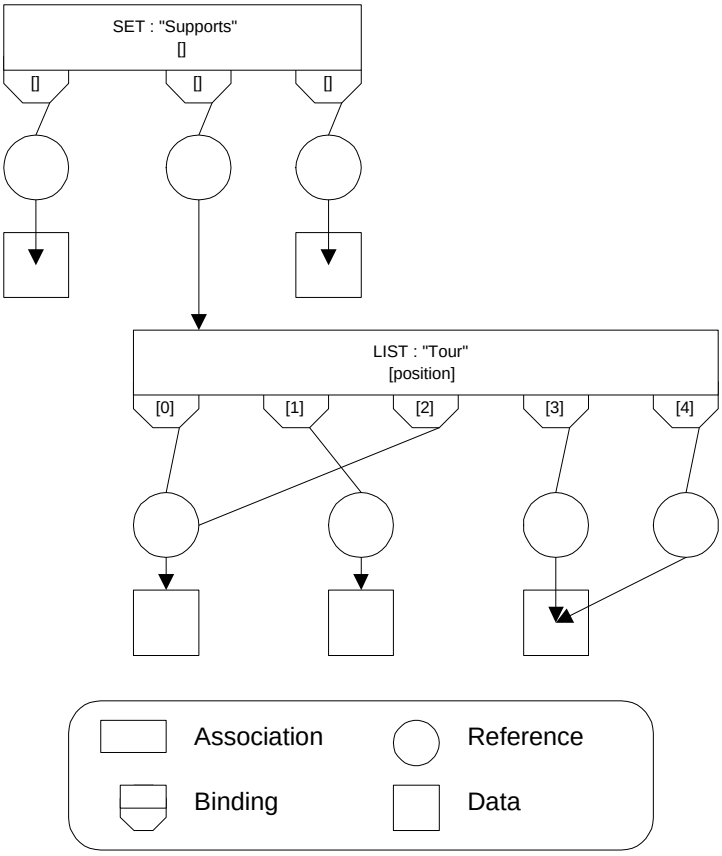

Figure 7.5: FOHM Structures: A Spatial Structure

In our example we have used the feature space to define the internal structure of a space. For example the list has a ‘position’ feature and all bindings must contain a mapping to that feature (i.e. their position in the list). However a feature space may be empty, as can be seen from the Set. In this case the structure type alone is enough to govern the structure (i.e. no reference may bind to a set more than once) and the feature vector is empty.

The feature space mechanism is a very powerful way of binding objects to associations with different internal structure, however we must extend Navigational and Taxonomic Hypertext to deal with these powerful bindings.

With Taxonomic Hypertext this is a fairly trivial operation, the *implicit external structure* simply becomes an extension of the categorisation process, where the fuzzy membership of a category becomes possible. In addition, if the people creating a taxonomy disagree on this implicit structure then they can divide the taxonomy at that point using the perspective feature as normal.

Navigational Hypertext has many more issues to deal with because it relies on the feature space to enable the navigation of associations. There are two ways to cope with the ‘pollution’ of the feature space.

Structure	Traversal: endpoints user reaches	Arrival: object user sees
Link	All destination or bi-directional endpoints except starting endpoint	All members
Set	All except starting endpoint	All members
Stack	Endpoints to either side	Top of stack
Queue	Next endpoint in queue	Start of queue
List	Endpoints to either side	All members
Matrix	Nearest endpoints	All members

Table 7.1: Semantics of Traversal and Arrival

1. Ignore it. A Navigational client can assume that if there is no direction feature then all members of an association can be treated as bi-directional for purposes of navigation.
2. Extend the navigation model. We can make the same assumption as above but also extend our model of navigation so that it understands some of the common features that would be used by the other domains and allows them to alter the effect of navigation.

In actual fact this is a decision that is beyond the scope of the model and should be made by individual clients, however it is important to think about the consequences of a client taking the second approach. If we had a standard definition of what it means for a client to navigate a list (as opposed to a link) then hypertext designers could build their hypertexts to take advantage of that functionality.

There are two places were the structure of an association could become important. The first is *traversal*, the act of following a link (Trigg, 1998). In normal Navigational Hypertext a link has a basic internal structure based on the notion of ‘direction’, if it could have different internal structure then the behaviour of the traversal could vary according to that structure. The second is *arrival*, the act of viewing a structure (for example at the end of a traversal). Here internal structure could effect the way in which members of the association are viewed. Table 7.1 shows some possible semantics for traversal and arrival over different structures.

These structural types would add valuable organization to otherwise disorganized hyperwebs. For example take a look at Figures 7.6 and 7.7.

In Figure 7.6 a collection of slides has been gathered together into a set. This means that the association is unordered but that no individual slide may appear more than once. Our link semantics dictate that when users arrive at a set they see all of the endpoints contained, thus if users arrive at the set of slides they would see all four slides. When

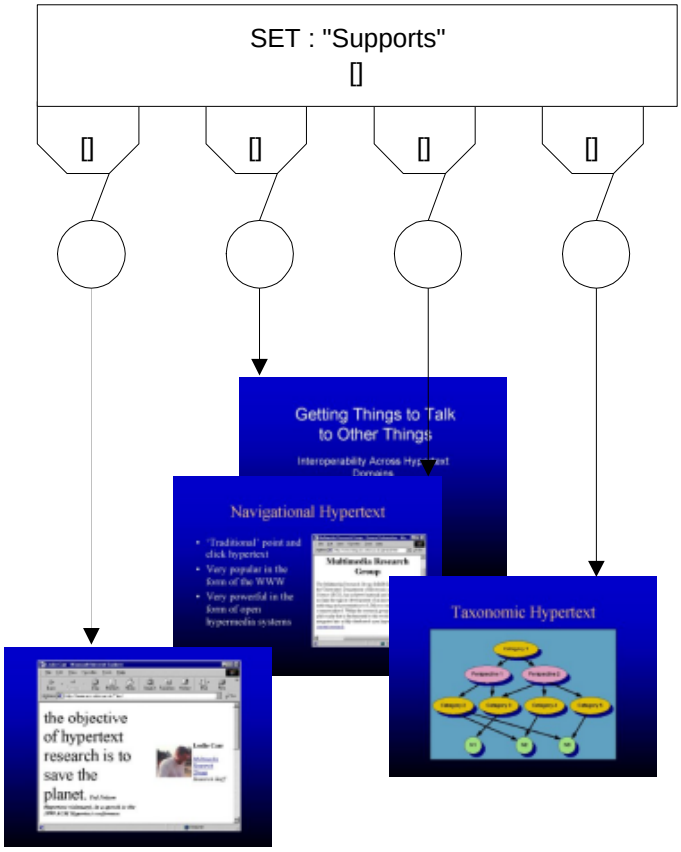

Figure 7.6: Link Structures : A Set of Slides

traversing a link the user always uses an endpoint that is a member of that link as a starting point. When traversing a Set the user arrives at all the endpoints except the starting one. So when traversing the set of slides from the slide on the left they would arrive at the three slides on the right.

In contrast Figure 7.7 shows the same four slides arranged in a stack. When a user arrives at a Stack they see only the top endpoint of the stack, in this case they would see the first slide. In addition when a user traverses a stack they arrive at the endpoints to either side of their starting point. So if the user started their traversal at the third slide they would arrive at the second and fourth.

7.4.3 Taxonomic Hypertext: Perspectives and Context

Although the OHP-Nav data model has no notion of context, context has long been an important issue in navigational systems, allowing a user to see different versions of documents or hyperwebs according to a particular viewpoint. In Taxonomic Hypertext context is realised via the use of a perspective object. These are designed to be placed in a categorisation hierarchy at the point where it splits according to the views of the authors.

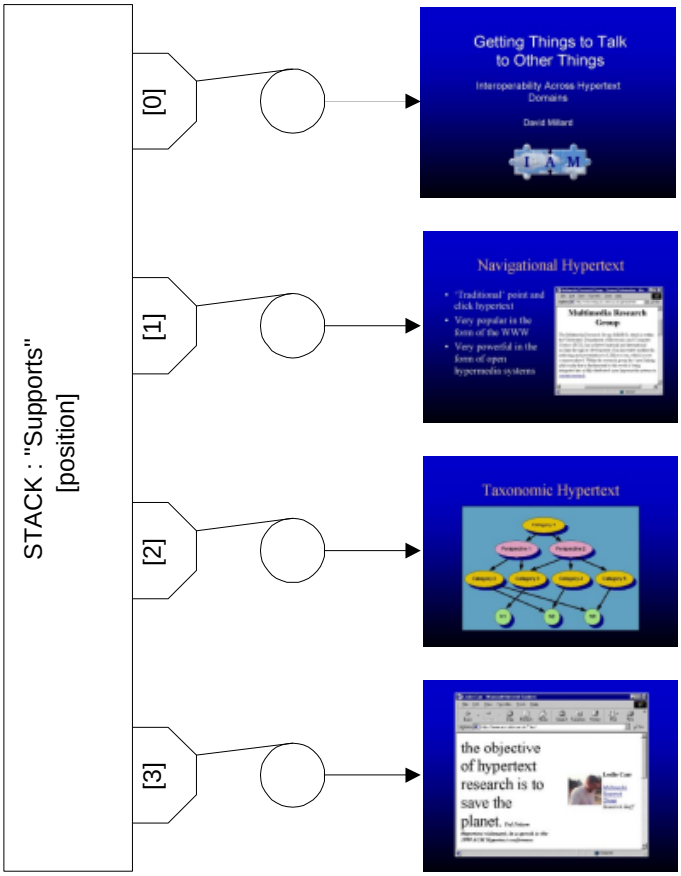

Figure 7.7: Link Structures : A Stack of Slides

The context of the viewer will determine which perspective (branch of the taxonomy) that they see.

When implementing a perspective in FOHM one could use an association of type ‘perspective’. However this does not fit in very well with the rest of the model. This is because the semantics of what a perspective does to traversal and arrival functionality has to be understood by the association containing those perspectives and not by the perspectives themselves. In addition what would it mean for that parent association to have structure? What is the meaning of a list of perspectives as opposed to a set?

Fortunately there is a way of shifting this knowledge back into the perspective object itself. Rather than a category containing perspectives representing a branch, in FOHM we say that a perspective object *is* the branch. If there were two versions of a data object you would create a perspective association that contained both versions. Now the arrival semantic of a perspective is to choose one of the data objects to reveal and the traversal semantic is essentially the question ‘what other views are there on this object?’

Figure 7.8 shows these alternative implementations of perspective. Note that in the FOHM model there are two versions of Category 1 - the result of viewing that Category

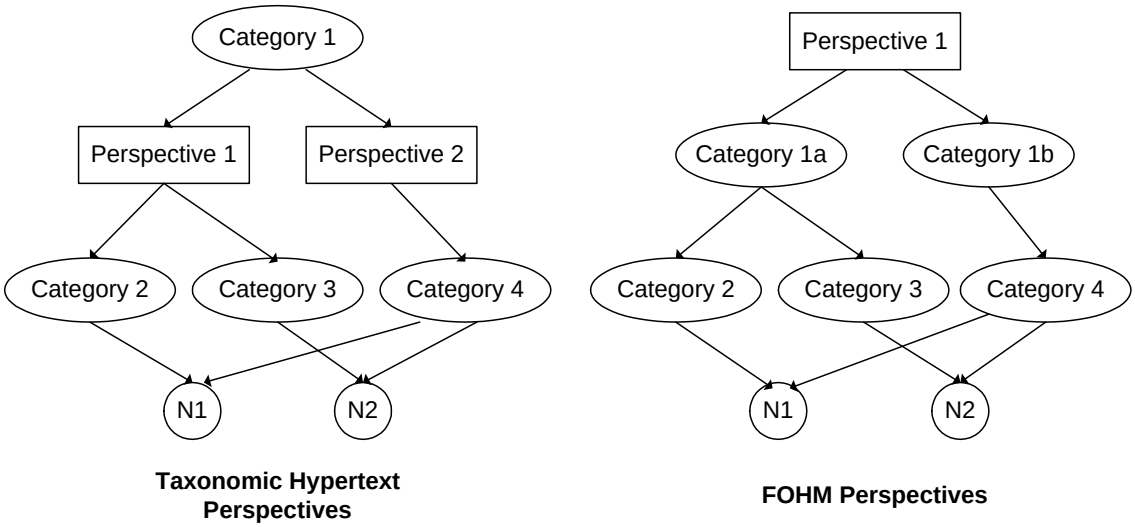

Figure 7.8: Alternative Implementations of Perspective

in different contexts.

Interestingly this is exactly the same structure as the ‘concept’ discussed in Section 6.3.2. A structure that binds alternative views together across the contextual axes of the information continuum. Thus in FOHM a perspective is viewed as just another typed association. Because of this the FOHM model of perspective contains some properties that are very desirable when dealing with contextual systems:

1. The ability to link to a specific node, whatever the contextual considerations (linking across context). This is achieved by linking to the node itself.
2. The ability to link to a node that is determined by context. This is achieved by linking to the perspective that contains the choice of nodes.
3. Everything about the structure can change in context. Including the type of that object (e.g. linking to a perspective could result in a data object in one context but result in an association object in another)

However FOHM only gives a framework in which context can operate, it does not define context itself. For example it does not have an explicit mechanism for determining on arrival to an association which members are visible. This is because the definition of these mechanisms is an open ended and tricky question. One that is beyond the scope of this thesis. Instead FOHM places the context mechanism into opaque or black-box objects and then concentrates on placing them correctly.

FOHM assumes the existence of two types of black-box object. Both are considered to be profiles, where a profile is an object that contains contextual information.

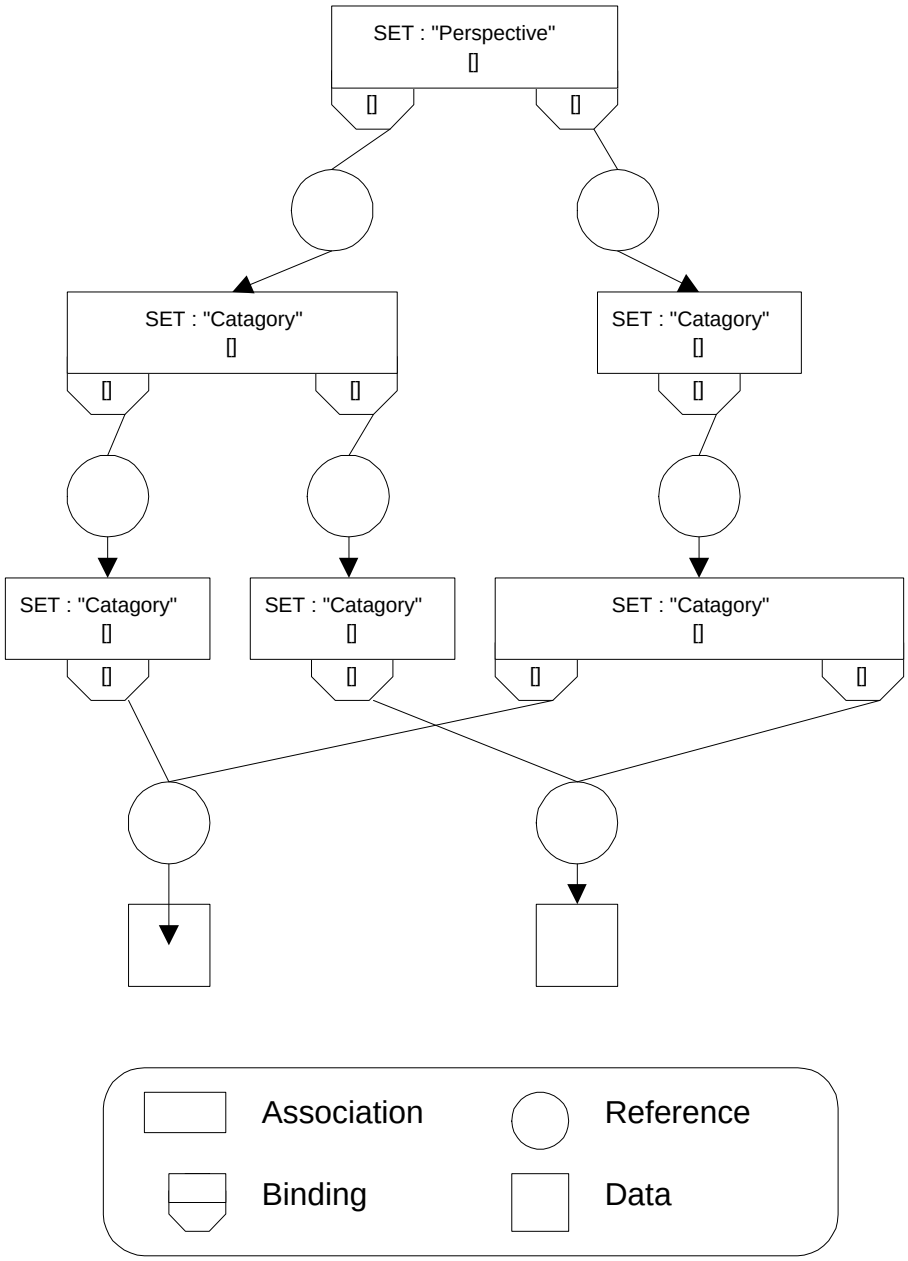

Figure 7.9: FOHM Structures: A Taxonomic Hierarchy

1. *User Profile.* This is an object that defines the state of the user. This could include information about the user (such as age and job position) as well as information relating to their interaction with the system (trails etc.). In addition it may include temporary user specified information that will help the user filter the information space (e.g. today I am interested in cars).
2. *Data Profile.* This is an object that describes another object in a contextual fashion. This could include information about the object (such as the date of its creation), versioning information as well as a description of to whom the object would be interesting.

In addition FOHM assumes the existence of a ‘magic function’ that compares a user profile to a data profile and decides if they match. A simple implementation could define the profiles as lists of keyword value pairs and then the function would choose an object based on the number of matching values.

Figure 7.9 shows a Taxonomic hierarchy described within the FOHM model. In this hierarchy the Perspective and Categories has been modelled as a set with the relationship type ‘perspective’ and ‘category’ respectfully. This example demonstrates two alternative views on how to categorise two data objects. The left fork of the concept views each object as belonging to a separate category while the right fork places them together in a single category.

FOHM assumes that each *binding* object contains a data profile and that when querying the system the user provides a user profile which the system can use to ‘filter’ results. Effectively choosing a subset of each structure to ‘reveal’. The ‘perspective’ is purely a particular use of this general mechanism. If a user arrives at the ‘perspective’ then their user profile is compared to each data profile (there is one for each branch) and one of the branches is revealed as a result.

Another place where the profile comparison might be used is in the process of querying for reference objects that point into a particular document. In this case for each reference that refers to that document the users profile is compared to all the reference’s bindings’ profiles and only the ones that match are shown. This kind of functionality is important in many adaptive hypermedia systems where the author of a hypertext wishes to guide a reader based on their task, goals or experience (Brusilovsky, 1996).

7.5 Summary

In Chapter 6 I described how a common vocabulary is required in order to discuss hypermedia structures over a common infrastructure. In this chapter I have explored the difficulties of unifying three hypertext domains (Navigational, Spatial and Taxonomic) with the aim of producing such a vocabulary and presented FOHM, a semantic language in which all three domains have a logical representation.

In particular I have described:

1. How Navigational Anchors can provide greater granularity of data in Taxonomic and Spatial Systems.
2. How the internal structure of different spaces (e.g. Sets, Lists, etc.) might provide a richer set of traversal and arrival semantics for the navigation of links

3. How Taxonomic Perspectives suggest a model of context that is constant with our notion of contextual dimensions (see Section 6.3.1) and which allows us to link to concepts as well as across context to a particular instance of an object.

FOHM is a powerful way to express hyperstructures, but on its own it does not provide interoperability. To do that an implementation of FOHM must be created based on some existing infrastructure, over which FOHM structures can be discussed.

Chapter 8

The Implementation of FOHM

8.1 Introduction

In this thesis I have described how early work with the Open Hypermedia Systems Working Group on the Open Hypermedia Protocol has grown into a general investigation of what constitutes hypermedia and what it means to navigate around and between different hypertext spaces and domains. This has resulted in the Fundamental Open Hypermedia Model which can represent consistently three of the major hypertext domains.

I have also examined the syntactic aspects of communication and explored human linguistics and its software equivalents in the agents world.

In this chapter I shall present an implementation of FOHM, based around the SoFAR agent framework, to explore new notions of cross-domain browsing. Two clients, one spatial and one navigational, were developed for the implementation and these clients are described and the issues faced in their design discussed.

Finally I will reflect on the new research field of Structural Computing described in Section 4.2.1. The CB-OHSs developed in this community share some of FOHM's multiple domain functionality and it is interesting to compare the two approaches and explore whether FOHM supports the Structural Computing philosophy.

8.2 Agent Implementation

To test FOHM I wanted to build on the Solent system described in Section 4.4 however the emphasis of the work was shifting away from infrastructures and I thought it best to choose an existing infrastructure, rather than modify the Solent system in some way.

query_if	Is this predicate true?
query_ref	What predicates match this predicate?
inform	This predicate is true
uninform	This predicate is false
request	The sender asks the recipient to make this action true

Table 8.1: SoFAR performatives for Communication

The Southampton Framework for Agent Research (SoFAR) (Moreau *et al.*, 2000) is a communication infrastructure based around dynamically discovered components or agents. All the work described in Chapter 4 is leading toward the adoption of a logically separate communication infrastructure and the consideration of performatives discussed in Section 5.3.3 indicates that a proposition based language coupled with 'speech acts' provides one of the most powerful.

SoFAR is designed around just such a mechanism. Because of this I chose to implement FOHM using SoFAR. This required the creation of three 'agents'. Two would be clients that understood Navigational and Spatial structures respectively, and displayed them appropriately to the user, the third would be a storage agent, responsible for storing the structures and answering questions posed by the first two agents. All three would speak FOHM exclusively using the ontology based communication of the SoFAR framework.

8.2.1 Description of SoFAR

SoFAR is a multi-agent framework designed for Distributed Information Management (DIM) tasks. To understand the philosophy of the framework it is necessary to examine both its architecture and process of communication.

SoFAR communication

Communication in SoFAR is based on collections of related propositions known as ontologies. Each proposition in the ontology asserts a particular fact, and therefore is known as a predicate. For example the predicate:

```
ReverseString("hello", "olleh")
```

asserts the fact that the reverse of "hello" is "olleh". Communication is achieved by passing predicates between agents along with one of a set of performatives that show the intention. There are four performatives available for general communication, these are shown in Table 8.1

register	The sender advertises their capabilities with a broker
unregister	The sender withdrawals a previously made advertisement
subscribe	The sender asks to receive any inform statements made about a predicate
unsubscribe	The sender cancels a previously made subscription

Table 8.2: SoFAR performatives for Meta-Communication

For example these predicates could be used with the 'ReverseString' predicate in the following ways:

- *query_if(ReverseString("hello", "olleh"))* Asks: 'is the reverse of "hello" "olleh"?'
- *query_ref(ReverseString("hello", variable))* Asks: 'what is the reverse of "hello"?'
- *inform(ReverseString("hello", "olleh"))* States: 'the reverse of "hello" is "olleh" '
- *uninform(ReverseString("hello", "hello"))* States: 'the reverse of "hello" is not "hello" '

Notice that in the 'query_ref' example the system uses a special *variable* value to show that a field within a predicate is not known. An agent sending such a query would expect some other agent to reply and fill in the variable field, thus answering the query.

In addition to predicates it is possible to define *actions*, these are objects that describe a particular action that can be invoked. There is only one performative that communicates actions, this is the last entry in table 8.1; *request*. This requests that the recipient performs the action.

SoFAR architecture

To allow agents to communicate it must be possible for them to locate one another within the infrastructure. To this end SoFAR contains a registration agent that is always running. When another agent starts it registers the predicates and actions it understands with the registry agent. Other agents can then query the registry agent for a list of agents that understand certain predicates or actions. There are an additional four performatives used for this sort of meta-communication. These are shown in Table 8.2.

For a full description of the capabilities of the SoFAR system see (Moreau *et al.*, 2000).

8.2.2 The FOHM ontology

SoFAR communicates by using sets of predicates grouped together as an *ontology*. An ontology can be thought of as a subset of the set of truth statements that can be made about the world. Therefore to allow agents to discuss FOHM structures required the creation of a FOHM ontology, the set of truth statements that can be made about FOHM.

As FOHM is primarily a data model I decided that it would be best to create a predicate for each first class object, such that the predicate asserts the existence of the object. Thus to 'inform' the 'association' predicate is to assert that that association exists. This way the query_if and query_ref predicates create an instant and powerful query language.

FOHM Predicates

Figure 8.1 shows part of the latest version of the FOHM ontology. The full ontology, which includes comments, can be found in Appendix F

One of the important things about SoFAR is its notion of ground vs. variable terms. A ground term is one in which every field has a defined value. A ground term will only 'match' another ground term with all the same values. A variable is an instance of a term that will match anything. By filling terms or predicates with a collection of variables and ground terms it is possible to create a pattern that will match against other predicates.

For this reason it is important to distinguish between a field that is a variable and one that has no value (but is ground). For this reason the FOHM ontology contains several objects prepended with the word *Undefined*. An Undefined object has no value but will only match other Undefined objects.

Another consideration was to avoid the resolution problem first identified with OHP-Nav (see Section 3.2.2). This is where all the objects in a data model reference each other by an identifier which results in clients generating a great deal of network traffic as they resolve all the references.

One way to avoid this problem is for components to deal in larger hyperstructures. For this reason the FOHM ontology was altered from its original form, where references were used extensively, to that described in Appendix F. Using this ontology a system is capable of choosing whether to include a reference (an IDRef) or an object (ObjectRef) within a Binding.

Not only does this reduce the resolution problem but some Deitic Expressions (as described in Section 5.3.2) can be employed that ensure that an object is only sent once, even though it may appear several times in the hyperstructure. Appearances other than

```
<term name="ReferencableObject" extends="Predicate" abstract="yes"/>

<term name="Association" extends="ReferencableObject">
<field type="StorageID" name="id"/>
<field type="String" name="relationshiptype"/>
<field type="String" name="description"/>
<field type="String" name="structuretype"/>
<field type="BindingVector" name="bindings"/>
<field type="StringVector" name="featurespace"/>
</term>

<term name="Binding" extends="Term">
<field type="StringVector" name="featurevalues"/>
<field type="Reference" name="reference"/>
</term>

<term name="Reference" abstract="yes" extends="Predicate"/>

<term name="ObjectRef" extends="Reference">
<field type="StorageID" name="id"/>
<field type="ReferencableObject" name="target"/>
<field type="LocSpec" name="locspec"/>
</term>

<term name="IDRef" extends="Reference">
<field type="StorageID" name="id"/>
<field type="ReferencableStorageID" name="target"/>
<field type="LocSpec" name="locspec"/>
</term>

<term name="Data" extends="ReferencableObject">
<field type="StorageID" name="id"/>
<field type="MediaObject" name="content"/>
</term>

<term name="UndefinedReferencableObject" extends="ReferencableObject"/>
```

Figure 8.1: Example of the FOHM XML Ontology Definition

the first can be replaced with a reference (a Local or Unique ID). This is analogous to referring to a person by name in a conversation and from that point on referring to that person as ‘he’ or ‘she’.

8.2.3 The FOHM Server

The first agent that needed to be developed was one that acted as a persistent storage device in the system, storing FOHM structures created by other agents and serving them

```
<term name="StoreAssociation" extends="Action">
<field type="Association" name="assoc"/>
</term>

<term name="StoreData" extends="Action">
<field type="Data" name="data"/>
</term>

<term name="StoreReference" extends="Action">
<field type="Reference" name="ref"/>
</term>
```

Figure 8.2: FOHM Storage Actions

up again on demand.

At a superficial level this seems simple and is possible using the predicates described above. A FOHM Server Agent can wait to be informed that certain structures exist and can then remember them. However there may be many inform messages flying around inside an agent cloud which could flood a FOHM Service Agent. For this reason I wanted to make creation more explicit.

In SoFAR this is easily accomplished by defining three storage actions, shown in Figure 8.2.

Now agents can seek other components that support these storage actions and use the ‘request’ performatives to ask that one is performed. Of course there is no rule that prevents future agents merely remembering inform performatives.

A FOHM Server was developed that used these principles to store information in memory, with a simple persistence mechanism that preserved the data between different executions of the Server.

8.2.4 The Spatial Client

The Spatial Client was the first of the two agents that I implemented to talk to the FOHM Server. Previous Spatial Hypertext applications have placed a great deal of emphasis on the spatial parsing of the users view to explicitly create the structures they have implicitly created. I was less interested in the process of spatial parsing than I was in the idea of spatially viewing structures, so for the FOHM implementation I concentrated on the comprehension and display of structures and avoided spatial parsing functionality entirely. The client did have creation functionality but it forced users to explicitly create structure themselves.

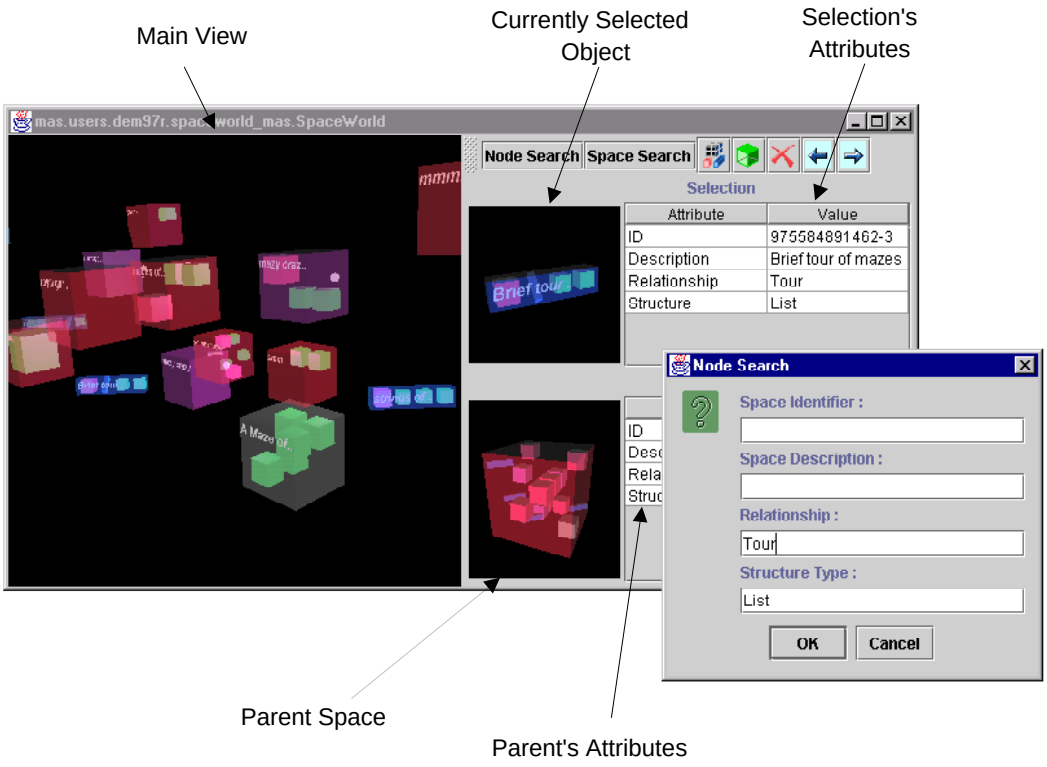

Figure 8.3: The Spatial FOHM client

To represent the Spatial structures I chose to use a three dimensional display where the users could see through one level of structure (i.e. could see inside any composites) and could navigate by ‘zooming in’ to any composite by double clicking on it, or could zoom out by double clicking on the empty background. In addition to the main display users could also see two smaller three dimensional views. One of which continually showed the parent composite in which their view currently resided and the second of which showed the selection of the moment (made by single clicking on an object). Both these secondary views also show the attributes of the objects in question. Figure 8.3 shows a screen shot of the Spatial Client, displaying structures retrieved from the FOHM storage agent. The search dialog (on the bottom right hand side of the figure) provides a mechanism for users to find spaces within the system to act as starting points.

The client’s display was chosen to complement the 6D model described in section 6.3.1. Here, as in previous figures representing that model, different structures are shown in different colours and shapes. Using this type of display means that the only spatial characteristics available for the type of explicit internal structure described in Section 7.3.4 are structural ones such as ‘position’, since colour and shape are ‘used up’ by the display engine.

As previously mentioned creation support is limited by the lack of a spatial parser.

However the user can choose to create a new node or composite within the current parent object (or without a parent if none is currently being displayed). The Spatial client understands three types of structure:

1. *Sets* which it renders as red cubes with the content's positions determined by a hash value derived from their id.
2. *Lists* which it renders as rotating blue tubes, square in cross section. The contents' position is determined by the feature vector of each content item.
3. *Matrixes* which it renders as grey rectangular prisms, whose contents' position is determined by the feature vector of each content item.

If the client encounters an object with a different structure type then it renders it as a Set.

8.2.5 The Navigational Client

The Navigational Client was designed as a general purpose Client capable of handling an extensible set of media types, initially image and text documents. The display of the client is split into two sections. The right hand pane displays the document contents and any endpoints contained in that document. The smaller left hand frame displays the Navigational gateways open to the user at the current time. These may be the endpoints of any selected links or other documents associated to the current one by spatial structure. Figure 8.4 shows a screen shot of the Navigational Client, displaying a document retrieved from the FOHM storage agent.

The navigational options displayed on the left hand pane fall into three categories:

1. *Selected.* This represents the endpoints reached by following the current selection - by clicking on different hotspots in the right hand pane these options change according to a traversal operation applied to the selected endpoint
2. *Organizational.* These represent the association objects that contain the current document. Under each association the endpoints reached under a traversal operation are listed. For example with a list structure the next and previous endpoints are available.
3. *Perspective.* This represents the different views that there are of this object. Under each individual perspective the alternatives are displayed according to follow link rules. However as the Navigational Client currently has no notion of what a profile should be, it simply displays all the different perspectives.

A user can navigate to another document (or space) by clicking on it in the navigation window.

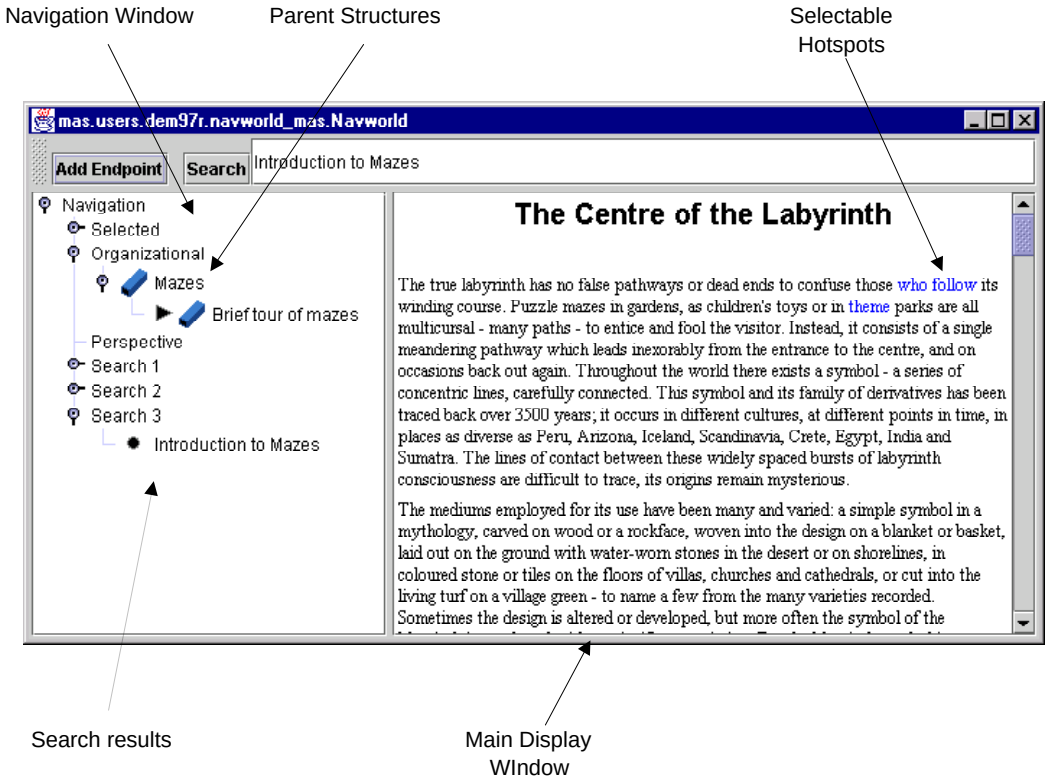

Figure 8.4: The Navigational FOHM client

The Navigational Client also allows users to make queries about nodes in the system, the results of these searches are made available as additional categories when the results come back. This mechanism is an ideal way for users to find a starting position within the hyperstructure.

8.3 Cross-Domain Browsing

The main point of implementing the two FOHM clients described above was to explore the way in which the two applications might view each other's data. As might be expected the most interesting aspects of this occurred when either client was browsing structures that would not normally appear in its own domain.

8.3.1 Browsing Spatial Structure

Figure 8.5 shows three screen shots of the clients, as the user navigates through a spatial structure to arrive at a navigational view of a node.

The first screen shot shows a view of a list, this contains two nodes (at the beginning and end) and also three sub-spaces. By double clicking on the first of these three, a list named 'Brief Tour of Mazes', the user 'zooms' inside it and can now see its contents.

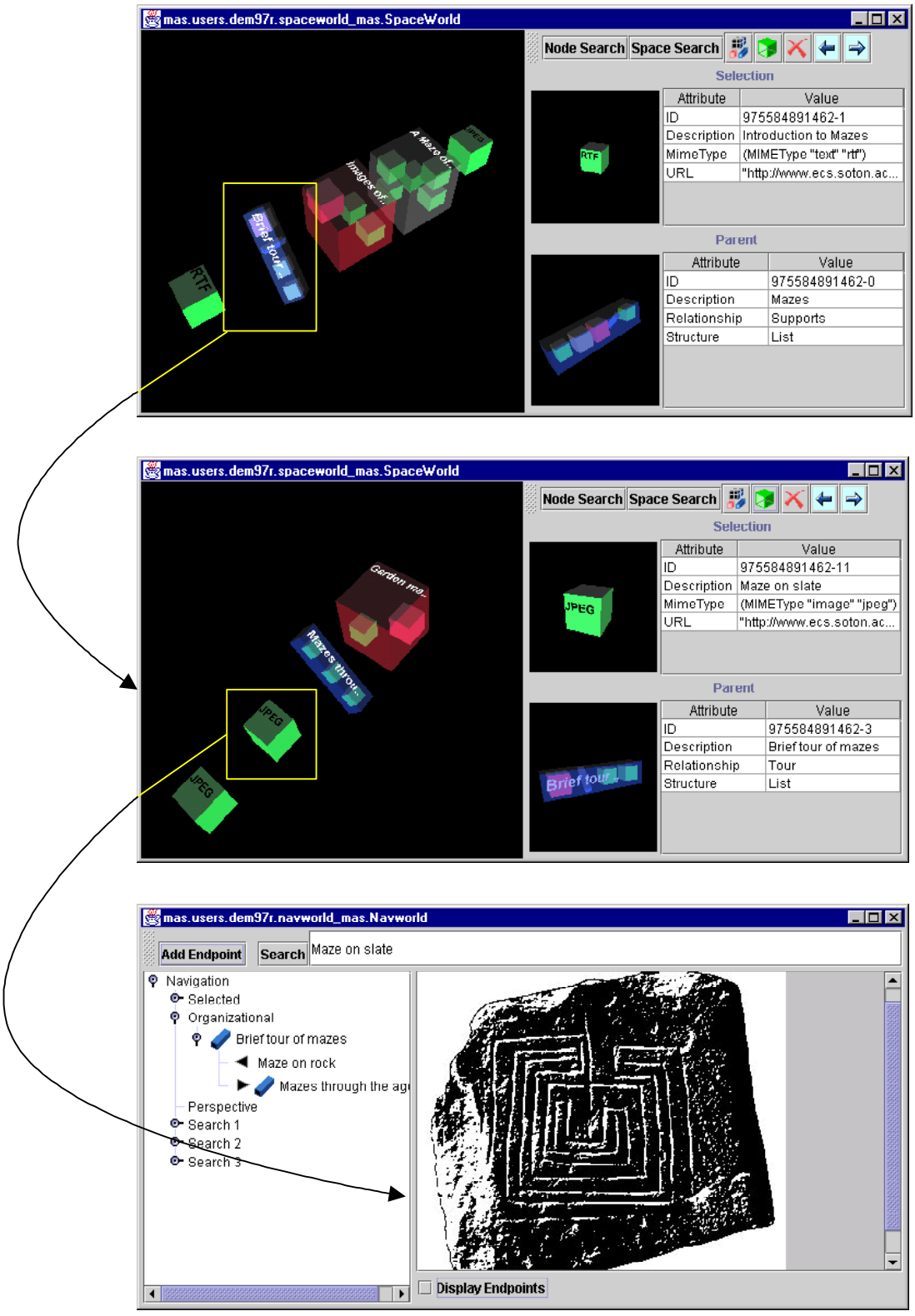

Figure 8.5: Navigating through two Spaces to a Node

The second screen shot shows this view, notice that the parent object shows the ‘Brief Tour of Mazes’ list.

By selected one of the nodes displayed the user opens up the Navigational Client and displays its contents. This is shown in the third screen shot. Since the Navigational Client treats spaces as links, the ‘Brief Tour of Mazes’ list is displayed in the left hand pane. Traversal semantics are applied to the list, resulting in two of its bindings (link endpoints) becoming available. The first takes you back to the previous node, a JPEG image called ‘Maze on a rock’, while the second takes you forward to the next node, in this case another list, ‘Mazes through the ages’. Both these objects were clearly visible to either side of this displayed node in the second screen shot.

8.3.2 Browsing Navigational Structure

Figure 8.6 shows two complementary screen shots from each client. The first shows the Navigational client in the process of creating a link. The user has selected a region (displayed as a box, superimposed on the image) and has added that to a list of current link endpoints in the Link Creation Pane (these will become bindings in a new association object). When they have finished adding endpoints the user can click the ‘create link’ button to send a StoreAssociation request to the FOHMServer with the completed structure.

Given that the binary link shown in the first screen shot has been successfully created we can view it by opening the Spatial Client and searching for all spaces with the structure type ‘Set’. The second screen shot shows the results of that search, the highlighted set is the link that the user has just created.

8.3.3 Issues

These examples show that both clients can interpret each other’s structure when it is depicted in the FOHM model, however there were a number of issues that had to be resolved before they could be implemented.

Levels of Structure

One of the main things that had to be decided regarding the Navigational Client was how to display the structures in the left hand pane; when these structures contained other structures, how many levels should be opened?

I decided that the client would allow only one level to be expanded. I thought that to enable the viewer to browse the entire hierarchy would be far too similar to the functionality of the Spatial Client. This actually highlights an interesting difference between the

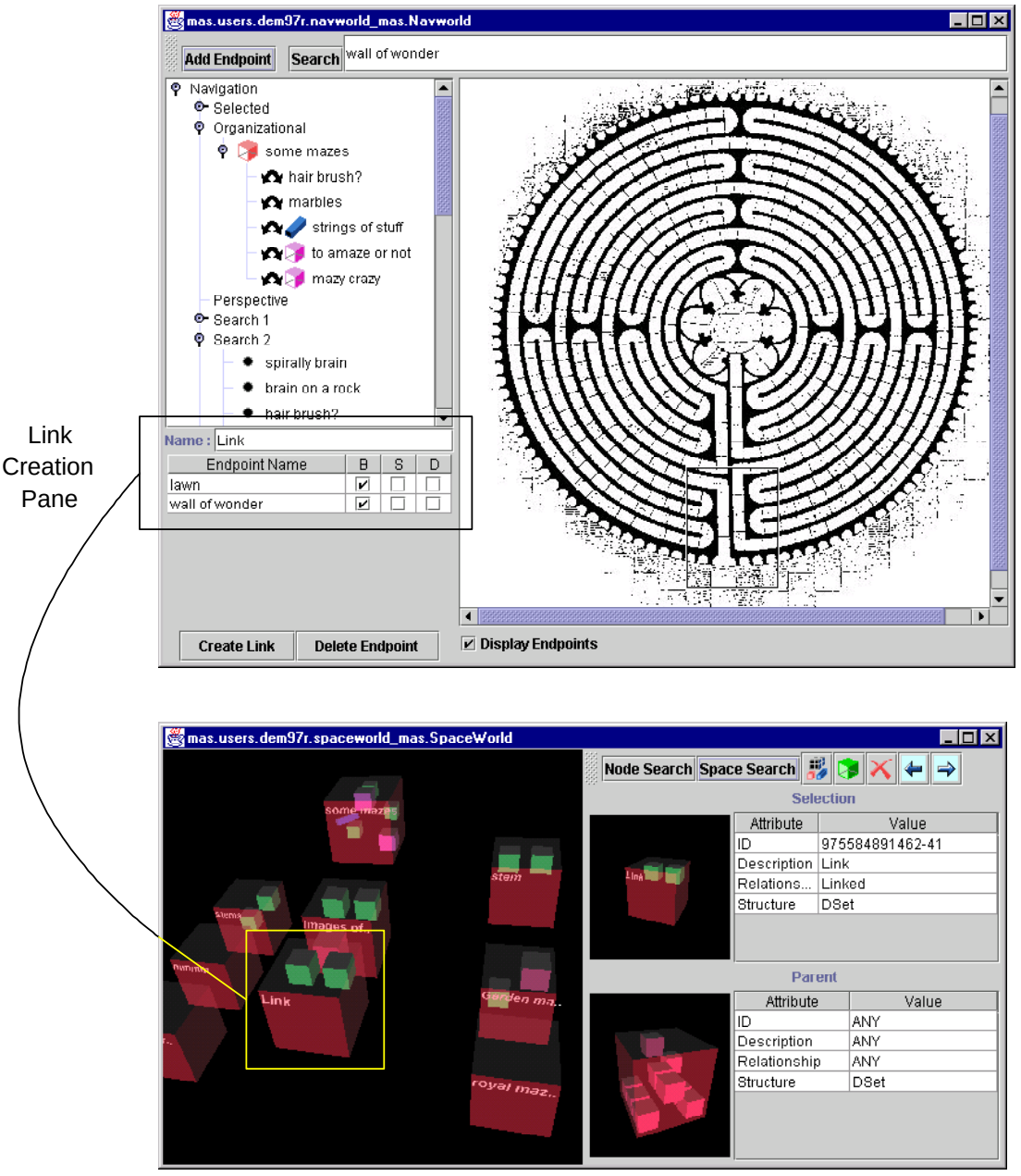

Figure 8.6: Creating a Link and Viewing it as a Space

two clients (and two domains).

In the Spatial domain a user can explore down a spatial hierarchy (by zooming into sub-spaces) but at any one time they can only go up to a *single* parent space. On the other hand in the Navigational domain the user can see *multiple* parents, effectively by asking the question ‘who links to me?’

It is a reflection on the cross-domain nature of the FOHM structure that the boundaries between the clients are blurred. There is nothing to prevent a single client being built that combines the functionality of both.

The Display of Anchors and Perspectives

During the implementation process some choices had to be made concerning things that the FOHM Spatial Client had to cope with that other spatial tools do not. The first two of these choices were presentation ones; how was the client going to display References that referred to part of a node (normally only encountered in Navigational Hypertext) and how was it going to display perspective objects (normally only encountered in Taxonomic Hypertext).

I chose to display these References in exactly the same manner as References to entire nodes. It would be trivial however to create some sort of visual marker to distinguish the two. The more difficult choice was how to view perspectives, does the spatial user see the perspective object itself, or is the perspective automatically resolved into one of its constituent parts?

In fact this is part of a larger question; does the Spatial Client simply show the structure to the user, or does it invoke arrival semantics at any point to resolve that structure. In the case of a perspective association that follow link would resolve to be one of objects inside the association based on context, in the case of a list association it would resolve to be the first item in the list, (essentially behaviour determined by that associations arrival and traversal rules (see Section 8.2.5)).

Applying Arrival and Traversal Semantics

It is interesting to look at where I used arrival and traversal semantics within both clients.

The Spatial Client never invoked either of these semantics, preferring instead to display structures in their entirety. The semantics belong to the Navigational domain.

The Navigational Client invoked traversal semantics when following a link. Either when the user clicked on a hotspot in the document or when they expanded one of the

```
<term name="FollowLink" extends="Predicate">
<field type="Binding" name="source"/>
<field type="Association" name="association"/>
<field type="Binding" name="destination"/>
</term>
```

Figure 8.7: FOHM FollowLink Predicate

nodes in the tree to see which documents shared its parents spaces. However my choice not to expand sub-trees meant that arrival semantics were never applied, as whenever a user selected a space, that space was shown in the Spatial client instead.

Implementation concerns aside there is a more fundamental question: *How* do you apply these semantics?

Associations are resolved by querying other agents in the system with Association predicates that match a certain pattern (i.e. Associations that anchor on a particular document). It is the FOHM Storage Agent that applies traversal rules to associations it knows about and returns the results (a subset of the bindings). In this implementation it used the following rules:

1. *Sets or Links* Given that the start endpoint is ‘bi’ or ‘source’ it returns all the other endpoints in the association that are either ‘bi’ or ‘dest’, excluding the start endpoint. Members of Sets are considered to be bound exclusively as bi-directional.
2. *Lists* It returns the endpoints to either side of the starting position. I.e. if a follow link is made from position two, it returns positions one and three.
3. *Matrixes* It returns the endpoints ‘nearest’ to the starting endpoint, based on a concentric search of the matrix.

These semantics are best placed in the server to allow it to make an absolute judgment on what the user can or cannot see, but there is nothing to prevent clients from applying their own rules or filters. However there is a problem. When the FOHM Server is queried about Associations, how does it know whether the querying agent wants an overview (like the Spatial Client), traversal semantics (like the Navigational Client) or arrival semantics?

I avoided this problem with adding a task based predicate to the FOHM ontology, shown in Figure 8.7.

This predicate states that when the *source* binding is followed across the *association* it resulted in the *destination* binding. Now the FOHMServer can respond to queries exhaustively, while responding to FollowLink queries by applying traversal semantics.

In the later case, it replies with one completed FollowLink predicate for each destination binding.

8.4 FOHM as an Interoperability Approach

Given that the FOHM model has been described and a prototype implementation presented, it is now time to think about FOHM in relation to interoperability and other interoperability efforts.

8.4.1 FOHM and Structural Computing

In Section 4.2.1 I described a new approach to information systems known as Structural Computing, where hypermedia is recognised as a special case of the more general philosophy of dealing with structure. I also described a type of system known as a Component-Based Open Hypermedia System (CB-OHS) which supported this philosophy.

What is FOHM in Relation to Structural Computing?

A great deal of the work undertaken so far in Structural Computing is based around a software system known as Construct (Wiil & Nürnberg, 1999). Construct is the codebase successor of HOSS (Nürnberg, 1997), it is a CB-OHS that attempts to provide an extensible open hypermedia platform based on the latest OHSWG standards.

In Construct the middleware layer is opened up into an extensible set of *structure servers*. The servers use a common back end storage facility based on generic structural primitives and then offer some set of functionality over that structure via a specific API (or protocol language). For example a Navigational Structure Server might offer navigational abilities, while a Spatial Structure Server would offer an API specialising in spatial functionality. Additional work has also been done to investigate the ways in which the structures served might by specified using templates (Vaitis *et al.*, n.d.) in order to define the acceptable structures explicitly and in the extreme case, even tailor the functionality of the server.

The multiple Structure Server approach does not guarantee cross-domain structures as the structures stored beneath the servers are not necessarily continuous. But it has been argued that such support could be provided by placing translation functionality into each structure server. Thus a navigational Structure Server might also deal with and translate spatial structure.

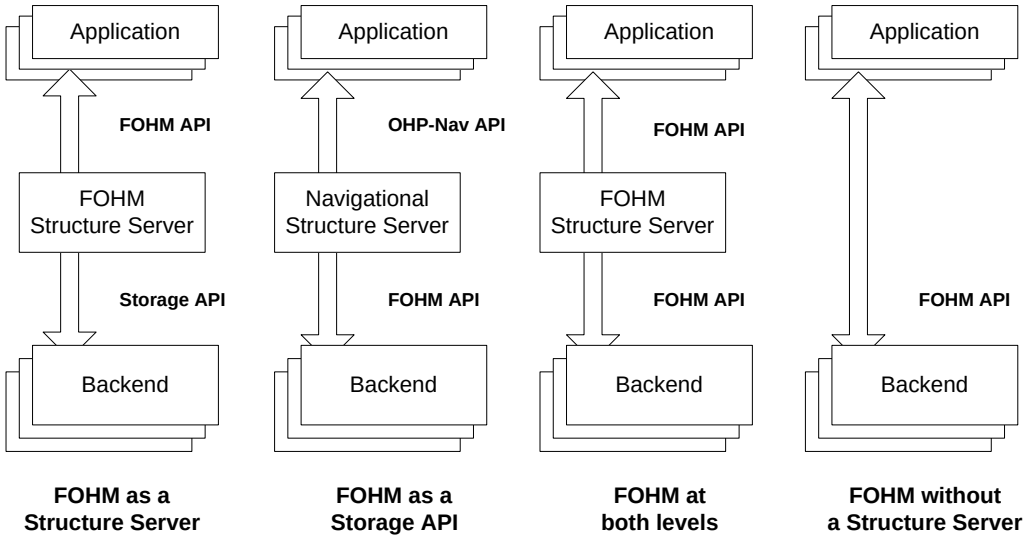

Figure 8.8: Integrating FOHM with Construct

This has the advantages of the translation approach (i.e. extensibility) mentioned in Section 7.2 and also places the functionality in a convenient single location, allowing lightweight clients to be built.

Combining FOHM and Construct

FOHM is a model that requires a syntactic implementation before it can be used. As FOHM places no restrictions on architecture it should be entirely possible to use FOHM within the Construct environment. There are two places where it could arguably be incorporated (see Figure 8.8).

1. *Structure Server API* A FOHM based protocol could be supported by a dedicated Structure Server and supported on a par with more specific protocols such as OHP-Nav.
2. *Back end API* Alternatively a FOHM based protocol could be used by the Structure Servers and their common store to communicate.

Firstly we consider the case where FOHM replaces specific domain protocols. Thus rather than a Navigational, Spatial and Taxonomic Structure Server we provide a single FOHM Server.

I talked extensively about the Information Continuum in Section 6.3 and argued that rather than think of these domains as independent of one another we should now consider them part of a larger continuous information space with a common structure. In this way users can navigate seamlessly around the structure and problems such as context can be

approached consistently across all domains at once. This multi-domain structure should not be hidden from clients - it should be exploited by them.

For this reason I believe that FOHM makes a much more powerful structure server for hypermedia than a collection of individual domain servers.

Secondly we have to consider the idea of using FOHM as a storage model. The only problem with this is that FOHM is specialised to deal with associations, found in hypermedia structures (of all domains), while structure servers are designed to deal with any form of structure. As such it is an ideal replacement for separate hypermedia domain protocols but not necessarily ideal for an arbitrary structure storage protocol, although it has been claimed that typed associations are as expressive, and in some ways more expressive, then meta-data (Moore & Moreau, n.d.).

FOHM was based on the premise that to be practical, a model would have to represent the highest common structure across the domains considered. Only by representing the highest common structure could the performance penalties demonstrated by our XML experience be avoided (see Section 4.5). Due to this I would argue that an arbitrary storage backend is too general to be deployed effectively in a hypermedia environment. FOHM, on the other hand, provides only the structures required.

For this reason I believe that FOHM provides a much more practical service for storing *hypermedia structures* than an arbitrary storage back-end.

Is FOHM a General Language for Structure?

If we argued that FOHM should be used at *both levels* of a CB-OHS then we would be effectively relegating any Structure Server to an optional middleware filter (translating one type of FOHM structures into another), in a similar way to the CSF in the original OHP definitions (see Section 3.1.2).

In an analogous way we could argue that there is no need for any ontologies in the SoFAR framework other than FOHM, as all statements of fact (predicates) can be represented in the FOHM model.

For example consider the ReverseString Predicate discussed earlier, that states that the reverse of ‘hello’ is ‘olleh’:

```
ReverseString("hello", "olleh")
```

This can be represented in FOHM by two data objects bound to an association of type ‘Reverse’. We can even add further information by binding ‘hello’ with the feature

‘forward’ and ‘olleh’ with the feature ‘reversed’.

However, FOHM is not a general model for discussing structure, it is designed around navigable hyperstructure. For this reason I believe that it is best used to unite the Structure Servers (or different taxonomies) that would be otherwise associated with the different hypermedia domains. Although this does somewhat reduce the argument for Structural Computing the two are not mutually exclusive.

8.4.2 The Semantic Content of FOHM

FOHM is an attempt to define a semantic language, a common vocabulary which components (agents) can use when communicating. However there are issues concerning the actual semantic contents of the model in regard to conveying behaviour.

The OHP-Service protocol described in Section 4.3 worked by encoding services as opaque black boxes with defined input and outputs. A client could request a list of all services from a server and then display the ones it understood. In contrast the SoFAR agent framework allows a component to ask if any other components support a particular service. In other words components are expected to be proactively seeking out services that they understand.

The difficulty comes when the worlds of service invocation and hypermedia collide and services become attached to associations, i.e. the result of a Follow Link on an endpoint will be the invocation of a ‘Find Similar Images’ service.

It is at this point that it becomes necessary to return to the problem discussed in Section 4.6, what is the difference between a query, an operation and a service?

Where are the Computations?

It has been pointed out that some users require a great deal of control over their hyperstructures, to the point of changing the behaviour attached to those structures (Rosenberg, n.d.). In FOHM terms this would mean that the actual *arrival* and *traversal* semantics of an association would be attached to that association in some way so that clients could interpret it directly, rather than using one of a pre-defined set of behaviours chosen according to structure type.

In OHP-Nav we produced a Service definition that allowed clients to access arbitrary functionality with a standard, OHP consistent interface (see Section 4.3) but this was never formally attached to the follow link functionality enshrined in the protocol due to lack of a proper event model for an OHP system (i.e. at what point in a ‘follow link’

process is the service activated). As a result Services became separated from the hyperstructure and OHP-Service became a separate protocol for arbitrary service discovery.

A third view of behaviour is given by the CB-OHS architecture (see Section 4.2.1). Here behaviour is assumed to be a middleware or server side process. Extensible behaviour is provided by allowing systems designers to add to the set of behaviours.

Along with these notions we also have two separate ways of accessing the hyperstructure, a query language and dedicated operations such as 'Follow Link', which we can assume activates this behaviour in some way. One of the major challenges ahead is to unite these three views of behaviour and decide at which point a query becomes an opaque operation - or a behaviour activation request.

We can begin by identifying two types of such computations and providing defined terms with which to discuss them:

1. *Services* are designed to be opaque to a client, such as the OHP-Service computation objects. Services are provided externally to a client and are invoked rather than processed.
2. *Behaviours* are designed to be seen explicitly by a client, understood and then interpreted. Examples of such behaviour include gradual fades between linked nodes (Rosenberg, n.d.) and transclusions (Nelson, 1987).

Services

It is clear that current systems go someway to address the first of these two issues. In CB-OHS architectures new Services can be provided via additional Structure Servers, while a semantic language like FOHM allows components that are not hypermedia based to express their information in a hypermedia language (i.e. we do not need a 'Find Similar Images' service if that component can export its functionality via FOHM).

One question is, do we continue to implement operations like 'Follow Link' separately when a 'generic language' is available (i.e. a query is not expected to be exhaustive and the results may be subjective)? Perhaps queries and Follow Link requests are just queries with different user contexts? This seems more powerful and natural - as we are just asking other components about the structures that they know about.

The one aspect of Services that is not yet solved and remains an issue is the ability to integrate the hyperstructure itself with arbitrary services. I.e. linking to the 'Find Similar Images' service rather than a defined list of images enables a user to link to all documents that resemble a particular image.

Behaviour

Behaviour is rather under supported in both the FOHM and CB-OHS worlds. Both of these approaches rely on the types within the structures to map to a set of pre-defined and known behaviours. For example a transclusion could be implemented in FOHM by giving the association the type ‘transclusion’. Any clients that understood what that meant could treat the link appropriately.

A more advanced notion of behaviour would allow the behaviour itself to be specified in some way and attached to the structure. Informing compatible clients how to use arbitrary behaviour. Such a specification could be given in a scripting language, or even a binary format such as Java byte code.

8.5 Summary

In this chapter I have drawn from the discussion of syntax and semantics to produce a prototype cross-domain system based on the FOHM common semantic language.

The functionality of the two clients implemented show that there is no clear divide between the navigational and spatial domains, rather there is a continuous and gradual change between them.

In many ways the FOHM work has come full circle and arrived at the hyperbase roots of hypermedia research, evoking memories of early hypermedia models and systems such as Hyperbase (Schutt & Streitz, 1990) and HB1 and its derivatives (Schnase *et al.*, 1993). It has done this via an alternative route of reasoning about cross-domain compatibility for purposes of interoperability, inspired by the OHP work of the OHSWG. As such it is important to realise that while FOHM does not necessarily complement the architectural models of recent CB-OHSs, such as HOSS or Construct, it still acts as a validation of the philosophy of Structural Computing.

This philosophy states that structure, and separate behaviour across structure, forms the core of a general class of structural systems of which hypermedia is only one example. FOHM is a semantic language for the discussion of hypermedia structure of all forms, which fits into a communications paradigm of propositions and performatives which I believe is much more powerful and flexible then traditional protocol approaches.

Chapter 9

Conclusions

> *"The applications of science have built man a well-supplied house, and are teaching him to live healthily therein. They have enabled him to throw masses of people against one another with cruel weapons. They may yet allow him truly to encompass the great record and to grow in the wisdom of race experience. "*
>
> VANNEVAR BUSH, AS WE MAY THINK

Hypermedia is maturing as a research topic, the World Wide Web has brought its navigation paradigm to millions of users and helped integrate information technology with society in ways that were unimaginable only fifteen years ago.

However, it now seems that the calls for integration and standards that occurred at the beginning of the 1990's were premature. Mobile phones, Personal Digital Assistants and other new devices are changing the technical basis and assumptions of our systems beneath us. Coupled with the recognition of hypermedia as one aspect of a more general Structural Computing philosophy this has upset our notions of what it means to interoperate.

The interoperability effort of the OHSWG has attempted to cope with these rapid changes and realisations by dividing hypermedia systems into different domains and by tackling each domain differently, but consistently.

In this document I have taken a detailed look at the nature of the information spaces that we are attempting to model (from all these considered hypertext domains) and have incorporated notions of behaviour and context. This has produced a cohesive and continuous space of spaces that I call the Information Continuum.

It is the thesis of this document that the different hypermedia domains are artificial distinctions imposed upon this underlying continuum, and that this continuum can be represented in a single, extensible data model that is suitable for use in the common language approach to interoperability.

Further to this, the common language approach is presented not as a flawed, finite or over generalist ideal, but as a sensible and coherent way for components to interoperate and discuss relational structure for the purpose of navigation.

The Fundamental Open Hypermedia Model, FOHM, has been presented as one such common language and an implementation has been described that demonstrates the model being used for cross-domain purposes.

Finally, it is believed that much has been learned about the way in which components communicate. The human linguistics world has been found to have relevancy and lessons learnt from that world by the agents community have been successfully employed by the implementation of FOHM via the SoFAR framework.

9.1 The Threads of Research

As in many large, technical documents there has been a great number of different ideas and arguments presented in this thesis. One of the great ironies of hypertext and information system research is that the systems conceptualised would make the research process itself much more structured and coherent if those systems could ever be employed.

Indeed it is most unfortunate that a hypertext thesis cannot be submitted. However, hypertext as an idea predates the electronic machines that now enable it by at least a decade or so. In addition many linear texts can be thought of as ‘flattened’ hypertexts, that can be re-evaluated by those who revisit them in the light of the hypertext paradigm. It is in this spirit that I would like to draw together the threads of research that run through this document.

Each of the following sections briefly describes one of the threads that cuts across the chapters of this thesis. Each also presents the section numbers so that they can be revisited by hand by a reader, or automatically followed by those who have an electronic (hypertext enabled) version of this document.

9.1.1 Interoperability Standards

The desire to facilitate the interoperation of hypermedia systems was the original driving force behind this work. Drawing on the experience of researchers in the field, the Open

Hypermedia Protocol (OHP) became an ambitious project that forced the researchers involved to look beyond their individual research interests and try and understand the basic nature of information and its navigation.

An in depth exploration of what it means to communicate has revealed that the key to interoperation is not syntactical, although a communication infrastructure must exist, it is semantic. Components must agree on *what* they are communicating.

This desire for interoperation is as old as hypermedia itself. Section 2.2 describes how the early hypertext pioneers, Bush and Nelson, both wanted a world corpus of knowledge held within a global information system.

Section 2.5 explores some of the hypertext models that have been generated since then, the Dexter, Amsterdam and Trellis Models. These were based around notions of defining common terms, or analysing hypertexts rather than interoperation. The flag taxonomy and its requirements for interoperability are presented in Section 2.9.2 along with Sun's Link Protocol, an early attempt at standardisation that unfortunately was never adopted.

Section 2.9.3 presents the original OHP protocol, which was criticised for its lack of an underlying data model or architecture. Section 3.1 explains how as a result of these criticisms the protocol was divided into more finely grained domain specific protocols, OHP-Nav, OHP-Space, etc. This resulted in the first ever demonstration of interoperation between Open Hypermedia Systems at Hypertext '98 in Pittsburgh, described in Section 3.2.

The view that hypermedia is only one case of the more general philosophy of Structural Computing is presented in Section 4.2.1 which also describes the Component-Based Open Hypermedia Systems (CB-OHSs) that were beginning to appear at the time. Other issues were also being raised and Section 4.2.3 discusses naming as an important requirement for interoperability.

In my own work I was already dividing the problem of interoperation into two areas. The first was based around syntax and infrastructure. Section 5.2 looks at the mechanism with which components communicate (i.e. API vs. on the wire mechanisms). While Section 5.3.2 discusses how discourse takes place between human beings. I ultimately turned to the agents community for techniques to simulate this in software and Section 5.3.3 introduces the notion of performative and proposition based communication.

The second area concerned the semantics of communication, how meaning is conveyed, and what those semantics should be. The Information Continuum, presented in

Section 6.3.2, is a demonstration of how hypertext semantics about navigation and behaviour are common across all domains.

Section 7.2 looks at the ‘translation’ verses ‘common language’ approaches to conveying this continuum, and argues that a common language, represented by the Fundamental Open Hypermedia Model (FOHM) offers advantages in flexibility and representational power.

An implementation of FOHM was developed using the SoFAR agent framework as the common infrastructure (described in Section 8.2) while Section 8.2.2 explains how FOHM was represented using SoFAR predicates,

What makes FOHM an interesting case of interoperation is the ability for browsers of one domain to access the structures of another. Section 8.3 demonstrates this cross-domain browsing using the FOHM agents developed with SoFAR.

Finally, Section 8.4.1 looks at how FOHM fits into Structural Computing paradigm and the CB-OHS architecture and Section 8.4.2 examines the ability of both approaches to convey semantics for the purpose of interoperation, particularly the behaviour associated with link traversal.

9.1.2 Cross Domain Interoperability

An important contribution of this thesis is to help expand our notions of what it actually means to interoperate. I believe that the multiple OHP protocols create an artificial distinction between different kinds of hypermedia structures. FOHM is an attempt to reunite these structures. Structures that have always appeared in hypermedia systems and models but have not necessarily been recognised.

Section 2.4 describes Halasz’s seven issues. One of which is on the requirement for composite structures and an open question on whether they are the same as links. In addition Section 2.8.3 presents three views on context in hypermedia, the third of which is context as a filtering system on what you can see.

The idea of whole hypermedia systems based around composite structures is relatively recent. Section 2.8.2 introduces spatial hypertext as an alternative approach to information navigation and management.

When the question was raised of how OHP should deal with this new view of hypermedia it was decided to break the protocol into different protocol domains as described in Section 3.1. This idea of different services over structure was later used as part of the evidence supporting the paradigm of Structural Computing described in Section 4.2.1.

Section 6.3.2 introduces the Information Continuum, the notion that navigation and behaviour are common across all domains, and Section 7.4 describes what each domain brings to the common model, FOHM, to make it capable of representing that continuum.

Section 7.4.1 describes how the Navigational domain introduces the anchor, Section 7.4.2 how the Spatial domain includes classification within associations and Section 7.4.3 how the Taxonomic domain draws in perspectives and how these form a framework for context.

The FOHM Ontology is presented in Section 8.2.2 in the terms of the SoFAR agent framework and Section 8.3 demonstrates cross-domain browsing using the FOHM aware software clients.

Returning to the question of Structural Computing, Section 8.4.1 debates whether FOHM is a model suitable for representing *any* structure, but concludes that although it seems possible, such solutions are an inelegant misuse of FOHM, which is intended for navigable hyperstructure. As a result, although FOHM detracts somewhat from the argument for different structure servers (by uniting the hypermedia domains into a single structure server), it does not oppose it, as other non-hypermedia structure servers might exists.

9.1.3 Communication Infrastructure

In this thesis I have described how the key to interoperation is not syntactical but semantic. Components must agree on *what* they are communicating. However, for a system to exist there has to be a syntactical implementation, components must agree on *how* they are communicating. As part of this work I have looked at the architecture and communications paradigms of existing systems, and contrasted that with an exploration of human conversational techniques. This later work led me to use an agent infrastructure for my implementation of FOHM.

Section 2.3 describes the first working hypertext systems. These early systems (ZOG, Notecards etc.) were monolithic in nature with very little, if any, interaction with external applications or systems.

Section 2.4 describes how one of Halasz's seven issues, which were described as reflections on Notecards, concerns how future systems should be extensible and tailorable. This was an issue that hypermedia research community took to heart and Section 2.6 describes the development of open systems, such as Microcosm and DHM, which were capable of integrating with other components.

When OHP was written it was already thought that it would form part of an open system. Section 3.1.2 describes the architectural assumptions of the original OHP. In particular its dependance on the CSF, not only as a shim for non-compliant systems but also as a nexus for hypermedia functionality in the system.

A CSF was a crucial part of the Southampton OHP system described in Section 3.2.1 which was used for the Hypertext ’98 demonstration of interoperability. Configuring the CSF was non-trivial and Section 3.2.2 describes the additional protocol that was needed to manage it.

Section 3.3 raises the question that if we have a CSF why do we not have an Server Side Function (SSF)? The answer seems to be that both components are optional in the system according to the functionality the programmer wishes to place into the clients. For this reason the CSF was dropped from the assumed architecture.

This architecture was expanded with the notion of a Component Based Open Hypermedia System (CB-OHS). Section 4.2.1 describes how the philosophy of Structural Computing introduces the notion of multiple structure servers (as opposed to a single OHP server) over a communal storage back end.

The architecture of the OHP-Service aware Solent system is described in 4.4.1. In the Solent system components register the protocols they can understand with each other and use a dynamic discovery mechanism to discover other components that speak the same protocol that they do. Section 4.4.3 describes how this dynamic component approach allowed us to run multiple versions of a protocol at the same time, as well as using the same infrastructure with different protocols (i.e. OHP-Service and OHP-Nav).

I was becoming convinced that the OHSWGs interoperability effort was becoming side tracked by the issues of architecture and protocol design. Section 4.6 describes how I came to think of such architecture concerns as separate from hypermedia ones.

Once this distinction had been made I was free to examine general communication infrastructures. The case of an API verses an on-the-wire approach is made in Section 5.2 and the idea of operations as a basis for communication is challenged in Section 5.3.3 which introduces the notion of performative and proposition based communication.

Section 8.2.1 describes the architecture of SoFAR, the agent framework used to implement the FOHM model, which uses this performative approach to allow components (or agents) to communicate in a dynamic, ad-hoc manner.

As a way of returning to this work’s OHP roots, Section 8.4.1 describes FOHM’s relationship to Structural Computing and how the model might integrate with the CB-OHS

architecture as either a structure server to deal with all hypertext domains, a common structure storage model or possibly both.

9.1.4 Services and Behaviour

Services evolved from a perfunctory inclusion in OHP to an entire protocol in OHP-Service. Throughout this work there was much debate as to what constituted a Service, e.g. was 'Follow Link' a computation? Via the examination of Structural Computing environments and the Construct server in particular, coupled with a greater appreciation of the role of a communication infrastructure such as SoFAR, we were able to properly distinguish between behaviour and services and evaluate how well they were both supported.

In Section 2.4 I described how one of Halasz's original seven issues was the support of computation objects within a hypertext system. For some hypertext systems this support is crucial, for example in Section 2.8.1 we saw how such a computation object might by used to implement media-based navigation.

The OHP data model presented in Section 3.1.1 originally supported computations via a limited SCRIPTSPEC opaque, described in Section 3.1.5. It was expected that either clients could interpret the script directly or that such functionality could be 'farmed out' to helper applications such as the CSF (Section 3.1.2).

With the splitting of OHP into separate domain protocols came a more sophisticated approach to Computations. In particular Section 4.2.1 explains how the Structural Computing approach views behaviour and structure separately.

Within the OHP work itself Services were now seen as separate first class entities called Computations (Section 4.2.2). The idea that Services were first class caused them also to be considered the subject of a separate protocol and in Section 4.3 I describe the OHP-Service protocol that I developed.

A first implementation of OHP-Service, an open set of components called Solent, is described in Section 4.4 including work on quality of service and composite computations. Although several issues were raised during this implementation the most important one was a question posed in Section 4.6 on whether functionality such as 'Follow Link' is actually a Service, a Query or an Operation.

Section 6.3.1 presented the 6D model, which incorporates behaviour as a dimension, indicating that functionality such as 'Follow Link' is a discerning part of a hyperstructure. Follow Link behaviour is mentioned again, this time in the context of traversal and

arrival semantics, in the discussion of spatial hypertext structure that takes place within Section 7.4.2.

The SoFAR framework introduced in Section 8.2.1 includes a dynamic service discovery mechanism based around the notion of components registering predicates and actions with a central registry. Section 8.3.3 describes how the FOHM ontology that uses that system had to cope with invoking different behavioural semantics via the inclusion of a 'Follow Link' predicate.

Section 8.4.2 returns to the whole question of Services verses Queries or Operations in the light of the work undertaken with FOHM and agent frameworks. Section 8.4.2 attempts to define what constitutes a Service as opposed to behaviour and explains how one way to unify Services, Queries and Operations would be to view them as contextual modifiers applied to a basic query mechanism.

Finally, Section 8.4.2 looks at a definition of Behaviour as opposed to Service and critically discusses the level of support for behaviour in both FOHM and Construct.

9.1.5 Context

There was always a notion amongst the OHSWG researchers that context was an important part of hypermedia and that OHP should support it, however the lack of any real contextual systems and conflicting understanding about what constitutes context, meant that agreeing on any standard for inclusion in OHP was impossible. It was only when I started looking at Taxonomic hypertext that I realised that context was going to be crucial to any general model of hypermedia. A full analysis of context is beyond the scope of this thesis, but what I have ensured is that FOHM is designed with the understanding of where context fits and when it is invoked. These are the user and data profiles and the 'magic function' that compares them.

Although it is not immediately obvious, this in itself is a major contribution as it provides a framework for context, a hyperstructure into which different notions of context can be placed and will function. Understanding what context is and how it applies to hypertext has not always been so well understood.

Section 2.8.3 presents the three views on context that had been voiced in the literature when OHP was first being developed. The first two of these were essentially link anchors and workspaces, but the third was the view of context held here; context as a filtering or adaptive mechanism, adjusting what a viewer sees within the system.

Context appears more formally under the guise of Perspectives in Section 6.2.3, which describes the way in which taxonomic trees can branch according to different

opinion.

The 6D Model presented in Section 6.3.1 describes context as an information dimension, alongside time and behaviour. Section 6.3.2 builds on this view and describes how navigation and context are common across all dimensions of the information space of all spaces (the Information Continuum).

With the realisation that the time and behaviour dimensions are just part of an n-dimensional context space came the idea, presented in Section 6.3.2, that context is a composite that contains all versions of an object. In essence it represents a common concept object. Section 6.3.2 draws a parallel between this concept object and a contextual link (a link across one or more dimensions of the continuum) leading to the conclusion that contextual links are the mechanism that binds objects together as a common concept across these dimensions.

Once context has been so placed within the hyperstructure it becomes possible to think where else it should be invoked. Section 6.3.3 examines the idea of temporally infinite media (such as a TV broadcast) as opposed to a finite stream. Whilst the later is orthogonal to real time (i.e. they do not share the same time line as the user or the hyperstructure), in the infinite case they share the same notion of time. This poses the question; are temporal anchors actually contextual anchors, anchored to the entire stream but within a context that defines there temporal position?

Finally, Section 7.4.3 describes the way in which the FOHM model has expanded the Taxonomic idea of perspectives into a framework for context that relies on opaque ‘profile’ objects (a user profile, a data profile and a ‘magic function’ to compare them).

9.2 Future Work

In this thesis I have presented an argument that moves across the working definitions of three communication protocols or models.

The first was OHP-Nav, the Navigational specialisation of the OHP draft proposal. The second was OHP-Service a further specialisation protocol for the definition and invocation of opaque Service objects. The realisation that such dynamic functionality belonged to a general communications infrastructure sparked an exploration into the Information Continuum which resulted in the third definition; FOHM, a model capable of supporting all three of the hypertext domains currently being considered by the OHSWG.

However there is still work that could be done based around FOHM. To begin with, before FOHM can be used as a basis for interoperability, or applied to the OHP work, it

is important that the structures and relationships that can be expressed by each domain are formalised, so that a client that wishes to be complete knows what it must support. In addition although the definitions of Behaviour and Context are beyond the scope of this thesis there is still much interesting work to be undertaken in those areas using FOHM as a framework.

Perhaps most interestingly for hypermedia researchers, the FOHM model has several unique features that prompt us to think about hypermedia in new and revealing ways.

9.2.1 Formal Definition of Domains within FOHM

In this thesis I have presented several different ways that FOHM can be used to represent the domains of hypermedia. In Section 7.3.2 I presented part of a formal definition of FOHM that has yet to be completed. This would form a formal proof that FOHM was capable of representing all three domains with no loss of semantics. Section 7.4.2 contained some suggestions on the structure types that might be supported by FOHM and Section 8.3.3 describes the sub-set supported by the FOHM implementation.

It would be advantageous to define an absolute set of structuretypes, featurespaces and appropriate bindings for each domain. This would not prevent new components using the FOHM model in different ways but it would form a well-defined set of hyperstructures that clients that wished to be complete in one or more of the domains could support. This description might also include advice on how unrecognised structures should be treated.

For each structure we need to define:

1. The type of the structure (i.e. ‘Link’)
2. The feature space of the structure (i.e. ‘direction’)
3. The vectors that may bind to that feature (i.e. ‘src’, ‘dest’ or ‘bi’)
4. The arrival semantics (i.e. show all endpoints)
5. The traversal semantics (i.e. show all endpoints, other than the start endpoint, that are bound with ‘dest’ or ‘bi’)

The definition of these structures would require some user analysis of the more structurally complex domain of Spatial Hypertext in order to identify structures that were most useful.

In addition it would be useful to define how some of these structures could be used, so that clients knew how to represent structures such as guided tours (i.e. as Lists with the relationship type ‘Tour’). For the Taxonomic domain this is critical as Perspectives

are represented as Sets with the relationship type 'Perspective' and Categories are represented as Sets with the relationship type 'Category'.

9.2.2 Behaviour

One of the research threads that I have described running through this document was one based around Services and Behaviour (Section 9.1.4). The notion of providing Services has been an important part of the OHP work from the beginning and I spent a significant amount of time defining and demonstrating the OHP-Service protocol.

I now believe that Service discovery and invocation is best provided by the communications infrastructure itself, rather than a hypermedia protocol. However there is still the issue of Behaviour as opposed to Services.

In Section 8.4.2 I defined Behaviour as something that is designed to be seen explicitly by a client, understood and then interpreted. Thus behaviour could be used to define the process of traversal (i.e. a gradual fade or transclusion).

At the moment Behaviour is implemented in FOHM via the use of the relationship type within an Association. A client manipulating a link must know what actual behaviour the relationship type is alluding to. For example a client must understand that when it sees the relationship type 'transclusion' (first used in Xanadu and described in Section 2.2) the link must be followed and the results inserted into the document view in place of the link anchor.

There are two ways to make Behaviour work in an interoperable fashion:

1. Standardise the types of Behaviour in the same way as structure. This would involve creating a list of Behaviour names and the functionality associated with them. This would be a contentious process within the community but a well defined list of a manageable size would prove enough to allow the majority of structures to be expressed, and the majority of systems to interoperate.
2. Work on a mechanism to define Behaviour. This would be somewhat akin to Service description in OHP-Service, where particular behaviour functionality was wrapped up in a Behaviour object. This functionality could be written as a simple script, or more powerfully as compiled code (perhaps Java bytecode).

My feeling is that this second option, while appealing, is too difficult to accomplish when the viewer being used is unknown. However it is possible that particular hypermedia viewers become particularly popular (such as Microsoft Internet Explorer, or Netscape Navigator) and Behaviours could be tailored specifically for them.

9.2.3 Context

Understanding exactly where context fits into the hyperstructures of FOHM has been a major part of this thesis. It has involved exploring the domain of Taxonomic Hypertext and defining exactly what constitutes a context. The FOHM view is that context is the sum of all information about the current browsing situation, seen by the system as a user profile, defining the context of the user and a data profile, which provides information on the structure.

However the actual contents of these profile objects and the functionality of the 'magic function' remains opaque. A definite future direction for FOHM research is to explore different definitions of context and see how user trails, which are otherwise part of the hyperstructure, can be included.

Another interesting avenue of possible research involves looking at the use of context as a replacement of the anchor in temporal media. This actually goes back to the idea of an anchor contextualising the endpoint of a link (presented in Section 2.8.3). It is interesting to look at where and why we might draw the line between defining an anchor in terms of a region, as opposed to anchoring on the entire document and allowing context to resolve those anchors we can view.

The same concept applies when linking to or from real world physical spaces. Should an anchor be defined on a particular region of the world (i.e. a particular room, or street) or should it be anchored on the world as a whole and the users context (position within that world) determines which anchors they see?

9.2.4 Symmetry of FOHM

FOHM is rare amongst hypermedia models in that it allows for *Implicit External Structure*, as defined in Section 7.3.4. This essentially means that links have their own arbitrary internal structure and that endpoints within a link can be related in different ways (e.g. as a list, matrix, etc.).

This means that an Association is an object with some internal structure which has Binding objects attached at particular points in that structure. A Data is an object with some internal structure which has Reference objects attached at particular points in that structure.

Are Associations and Data the same thing?

Figure 9.1 shows two FOHM structures, a link and a list, over four Data objects, three of which represent a sequence of movies and one of which is a review of the first of those

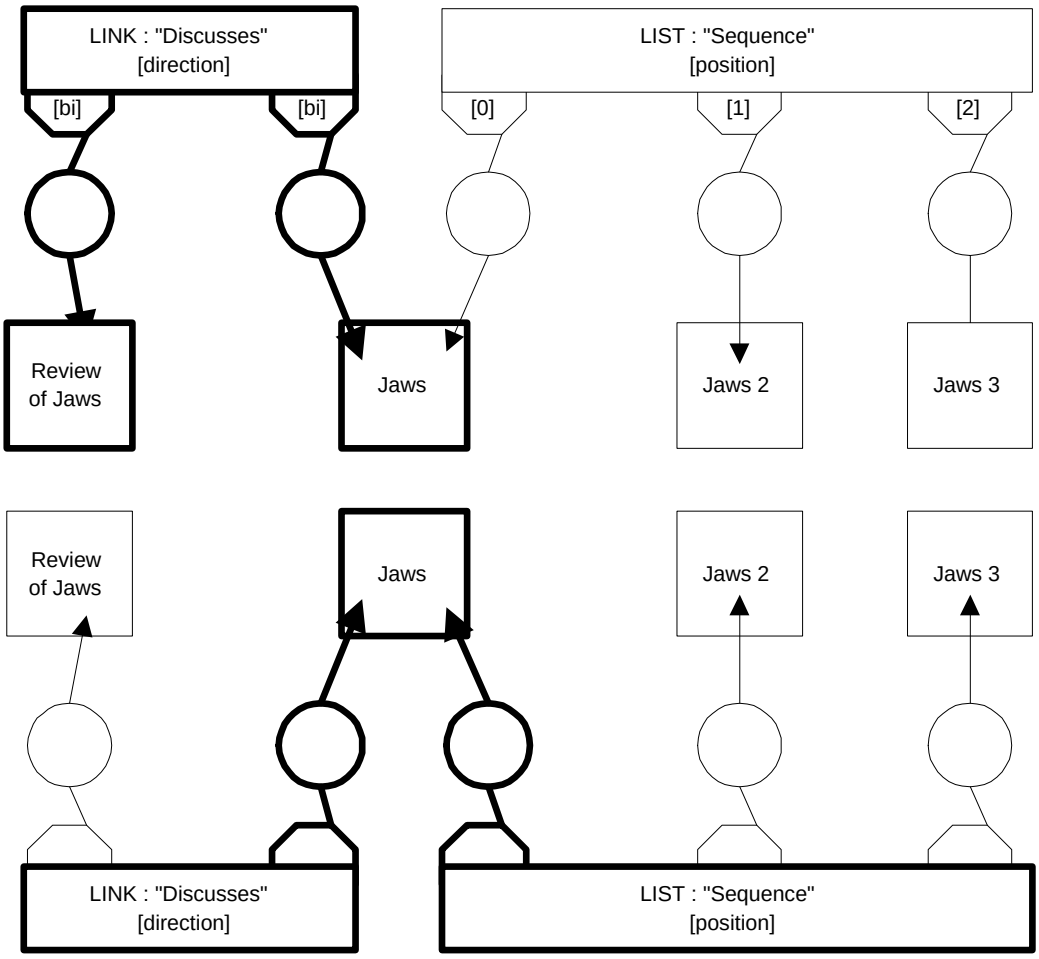

Figure 9.1: FOHM Symmetry

movies. But when the diagram is turned upside down it becomes apparent that the movie ‘Jaws’ is acting as a link between the two Association objects. It would seem that the only difference between a Data object and an Association is that Data objects have more complex internal structure.

Consider a HTML page which contains a bulleted list of embedded links to three movies (shown in Figure 9.2. In this case isn’t the HTML itself acting as a structured link between those documents?

There is a rather beautiful symmetry at work here that treats Data and Associations in exactly the same way. It indicates that the reason the Web got it wrong is not because it uses HTML as a data format but because HTML should be a link description format. One that includes lots of human readable semantics and presentation detail as well as some way to allow machines to understand the structure. The question I leave open for future research is whether or not emerging web standards such as XML might actually correct that mistake.

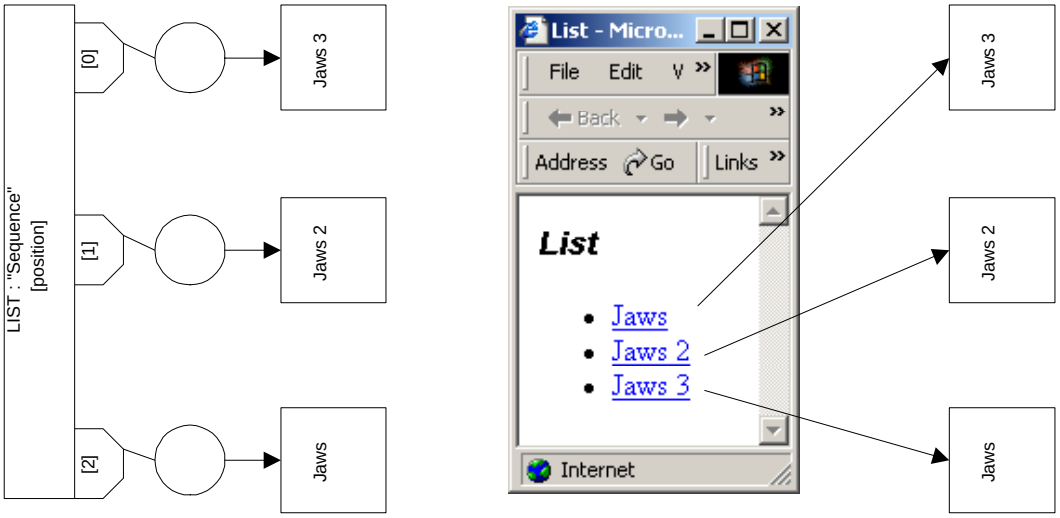

Figure 9.2: List as an Association and as a Data object

9.3 Interoperability in the Future

The work described within this thesis is based on the premise of the original OHP draft. That hypermedia clients should have a standard way of talking to hypermedia servers.

The pursuit of this seemingly simple goal has taken me down a long but profitable road. Firstly I have expanded the notion of what it means to interoperate. I have achieved this by abandoning the simple client to server assumptions of the original draft in favour of a more general notion of hypermedia components talking to one another.

Secondly I have broadened our ideas about communication. Recognising hypermedia as just one case of service provision over a universal communications infrastructure.

Most importantly I have examined hypermedia itself and explored the information space formed by the union of the various domains. I have called this the Information Continuum and produced FOHM, a model that is capable of representing all the structures in the continuum in a consistent and interoperable manner.

The use of hypermedia in the world is changing. Originally seen as a single machine application, the web forced hypermedia researchers and developers to think in distributed terms. In the future hypermedia will change again, as pervasive computing environments bring it to us at many different levels, on your desktop, on your television, on your PDA and on your mobile phone.

Throughout this change we must never lose sight of what hypermedia is all about. It is about helping people engage and interact with dynamic, living information. We can only ever achieve this if we focus on providing continuous and consistent structure whatever the domain, or the device. It is by understanding this continuum of information, and by

seeking ways in which people can navigate and comprehend it, that we take another step towards Bush's vision and finally start to encompass the great record.

Appendix A

OHP-Nav Definition - Extended Backus-Naur Form (EBNF)

A.1 The Definition Language

In order to define the mechanics of OHP we initially followed an Extended Backus-Naur Form (EBNF). We used the following constructs:

- `name = definition`: The name of a rule is simply the name itself and is separated from its definition by the equal "=" character.
- `"literal"`: defines a literal. The definition is case sensitive.
- {`elem1 | elem2`}: a "—" can be used to define either/or alternatives.
- {`elem`}*: A "*" indicates repetition, i.e. there can be any number of occurrences.
- {`elem`}+: A "+" indicates that one or more elements have to be present, i.e. there has to be at least one element.
- `[elem]`: Square brackets enclose optional elements.
- `<elem>`: Angle brackets enclose basic types that are not specified in more detail but can be dealt with by the receiving/sending components.
- ELEM: In the case where an element type is declared in capital letters this declaration can be seen as a placeholder that is defined somewhere else and just referred to at the current location.

Additionally we assume the following rules (the US-ASCII coded character set is defined by ANSI X3.4-1986):

- CHAR = US-ASCII character (octets 0 - 127)
- UPALPHA = any US-ASCII uppercase letter "A".."Z"

- LOALPHA = any US-ASCII lowercase letter “a”..“z”
- ALPHA = UPALPHA — LOALPHA
- DIGIT = any US-ASCII digit “0”..“9”
- CTL = any US-ASCII control character (octets 0 - 31) and DEL (127)
- CR = US-ASCII CR, carriage return (13)
- LF = US-ASCII LF, linefeed (10)
- SP = US-ASCII SP, space (32)

OHP messages will consist of text strings (CHAR). In the following sections the messages are shown formatted on multiple lines for ease of presentation. However, the messages themselves will be one continuous stream of ASCII text, unbroken by line feeds (LF). The messages consist of Tags, which are proceeded by a backslash (‘\’) and succeeded by a white space (SP). The characters that follow, up to the next tag or the end of the message are the tag contents. A tag content may be empty. In addition some tags denote a block of the message with some particular relevance. E.g. \LocSpec and \EndLocSpec.

The tags that should occur in each message are explained in the protocol definition. They may be presented in any order. The protocol is designed such that if tags are missing, the system will attempt to work with the remaining information, and if extra undefined tags are present, systems that do not understand these tags will simply ignore them. If a backslash (‘\’) occurs within the tag content, then it should be quoted by preceding it with a further backslash.

A.2 Message Header

As opposed to the earlier draft of OHP (Davis *et al.*, 1996), which defines a channel as an identifier for messages, we have defined a dedicated message header. As already mentioned above, OHP abstracts from the communication layer so the header does not include any host or port or other communication specific information.

Every message will have a message header. A message header will contain information about the transaction that both sides of the communication channel may from time to time need to examine. It is not immediately clear that all of this information is essential, but there is a general consensus that parts of this information will, at times, be necessary.

```
"\MID "  <a unique identifier for the particular message>
"\RMID " <the ID of the message this message is a reply to>
"\VID "  <current version of the protocol>
"\SID "  <a system dependant string which uniquely identifies some
```

```
          "transaction" or "session">
"\UID "  <a system dependant string uniquely identifying the user>
```

A.3 Opaques

There are certain entities within OHP that are treated as opaque (these appear within the protocol in uppercase). This means that their definition is unimportant for the specification of OHP itself although they will need to be understood by the applications using OHP. There are four opaques within the current protocol, LOCSPEC, PSPEC, CONTENTSPEC and SCRIPTSPEC). We expect a situation where a general definition of an opaque is needed (to allow interoperability) but where the protocol must be flexible enough to allow specialist applications to define their own standards for these opaques and to insert an appropriate byte stream; in order for such a byte stream to be placed within in ASCII protocol, it would be necessary to MIME encode the bytes.

So that an application (or the CSF) can parse all opaques, each opaque is accompanied by a version ID and enclosed within a begin and end tag. Thus a parser can check the version and skip the opaque if it is not one that is understood.

A.4 Messages

I will now present the messages in detail. Note that some of the messages might be rarely used, some even never. However, for reasons of consistency and symmetry of the protocol we defined and kept them as part of the standard.

A.4.1 Endpoints

Message "CreateEndpoint"

When a new endpoint has been created, the application program will send the following message to the link server. Generally an endpoint will require a dataref. Also the PSpec and Service tags may well be defaults (e.g. 'blue' and 'follow link' respectively). Note, that not only an application program, but also processes or scripts can create endpoints by sending this message.

```
MESSHEADER
"\Subject CreateEndpoint"
"\DataRefId " <DataRefId>
"\Direction " "Source" | "Destination" | "Bidirectional"
"\Service " <ServiceNmae>
```

```
"\PSpecId " <PSpecId>
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "GetEndpoint"

This message allows the application program to ask the link server for the current details of the endpoint identified.

```
MESSHEADER
"\Subject GetEndpoint"
"\EndpointId " <EndpointId>
```

Both messages, CreateEndpoint and GetEndpoint should be answered by the link server with an EndpointDef message.

Message "UpdateEndpoint"

Updates an endpoint's attributes given its ID.

```
MESSHEADER
"\Subject UpdateEndpoint"
"\EndpointId " <EndpointId>
"\DataRefId " <DataRefId>
"\Direction " "Source" | "Destination" | "Bidirectional"
"\Service " <Service>
"\PSpecId " <PSpecId>
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "DeleteEndpoint"

Deletes an endpoint given its ID.

```
MESSHEADER
"\Subject DeleteEndpoint"
"\EndpointId " <EndpointId>
```

Message “GetEndpointList”

The application program, while running, is expected to maintain the list of endpoints. It may obtain this list, for a given node, by sending the following message.

```
MESSHEADER
"\Subject GetEndpointList"
"\NodeId " <NodeId>
```

The link server will reply by sending a “EndpointListDef” message.

Message “ExecuteEndpoint”

When an endpoint is activated the following message will be sent to the link server to request that whatever action is associated with that endpoint (e.g. follow link) will be performed by the link server.

```
MESSHEADER
"\Subject ExecuteEndpoint"
"\EndpointId " <EndpointId>
```

Message “EndpointDef”

This message is sent by the link server on request from an application program (CreateEndpoint or GetEndpoint), or possibly the link server might send this autonomously in the case where a new endpoint has been made by a user in a different session. The tag “\Attributes” is a list of attribute name and value pairs that contain all the attributes the link server knows about this object.

```
MESSHEADER
"\Subject EndpointDef"
"\EndpointId " <EndpointId>
"\DataRefId " <DataRefId>
"\Direction " "Source" | "Destination" | "Bidirectional"
"\Service " <Service>
"\PSpecId " <PSpecId>
"\Attributes "
{"\Name "<the attribute’s name>
 "\Value "<its value>}*
"\EndAttributes "
```

Message “EndpointListDef”

Sent by the link server on request from an application program (endpoint).

```
MESSHEADER
"\Subject EndpointListDef"
{"\Endpoints "
 "\EndpointId " <EndpointId>
 "\DataRefId " <DataRefId>
 "\Direction " "Source" | "Destination" | "Bidirectional"
 "\Service " <ServiceName>
 "\PSpecId " <PSpecId>
 "\Attributes "
 {"\Name "<the attribute’s name>
  "\Value "<its value>}*
 "\EndAttributes "
"\EndEndpoints "}*
```

Message “DisplayEndpoint”

The link server might request the application program to display an endpoint (or “hotspot”) for example, as the end of a link that has been followed.

```
MESSHEADER
"\Subject DisplayEndpoint"
"\EndpointId " <EndpointId>
"\PSpecId " <PSpecId>
```

A.4.2 Datarefs

Message “CreateDataRefs”

Datarefs encapsulate attributes for location specifications and also the node itself. Some systems will require datarefs to be created automatically when an endpoint is created; others with more complex user interfaces might allow users to create datarefs and connect them to endpoints. Note that not only users but also system processes (including scripts) might create datarefs.

```
MESSHEADER
"\Subject CreateDataRef"
"\NodeId " <NodeId>
```

```
"\LocSpecVID " <LocSpecVersionID>
"\LocSpec "
LOCSPEC
"\EndLocSpec "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "GetDataRef"

This message allows the application program to ask the link server for the current details of the datarefs identified.

```
MESSHEADER
"\Subject GetDataRef"
"\DataRefId " <DataRefId>
```

Both messages, CreateDataRef and GetDataRef should be answered by the link server with a DataRefDef message.

Message "UpdateDataRef"

Updates a dataref's attributes given its ID.

```
MESSHEADER
"\Subject UpdateDataRef"
"\DataRefId " <DataRefId>
"\NodeId " <NodeId>
"\LocSpecVID " <LocSpecVersionID>
"\LocSpec "
LOCSPEC
"\EndLocSpec "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "DeleteDataRef"

Deletes a dataref given its ID.

```
MESSHEADER
"\Subject DeleteDataRef"
"\DataRefId " <DataRefId>
```

Message “DataRefDef”

This message is sent by the link server on request from an application program (Create-DataRef or GetDataRef). In the case where a new endpoint has been made by a user in a different session, the link server might send this autonomously. The tag “\Attributes” is a list of attribute name and value pairs that contain all the attributes the link server knows about this object.

```
MESSHEADER
"\Subject DataRefDef"
"\DataRefId " <DataRefId>
"\NodeId " <NodeId>
"\LocSpecVID " <LocSpecVersionID>
"\LocSpec "
LOCSPEC
"\EndLocSpec "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

A.4.3 Links

The next set of messages is concerned with the manipulation of links. In practice most application programs used for viewing data will not handle links, but only endpoints. This set of messages is provided partly in order to provide a symmetric set of messages handling all objects within the link server. However, we can envisage the case where a developer may wish to produce client side tools for the creation, manipulation and editing of links. From the point of view of the link server, it is unimportant whether the messages come from an application program or a link editor and OHP should provide such messages.

Message “CreateLink”

```
MESSHEADER
"\Subject CreateLink"
```

```
"\Endpoints " {"\EndpointId " <EndpointId>}+ "\EndEndpoints "
"\Description " <Description>
"\Type " <Type>
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "GetLink"

Gets the attributes of a link given its ID. Both messages, CreateLink and GetLink, should be answered by the link server with a LinkDef message.

```
MESSHEADER
"\Subject GetLink"
"\LinkId " <LinkId>
```

Message "UpdateLink"

Updates a link's attributes given its ID.

```
MESSHEADER
"\Subject UpdateLink"
"\LinkId " <LinkId>
"\Endpoints " {"\EndpointId " <EndpointId>}+ "\EndEndpoints "
"\Description " <Description>
"\Type " <Type>
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "DeleteLink"

Deletes a link given its ID.

```
MESSHEADER
"\Subject DeleteLink"
"\LinkId " <LinkId>
```

Message “GetLinkList”

An application program may be interested in manipulating the list of all links. Note that future versions of OHP may include a context tag or a query tag in order to allow the client to specify which list of links to get. At present it will deliver all links in the link server.

```
MESSHEADER
"\Subject GetLinkList"
```

The link server will reply by sending a ”LinkListDef” message.

Message “LinkDef”

Sent by the link server on request from an application program (CreateLink and GetLink).

```
MESSHEADER
"\Subject LinkDef"
"\LinkId " <LinkId>
"\Endpoints " {"\EndpointId " <EndpointId>}+ "\EndEndpoints "
"\Description " <Description>
"\Type " <Type>
"\Attributes "
{"\Name " <the attribute’s name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message “LinkListDef”

Sent by the link server on request from an application program (GetLinkList).

```
MESSHEADER
"\Subject LinkListDef"
{"\Links "
 "\LinkId " <LinkId>
 "\Endpoints " {"\EndpointId " <EndpointId>}+ "\EndEndpoints "
 "\Description " <Description>
 "\Type " <Type>
 "\Attributes "
 {"\Name " <the attribute’s name>
  "\Value " <its value>}*
```

```
 "\EndAttributes "
"\EndLinks "}*
```

A.4.4 Nodes

Again, as with links, we do not believe that the typical application program will be interested in manipulating nodes, but since nodes are handled by many link servers, it seems appropriate to provide an interface to these objects so that developers may produce client side tools to manipulate them, such as browsers. The reader is reminded that in this context a node is a wrapper object that contains the meta-data about a document, rather than the document itself.

OHP is not a protocol for dealing with the actual storage and retrieval of documents. We assume that at some stage a protocol will be created for doing this (HTTP is actually one such protocol, if limited in some aspects; the Open Document Management API proposes another (see Open Document Management API, 1997). Yet actually managing the content of documents is what will make OHP a useable protocol. For the moment we therefore abstract document location in the definition of the protocol by using an opaque string representing a document (”CONTENTSPEC”).

Message “CreateNode”

The following message will create a new node in the link server. This message is perhaps the one message related to nodes that might be used by an application program, in the case where a client side program had just loaded or created some new data content and wished to register it with the link server.

```
MESSHEADER
"\Subject CreateNode"
"\NodeName " <A title that the application may choose to display>
"\MimeType " <MimeType>
"\PreferredApp " <The name of the application which we would prefer
                  to use with this data>
"\ContentSpecVID " <ContentSpecVersionID>
"\ContentSpec "
CONTENTSPEC
"\EndContentSpec "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
```

```
"\EndAttributes "
```

Message “GetNode”

Gets a node’s attributes given its ID.

```
MESSHEADER
"\Subject GetNode"
"\NodeId " <NodeId>
```

Both messages, CreateNode and GetNode should be answered by the link server with a NodeDef message.

Message “UpdateNode”

Updates a node’s attributes given its ID.

```
MESSHEADER
"\Subject UpdateNode"
"\NodeId " <NodeId>
"\NodeName " <A title that the application may choose to display>
"\MimeType " <MimeType>
"\PreferredApp " <The name of the application which we would prefer
                  to use with this data>
"\ContentSpecVID " <ContentSpecVersionID>
"\ContentSpec "
CONTENTSPEC
"\EndContentSpec "
"\Attributes "
{"\Name " <the attribute’s name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message “DeleteNode”

Deletes a node given its ID.

```
MESSHEADER
"\Subject DeleteNode"
"\NodeId " <NodeId>
Message "GetNodeList"
```

Message “GetNodeList”

An application program may be interested in manipulating the list of all nodes. Note that future versions of OHP might include a context tag or a query tag in order to allow the client to specify which list of nodes to get. At present it will deliver all nodes in the link server.

```
MESSHEADER
"\Subject GetNodeList"
```

The link server will reply by sending a NodeListDef message.

Message “NodeDef”

Sent by the link server on request from an application program.

```
MESSHEADER
"\Subject NodeDef"
"\NodeId " <NodeId>
"\NodeName " <A title that the application may choose to display>
"\MimeType " <MimeType>
"\PreferredApp " <The name of the application which we would prefer
                  to use with this data>
"\ContentSpecVID " <ContentSpecVersionID>
"\ContentSpec "
CONTENTSPEC
"\EndContentSpec "
"\Attributes "
{"\Name " <the attribute’s name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message “NodeListDef”

Sent by the link server on request from an application program (GetNodeList).

```
MESSHEADER
"\Subject NodeListDef"
{"\Nodes "
 "\NodeId " <NodeId>
 "\NodeName " <A title that the application may choose to display>
```

```
 "\MimeType " <MimeType>
 "\PreferredApp " <name of the application which we would prefer
                   to use with this data>
 "\ContentSpecVID " <ContentSpecVersionID>
 "\ContentSpec "
 CONTENTSPEC
 "\EndContentSpec "
 "\Attributes "
 {"\Name " <the attribute's name>
  "\Value " <its value>}*
 "\EndAttributes "
"\EndNodes "}*
```

A.4.5 Scripts

Scripts play an important part in some hypermedia systems (e.g. Hypercard and Multicard), and are hardly used in others. Scripts may be classified into two types:

- Server End Scripts. These are the scripts that are carried out when some particular event occurs or some action is requested. From the viewer's point of view there is no difference between some process being run or a link being followed by the link server. Of course these scripts may cause new messages to be sent back to the application program, to change its presentation in some way.
- Client End Scripts. These are scripts that are sent to the application program by the link server. In general they are either sent as part of a LocSpec in order to identify an endpoint or they are sent as a process which the application program will be expected to run. For example in order to change the presentation of the data in some way. OHP must provide support for such scripts if the application program wishes to use them.

From the point of OHP, Scripts themselves are opaque. However it is useful to attempt some standardisation of the content of the Script. This is discussed in Appendix D.

Message "CreateScript"

This message is used to create a new script.

```
MESSHEADER
"\Subject CreateScript"
"\ScriptVID " <ScriptVersionID>
```

```
"\Script "
SCRIPTSPEC
"\EndScript "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "GetScript"

Gets the attributes of a script given its ID. Both messages, CreateScript and GetScript, should be answered by the link server with a ScriptDef message.

```
MESSHEADER
"\Subject GetScript"
"\ScriptId " <ScriptId>
```

Message "UpdateScript"

Updates a script's attributes given its ID.

```
MESSHEADER
"\Subject UpdateScript"
"\ScriptId " <ScriptId>
"\ScriptVID " <ScriptVersionID>
"\Script "
SCRIPTSPEC
"\EndScript "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "DeleteScript"

Deletes a script given its ID.

```
MESSHEADER
"\Subject DeleteScript"
"\ScriptId " <ScriptId>
```

Message "ExecuteScript"

A client may send this message to the server, in which case the server will execute the given script, or a link server may send this message to a client, in which case if the client does not currently have the script identified it will need to send a GetScript message back in order to execute the script on the client.

```
MESSHEADER
"\Subject ExecuteScript"
"\ScriptId " <ScriptId>
```

Message "ScriptDef"

Sent by the link server on request from an application program.

```
MESSHEADER
"\Subject ScriptDef"
"\ScriptId " <ScriptId>
"\ScriptVID " <ScriptVersionID>
"\Script "
SCRIPTSPEC
"\EndScript "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

A.4.6 Presentation Specifiers

Alternative presentation of hotspots is an important feature in some systems. OHP allows users to specify different presentations. From the point of view of OHP and link servers, presentation specifiers (PSpecs) are opaque strings, which are interpreted by the application program. However, in order to allow for interoperability of application programs, and in order to allow server end scripts to change the presentation in a client it is useful to attempt some standardisation of the content of the PSpec. This is discussed in Appendix B Presentation Specification.

Message "CreatePSpec"

Used to create a presentation specification.

```
MESSHEADER
"\Subject CreatePSpec"
"\PSpecVID " <PSpecVersionID>
"\PSpec "
PSPEC
"\EndPSpec "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "GetPSpec"

Returns a presentation specification's attributes given its ID. Both messages, CreatePSpec and GetPSpec, should be answered by the link server with a PSpecDef message. This ID then can be used as parameter in further message calls.

```
MESSHEADER
"\Subject GetPSpec"
"\PSpecId " <PSpecId>
```

Message "UpdatePSpec"

Updates a presentation specification's attributes given its ID.

```
MESSHEADER
"\Subject UpdatePSpec"
"\PSpecId " <PSpecId>
"\PSpecVID " <PSpecVersionID>
"\PSpec "
PSPEC
"\EndPSpec "
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "DeletePSpec"

Deletes a presentation specification given its ID.

```
MESSHEADER
"\Subject DeletePSpec"
"\PSpecId " <PSpecId>
```

Message ”PSpecDef”

Sent by the link server on request from an application program.

```
MESSHEADER
"\Subject PSpecDef"
"\PSpecId " <PSpecId>
"\PSpecVID " <PSpecVersionID>
"\PSpec "
PSPEC
"\EndPSpec "
"\Attributes "
{"\Name " <the attribute’s name>
 "\Value " <its value>}*
"\EndAttributes "
```

A.4.7 Services

Message “GetServices”

The ”GetServices” message is to be sent from application program to link server in order to know which additional services are available, above the standard set which manipulate dataRefs, endpoints, links, nodes and presentations. The link server should answer this request with a ServicesDef message. Services are defined by a description and a service name.

```
MESSHEADER
"\Subject GetServices"
```

Message “ExecuteService”

The ExecuteService message is a very general message that can be used by the application program to request a link server to do one of its services. The available services can be retrieved by the GetServices message.

```
MESSHEADER
"\Subject ExecuteService"
```

```
"\Service " <name of the service>
"\EndpointId " <EndpointId>
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

Message "ServicesDef"

Sent by the link server on request from an application program (GetServices).

```
MESSHEADER
"\Subject ServicesDef"
{"\Services "
 "\Service " <name of the service>
 "\Description " <a string which may be used to describe the service
                  to a user>
"\EndServices " }*
```

The minimum reply should include the service named "FollowLink", which all link servers must handle. All other services are server dependant.

Message "ClosingNode"

This message is sent from the application program to the link server when a node is closed by the user.

```
MESSHEADER
"\Subject ClosingNode"
"\NodeId "<NodeId>
```

Message "DisplayNodeContent"

The link server might send this message for instance as the result of a follow link from another application.

```
MESSHEADER
"\Subject DisplayNodeContent"
"\NodeId "<NodeId>
"\ReadOnly" {"True" | "False"}
```

Message “DisplayDataRef”

Equally, location specifications might want to be displayed by the link server. This is done by sending the DisplayDataRef message; note that datarefs have the attributes NodeId and location specification.

```
MESSHEADER
"\Subject DisplayDataRef"
"\DataRef " <DataRefId>
"\PSpecId " <PSpecId>
```

Message “CloseNode”

This message is sent by the link server to a node that it wishes to close, e.g. in order to free a lock on a document.

```
MESSHEADER
"\Subject CloseNode"
"\NodeId "<NodeId>
"\UpdateNode "{"True" | "False’’}
```

The application is then responsible for closing the node itself, ensuring that it has updated its contents if required by the UpdateNode tag. In case of any errors the application program should send back an error message.

Message “Error”

The Error message might be sent by any component. It is composed of the following structure:

```
MESSHEADER
"\Subject Error"
"\MessageId " <Id of message that caused the error>
"\ErrorSubject " <the type of error that occurred>
"\ErrorMessage " <the actual error string>
```

A.5 Opaques

Although certain parts of the protocol were left opaque for practical interoperability an on-the-wire definition is required and proposals for location specifications, presentation specifications, content identifier specifications and script specifications were created.

A.5.1 Location Specifications (LOCSPEC)

The LOCSPEC defines a position within a document, to which a link can anchor itself.

```
LOCSPEC =
"\ContentType " <type, e.g., ASCII, binary>
"\Content "     <Mime encoded text string>
"\LocVID "      <LocVersionID>
"\Loc "
    LOC <a LOC object>
"\EndLoc "
```

Location specifications consist of a content type, the content itself as well as the actual location data. LOC is defined as

```
Loc = { NAMELOC | DATALOC | TREELOC | PATHLOC | SCRIPTLOC | NALOC }
```

Name Space Locations Name space locations are used to reference an object by its name. The definition of name locations is as follows:

```
NAMELOC =
"\LocType NameLoc"
```

Data Locations Data locations are co-ordinate locations, they allow a user to define a location as a position of a defined scale.

```
DATALOC =
"\LocType        DataLoc"
"\Quantum "      { "string" | "int" | "byte" | "utc" }
"\CountList "    <list of Strings being interpreted by the
                  application program>
"\RevCountList " <list of Strings being interpreted by the
                  application program>
"\Overrun "      { "ignore" | "error" | "trunc" }
```

Tree Locations Tree locations can be used for addressing a single object within a tree. The addressing is done in a way that on each level of the tree an object is selected by its position.

```
TREELOC =
```

```
"\LocType        TreeLoc"
"\Overrun "      { "ignore" | "error" | "trunc" }
"\CountList "    <list of numbers>
```

Path Locations As opposed to tree locations path locations are used to address a range of nodes by a path. This is done by addressing a tree as though it were a matrix in which the rows are the levels of the tree and the columns are the paths. Therefore the count list of a path location consists of an even number of pairs: the first pair identifies the columns (i.e. the paths to the leaves) and the second pair selects the rows.

```
PATHLOC =
"\LocType       PathLoc"
"\NodeId "      <NodeId>
"\Overrun "     { "ignore" | "error" | "trunc" }
"\CountList "   <list of numbers>
```

Script Locations OHP does not define a specific query mechanism for addressing locations. However, by allowing script locations we define a way for those hypertext systems that use scripts to identify and address locations at the client's side.

```
SCRIPTLOC =
"\LocType       ScriptLoc"
"\ScriptId      <ScriptId>
```

Inaccessible Locations An idea borrowed from HyTime is to address data that is currently not accessible. We call this location specification NALoc for "Not Accessible" location.

```
NALOC =
"\LocType NALoc"
"\Location " <string as description of the object and its physical
             location>
```

A.5.2 Presentation Specifications (PSPEC)

PSPECS may be used to store, and thus re-apply, a required presentation for any hypertext object that a client may wish to display. A client might store a PSPEC as a byte stream which would then later be retrieved and interpreted by the same client.

```
PSPEC =
```

```
"\Name "        <a string>
["\Colour "     <colour>]
["\Style "      <style>]
["\Visibility " "true" | "false"]
```

Following the original proposal we suggest a triplet of colour, style and visibility, each of which is optional. The above definition allows us to only put these things if we have them, and we could also add other tags if needed. We will define some tag values which will always be understood.

We assume that the eight primary computer colours are supported, i.e. black, blue, cyan, white, magenta, green, red and yellow.

As minimal subset of styles we define flashing, bold, italic, outlined and shaded.

This syntax may be extended by the addition of further name and value pairs. Much further work needs to be done on discovering the sort of attributes which people wish to store.

A.5.3 Content Specifications (CONTENTSPEC)

We recommend that the following definition for CONTENTSPEC be used, until such time as further work has improved on this.

```
CONTENTSPEC =
"\Name "      <the content's name, typically a file name, a URL or
               a DMS handle>
"\Location "  "FileSystem" | "Internet" | "DMS"
"\Attributes "
{"\Name "     <the attribute's name>
 "\Value "    <its value>}*
"\EndAttributes "
```

As far as the link server is concerned the CONTENTSPEC information is opaque. It stores the string, and when the node is required it sends this back. The client side is expected to provide some component which will be able to interpret this string in the cases where the client is asked to display a document. Typically this will be the CSF, which will then get the document from the file system, Internet or the Document Management System (DMS) that it is using (which may actually be the link server itself if the link server is a hyperbase).

A.5.4 Script Specifications (SCRIPTSPEC)

We recommend that the following definition for SCRIPTSPEC be used.

```
SCRIPTSPEC =
"\Language " <name of the script language, e.g. JavaScript, etc.>
"\Data " <the actual script>
"\Attributes "
{"\Name " <the attribute's name>
 "\Value " <its value>}*
"\EndAttributes "
```

This is a simple definition, but will allow scripting to be added to an application with little extra complication.

Appendix B

OHP-Nav Definition - eXtended Markup Language (XML)

B.1 The Definition Language

The eXtended Markup Language (XML) is a simple meta-language for creating elements, attributes and values similar to HTML. XML documents that conform to a certain layout and shape are said to be 'well-formed'. The description of a standard and shape are written in an accompanying Document Type Definition (DTD).

The first DTD developed for OHP-Nav is presented below. It is a simple element tree that uses opaque 'PCDATA' for all the values, no attributes are used. This DTD was created after the EBNF definition described in Appendix A and is thus slightly more advanced. In particular it includes operations for dealing with lists of all the basic objects as well as the objects themselves. However as it was written for the practical demonstration of OHP-Nav at HT'98 it does not properly define those parts of the protocol that were not used at that time, this includes the CONTENTSPEC and all the types of LOCSPEC other that DataLoc, which have been simplified.

B.2 The Document Type Definition

```
<!--

  OHP Navigational Document Type Definition for defining OHPNav messages.
  This file has been last modified 10 June 1998.
-->
```

```
<!ENTITY ohpnavversion  "OHPNAV 1.0.Soton-10-Jun-1998">

<!ENTITY \% allMessNames
 "createendpoint | getendpoint     | updateendpoint
| deleteendpoint | getendpointlist | executeendpoint
| endpointdef   | endpointlistdef | displayendpoint
| createdataref  | getdataref     | updatedataref
| deletedataref  | datarefdef      | createlink
| getlink        | updatelink      | deletelink
| getlinklist    | linkdef         | linklistdef
| createnode     | getnode     | updatenode
| deletenode     | getnodelist     | nodedef
| nodelistdef    | getservices     | executeservice
| servicedef     | servicelistdef  | closingnode
| displaynode | displaydataref | closenode
| error">

<!ELEMENT OHPNav  (messageheader, (\%allMessNames;))>

<!-- MESSAGEHEADER and IDs-->

<!ELEMENT messageheader  (mid, rid, vid, sid, uid, fid)>

<!ELEMENT mid   (#PCDATA)>
<!ELEMENT rid   (#PCDATA)>
<!ELEMENT vid   (#PCDATA)>
<!ELEMENT sid   (#PCDATA)>
<!ELEMENT uid   (#PCDATA)>
<!ELEMENT fid   (#PCDATA)>

<!ELEMENT nodeid    (#PCDATA)>
<!ELEMENT readonly   (#PCDATA)>
<!ELEMENT nodename   (#PCDATA)>
<!ELEMENT mimetype   (#PCDATA)>
<!ELEMENT preferredapp   (#PCDATA)>
<!ELEMENT contentspecvid (#PCDATA)>
<!ELEMENT name   (#PCDATA)>
<!ELEMENT location  (#PCDATA)>
<!ELEMENT value  (#PCDATA)>
<!ELEMENT datarefid   (#PCDATA)>
<!ELEMENT direction   (#PCDATA)>
```

```
<!ELEMENT endpointid  (#PCDATA)>
<!ELEMENT contenttype   (#PCDATA)>
<!ELEMENT content   (#PCDATA)>
<!ELEMENT locvid  (#PCDATA)>
<!ELEMENT quanta   (#PCDATA)>
<!ELEMENT countlist   (#PCDATA)>
<!ELEMENT revquanta   (#PCDATA)>
<!ELEMENT revcountlist   (#PCDATA)>
<!ELEMENT overrun  (#PCDATA)>
<!ELEMENT description   (#PCDATA)>
<!ELEMENT type    (#PCDATA)>
<!ELEMENT url   (#PCDATA)>
<!ELEMENT serviceid  (#PCDATA)>
<!ELEMENT messageid   (#PCDATA)>
<!ELEMENT errorsubject   (#PCDATA)>
<!ELEMENT errormessage   (#PCDATA)>
<!ELEMENT update  (#PCDATA)>
<!ELEMENT linkid   (#PCDATA)>
<!ELEMENT version   (#PCDATA)>

<!-- ATTRIBUTE -->
<!ELEMENT attribute   (name, value)>


<!-- ATTRIBUTES -->
<!ELEMENT attributes  (attribute*)>

<!-- CONTENTSPEC -->
<!ELEMENT contentspec   (version, url, attributes)>

<!-- LOCSPECS -->
<!ELEMENT nameloc   (version, contenttype, content, spec)>
<!ELEMENT treeloc   (version, contenttype, content, spec)>
<!ELEMENT naloc   (version, contenttype, content, spec)>
<!ELEMENT pathloc   (version, contenttype, content, spec)>
<!ELEMENT scriptloc   (version, contenttype, content, spec)>
<!ELEMENT dataloc   (version, contenttype, content, quantalist,
                        countlist, revquantalist, revcountlist,
                        overrun)>

<!ELEMENT spec           (#PCDATA)>
```

```
<!-- ENDPOINT -->
<!ELEMENT endpoint   (endpointid, dataref, direction, service,
                          pspec, attributes)>
<!ELEMENT endpoints   (endpoint*)>

<!-- PSPEC -->
<!ELEMENT pspec  (version, colour, style, visibility)>
<!ELEMENT colour   (#PCDATA)>
<!ELEMENT style   (#PCDATA)>
<!ELEMENT visibility   (#PCDATA)>

<!-- DATAREF -->
<!ELEMENT dataref   (datarefid, node, (nameloc | dataloc |
                          treeloc | pathloc | scriptloc | naloc),
                          attributes)>

<!-- LINK -->
<!ELEMENT link     (linkid, endpoints, description, type,
                          attributes)>
<!ELEMENT links   (link*)>

<!-- NODE -->
<!ELEMENT node     (nodeid, nodename, mimetype, preferredapp,
                          contentspec, attributes)>
<!ELEMENT nodes   (node*)>

<!-- SERVICE -->
<!ELEMENT service   (serviceid, description)>
<!ELEMENT services   (service*)>

<!--
     the actual OHPNavigational messages
-->

<!-- CREATEENDPOINT -->
<!ELEMENT createendpoint  (endpoint)>

<!-- GETENDPOINT -->
<!ELEMENT getendpoint   (endpointid)>
```

```
<!-- UPDATEENDPOINT -->
<!ELEMENT updateendpoint  (endpoint)>

<!-- DELETEENDPOINT -->
<!ELEMENT deleteendpoint  (endpointid)>

<!-- GETENDPOINTLIST -->
<!ELEMENT getendpointlist  (nodeid)>

<!-- EXECUTEENDPOINT -->
<!ELEMENT executeendpoint  (endpointid)>

<!-- ENDPOINTDEF -->
<!ELEMENT endpointdef   (endpoint)>

<!-- ENDPOINTLISTDEF -->
<!ELEMENT endpointlistdef  (endpoints)>

<!-- DISPLAYENDPOINT -->
<!ELEMENT displayendpoint  (endpoint, pspec)>

<!-- CREATEDATAREF -->
<!ELEMENT createdataref  (dataref)>

<!-- GETDATAREF-->
<!ELEMENT getdataref   (datarefid)>

<!-- UPDATEDATAREF -->
<!ELEMENT updatedataref  (dataref)>

<!-- DELETEDATAREF -->
<!ELEMENT deletedataref  (datarefid)>

<!-- DATAREFDEF -->
<!ELEMENT datarefdef   (dataref)>

<!-- CREATELINK -->
<!ELEMENT createlink   (link)>

<!-- GETLINK -->
<!ELEMENT getlink   (linkid)>
```

```
<!-- UPDATELINK -->
<!ELEMENT updatelink   (link)>

<!-- DELETELINK -->
<!ELEMENT deletelink   (linkid)>

<!-- GETLINKLIST -->
<!ELEMENT getlinklist   EMPTY>

<!-- LINKDEF -->
<!ELEMENT linkdef   (link)>

<!-- LINKLISTDEF -->
<!ELEMENT linklistdef   (links)>

<!-- CREATENODE -->
<!ELEMENT createnode   (node)>

<!-- GETNODE -->
<!ELEMENT getnode   (nodeid)>

<!-- UPDATENODE -->
<!ELEMENT updatenode   (node)>

<!-- DELETENODE -->
<!ELEMENT deletenode   (nodeid)>

<!-- GETNODELIST -->
<!ELEMENT getnodelist   EMPTY>

<!-- NODEDEF -->
<!ELEMENT nodedef   (node)>

<!-- NODELISTDEF -->
<!ELEMENT nodelistdef   (nodes)>

<!-- GETSERVICES -->
<!ELEMENT getservices   EMPTY>

<!-- EXECUTESERVICE -->
```

```
<!ELEMENT executeservice (serviceid, endpointid, attributes)>

<!-- SERVICEDEF -->
<!ELEMENT servicedef   (service)>

<!-- SERVICELISTDEF -->
<!ELEMENT servicelistdef (services)>

<!-- CLOSINGNODE -->
<!ELEMENT closingnode   (nodeid)>

<!-- DISPLAYNODE -->
<!ELEMENT displaynode    (node, readonly)>

<!-- DISPLAYDATAREF -->
<!ELEMENT displaydataref (dataref, pspec)>

<!-- CLOSENODE -->
<!ELEMENT closenode   (node, update)>

<!-- ERROR -->
<!ELEMENT error   (messageheader, errorsubject, errormessage)>
```

Appendix C

OHP-Nav Definition - Interface Definition Language (IDL)

C.1 The Definition Language

The Interface Definition Language (IDL) is a implementation independent specification language. It specifies structures and operations that can then be converted into any particular implementation.

The IDL below shows the first version of OHP-Nav. Data types start with a capital letter, e.g. EndPoint, and method names start with a small letter, e.g. getEndPoint

We followed the XML definition/implementation and use IDs for client to server messages and object references for server to client messages. The idea is that the client only passes the ID to server (and the server 'knows' about the object); the server in turn passes whole objects to the client to avoid further messages such as getObjectForID.

C.2 The IDL Definition

```
/*OHPNav.idl - an interface definition for the open hypermedia
  navigational interface
  last change June 10 1998
*/

module ohp {
  interface OHPNavigational {
```

```
  //==================== Exceptions ========================//
  exception  OHPElementNotFoundException {string reason; };
  const string ohpnavversion = "OHPNAV 1.0.Soton-10-Jun-1998";

  struct MessageHeader {
string MID;
string RID;
string VID;
string SID;
string UID;
string FID;
  };
  enum Direction {source, destination, both};
  struct LocSpec {
  string contentType;
string mimeEncodedContent;
string version;
string theSpec;
  };
  typedef LocSpec DataLoc;
  typedef LocSpec NameLoc;
  typedef LocSpec TreeLoc;
  typedef LocSpec PathLoc;
  typedef LocSpec NALoc;
  typedef LocSpec ScriptLoc;

  struct Attribute {
        string name;
        string value;
  };

  typedef sequence<Attribute> Attributes;

  struct ContentSpec {
string version;
string url;
Attributes aList;
  };
```

```
  struct PSpec {
string version;
string spec;
string color;
string style;
string visibility;
  };

  struct Node {
string ID;
string name;
string mimeType;
string preferredApp;
ContentSpec cSpec;
        Attributes aList;
  };

  struct DataRef {
string ID;
       Node node;
       LocSpec lSpec;
       Attributes aList;
  };

  struct Service {
string ID;
string description;
  };

  struct EndPoint {
       string ID;
       DataRef dataRef;
       Direction direction;
       Service service;
       PSPec pSpec;
       Attributes aList;
  };

  typedef sequence<EndPoint> EndPoints;
```

```
  struct Link {
string ID;
string description;
EndPoints eList;
string type;
Attributes aList;
  };

  typedef sequence<Link> Links;
  typedef sequence<Node> Nodes;
  typedef sequence<Service> Services;

  //=================== The Messages ========================//

  //createEndPoint message
  EndPoint createEndPoint(in string dataRefID, in Direction ld,
in string serviceID, in PSpec p, in Attributes aList);

  //getEndPoint
  EndPoint getEndPoint(in string endPointID)
                raises (OHPElementNotFoundException);

  //updateEndPoint
  EndPoint updateEndPoint(in string endPointID, in string
        dataRefID, in Direction ld, in string serviceID,
        in PSpec p, in Attributes aList)
                raises (OHPElementNotFoundException);

  //deleteEndPoint
  void deleteEndPoint(in string endPointID)
                raises (OHPElementNotFoundException);

  //getEndPoints
  EndPoints getEndPointList(in string nodeID);

  //executeEndPoint
  void executeEndPoint(in string endPointID)
                raises (OHPElementNotFoundException);
```

```
//endPointDef
EndPoint endPointDef();

//endPointListDef
EndPoints endPointListDef();

//displayEndPoint
void displayEndPoint(in EndPoint ep, in PSpec p);

//createDataRef
DataRef createDataRef (in string nodeID, in LocSpec l,
                       in Attributes aList);

//getDataRef
DataRef getDataRef(in string dataRefID)
               raises (OHPElementNotFoundException);

//updateDataRef
DataRef updateDataRef(in string dataRefID, in string nodeID,
                      in LocSpec l, in Attributes aList)
               raises (OHPElementNotFoundException);

//deleteDataRef
void deleteDataRef(in string dataRefID)
               raises (OHPElementNotFoundException);

//dataRefDef
DataRef dataRefDef();

//createLink
Link createLink(in EndPoints el, in string description,
                in string type, in Attributes aList);

//getLink
Link getLink(in string linkID)
               raises (OHPElementNotFoundException);

//updateLink
```

```
  Link updateLink(in string linkID, in EndPoints el,
                  in string description, in string type,
                  in Attributes aList)
raises (OHPElementNotFoundException);

  //deleteLink
  void deleteLink(in string linkID)
                raises (OHPElementNotFoundException);

  //getLinkList
  Links getLinkList();

  //linkDef
  Link linkDef();

  //linkListDef
  Links linkListDef();

  //createNode
  Node createNode(in string name, in string mimeType,
          in string preferredApp, in ContentSpec c,
          in Attributes aList);

  //getNode
  Node getNode(in string nodeID)
                raises (OHPElementNotFoundException);

  //updateNode
  Node updateNode(in string nodeID, in string mimeType,
in string preferredApp, in ContentSpec c,
in Attributes aList)
         raises (OHPElementNotFoundException);

  //deleteNode
  void deleteNode(in string nodeID)
                raises (OHPElementNotFoundException);

  //getNodeList
  Nodes getnodelist();
```

```
  //nodeDef
  Node nodeDef();

  //nodeListDef
  Nodes nodeListDef();

  //getServices
  Services getServices();

  //executeService
  void executeService(in string serviceID, in string endPointID,
                      in Attributes aList);

  //serviceDef
  //obsolete??
  Service serviceDef();

  //serviceListDef
  Services serviceListDef();

  //closingNode
  void closingNode(in string nodeID);

  //displayNode
  void displayNode(in Node n, in boolean readOnly);

  //displayDataRef
  void displayDataRef(in DataRef dr, in PSpec p);

  //closeNode
  void closeNode(in Node n, in boolean update);

  //error
  void error(in MessageHeader mh, in string errorSubject,
in string errorMessage);
  };
}; //module
```

Appendix D

OHP-Service Definition (XML)

D.1 OHP Service Document Type Definition

The DTD below defines the on-the-wire specification for OHP-Service as it was used for the demo at Hypertext ’99 in Darmstadt, Germany. It is a separate proposal to the OHP-Nav protocol. This latest DTD refers to Service objects as ‘Computations’ and includes the definition of Composite Services.

D.2 The Document Type Definition

```
<!--  OHP Service Document Type Definition for defining OHPService
                        messages.
Last change: Jan 28 1999, 17:35 GMT-->

<!ENTITY % OHPSERVICEVERSION "OHPSERVICE-1.0.Darmstadt-1999">

<!ENTITY % allMessNames   "RETRIEVESERVICES  | SERVICESRETRIEVED |
  RETRIEVECSERVICES  | CSERVICESRETRIEVED |
EXECUTESERVICE | SERVICEEXECUTED |
SERVICEPROGRESS ">;

<!ENTITY % abObjInfo  "MYTYPE, DESCRIPTIONSET?, CHARACTERISTICSET?">
<!ENTITY % hmObjInfo  "(%abObjInfo;), COMPID?, PSPECSET?">

<!--  MESSAGESET
This is the basic structure of an OHPService message to be
sent over the wire as text (with 19 leading bytes expressing
```

```
the length of the message). An OHPService document consists
of one message plus any number of additional messages. -->

<!ELEMENT OHP          (MESSAGESET)>
<!ELEMENT MESSAGESET  (MESSAGE, MESSAGE*)>

<!--  MESSAGE STRUCTURE
An OHP message consists of a message header  plus a message
body which is one of the messages. -->

<!ELEMENT MESSAGE       (MESSAGEHEADER, (%allMessNames;))>
<!--USER DETAILS
Currently we only support a userid and an optional name.
Further extensions might be needed for collaboration. -->

<!ELEMENT ACCOUNT (USERID, NAME?)>
<!ELEMENT USERID (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!--  MESSAGEHEADER
The message header contains the following fields.
- SENDER (optional): the sending component
- RECEIVER (optional): the receiving component. Both fields
  could be (are) used to allow routing of messages.
- SERIAL: the unique message ID
- RETURNSERIAL (optional: used if a message is referring to
  a previous serial.
  - SESSION (optional): a session identifier
- ACCOUNT (optional): user data
- MNAME: the name of the message. This field allows e.g.
  routing of messages by looking at the header only (the
  content could be encrypted).
- PROTOCOL: this is version information about the protocol
  spoken.
- CERT (optional): an optional certificate that can be used
  for security reasons.
- CONTEXTIDSET: the set of contexts that this message
  applies to.
- PERFORMATIVE (optional): a performative as known from agent
  communication languages that allows dealing with messages
```

```
  without actually understanding their content. -->

<!ELEMENT MESSAGEHEADER  (SENDER?, RECEIVER?, SERIAL,
                          RETURNSERIAL?, SESSION?, ACCOUNT?, MNAME,
                          PROTOCOL, CERT?, CONTEXTIDSET,
                          PERFORMATIVE?)>

<!ELEMENT SENDER (#PCDATA)>
<!ELEMENT RECEIVER (#PCDATA)>
<!ELEMENT SERIAL (#PCDATA)>
<!ELEMENT RETURNSERIAL (#PCDATA)>
<!ELEMENT SESSION (#PCDATA)>
<!ELEMENT MNAME (#PCDATA)>
<!ELEMENT PROTOCOL (#PCDATA)>
<!ELEMENT CERT (#PCDATA)>
<!ELEMENT CONTEXTIDSET (CONTEXTID, CONTEXTID*)>
<!ELEMENT CONTEXTID (#PCDATA)>
<!ELEMENT PERFORMATIVE (#PCDATA)>

<!-- Operations -->

<!ELEMENT RETRIEVESERVICES  EMPTY>
<!ELEMENT SERVICESRETRIEVED (COMPUTATIONSET) >
<!ELEMENT EXECUTESERVICE  (SERVICEID, INPARAMSET?)>
<!ELEMENT SERVICEEXECUTED  (SERVICEID, OUTPARAMSET?)>
<!ELEMENT SERVICEPROGRESS (SERVICEID, PROGRESS)>
<!ELEMENT RETRIEVECSERVICES  EMPTY>
<!ELEMENT CSERVICESRETRIEVED (CCSET) >

<-- progress is given as a percentage (e.g. 15.5 = 15.5% complete) -->
<!ELEMENT PROGRESS (#PCDATA)>

<-- Computation -->

<!ELEMENT COMPUTATIONSET (COMPUTATION, COMPUTATION*) >
<!ELEMENT COMPUTATION ((%hmObjInfo;), ID, SPECID, NAME?, FUNCTIONNAME?,
INTEMPLATESET?, OUTTEMPLATESET?, MIMETYPESET?, CODESPEC?,
ETC?)>
```

```
<!-- this is the menu entry or button name -->
<!ELEMENT FUNCTIONNAME (#PCDATA)>
<!-- Expected Time to Completion, can take the values Instant, Fast,
     Medium, Slow, Very Slow -->
<!ELEMENT ETC (#PCDATA)>
<!-- specid is the same for all services that do the same thing,
     only id is unique -->
<!ELEMENT SPECID (#PCDATA)>
<!ELEMENT CODESPEC (#PCDATA)>
<!ELEMENT MIMETYPESET (MIMETYPE, MIMETYPE) >
<!ELEMENT MIMETYPE (#PCDATA)>


<!ELEMENT INTEMPLATESET  (INTEMPLATE, INTEMPLATE*)>
<!ELEMENT OUTTEMPLATESET (OUTTEMPLATE, OUTTEMPLATE*)>

<!--    intemplates define what input parameters are valid.
        PossibleValSet is like an enumeration of possible
        values.  -->
<!ELEMENT INTEMPLATE   (((HRANGE, LRANGE) | POSSIBLEVALSET?),
                           DEFAULT, NAME, TYPE)>
<!ELEMENT POSSIBLEVALSET (VALUE, VALUE, VALUE*)>
<!ELEMENT HRANGE (#PCDATA)>
<!ELEMENT LRANGE (#PCDATA)>
<!ELEMENT DEFAULT (#PCDATA)>

<!ELEMENT OUTTEMPLATE (NAME, TYPE) >


<-- Parameters -->

<!ELEMENT INPARAMSET  (INPARAM, INPARAM*)>
<!ELEMENT INPARAM  (NAME, PARAMVALUE)>
<!ELEMENT OUTPARAMSET  (OUTPARAM, OUTPARAM*)>
<!ELEMENT OUTPARAM  (NAME, PARAMVALUE, RANK?)>

<-- a paramvalue is represented here by a string, but thats string
        could actually represent other XML structures -->
<!ELEMENT PARAMVALUE  (#PCDATA)>
```

```
<!ELEMENT RANK  (#PCDATA)>


<!--  COMPOSITE COMPUTATIONS (CC)
the input and output parameters are to be extracted from
the actual services as specified in the COMPUTATION's GRAPH,
i.e. a cc cannot be invoked! -->

<!ELEMENT CCSET (CC, CC*) >
<!ELEMENT CC  ((%hmObjInfo;), ID, COMPUTATIONGRAPH,
                        FUNCTIONNAME?, NAME, SPECID)>

<!ELEMENT COMPUTATIONGRAPH  ((PARALLEL | SPECID),
                             (PARALLEL | SPECID)*)>
<!ELEMENT PARALLEL  (SPECID, SPECID, SPECID*)>
```

Appendix E

Performatives

E.1 KQML Performatives

KQML aims to serve several needs of inter-agent communication. These can be summarised as:

- querying and information passing (e.g. evaluate, ask-if, tell)
- managing multiple responses to queries (e.g. ask-all, stream-all, standby, ready, next)
- managing capability definition and dissemination (e.g. advertise, recommend)
- managing communications (e.g. register, forward, broadcast)

The list of KQML Performatives in Alphabetical Ordering (referring the proposition passed as 'content'):

achieve
advertise tell my friends that I can process a message like the one in content
ask-if ask the other peer whether it supports a particular service?
ask-all return all results of this service in one message
ask-one return one result of this service in one message
broadcast forward the content to all components that you know of
broker-one asks a broker to resolve the message to one result via another component and return it in a forward
broker-all asks a broker to resolve the message to all results via another component and return it in a forward
delete-one
delete-all

deny
discard throw away all the remaining responses that are waiting
eos end of stream marker for stream-all
error last message was malformed, i.e. I couldn't parse it
forward send content onto the component named in the to: field insert
next requests the next response that is waiting (see standby)
ready the response is now ready
recommend-one returns a component that can (and will) resolve this message
recommend-all returns all components that can (and will) resolve this message
recruit-one asks a broker to resolve the message to one result via another component and return it directly
recruit-all asks a broker to resolve the message to all results via another component and return it directly
register here I am, my name is X rest requests all the responses that are waiting (followed by a eos)
sorry last message was OK but I don't know what to do with it
standby I will let you know when the response is ready
stream-all return all results of this service in a series of messages
subscribe inform me of all messages of type content
tell asserts a truth about the ls/hb
transport-address change of address
unadvertise tell others that I cannot process a message like the one in content
uninsert
unregister I'm off, my name was X
untell asserts a falsehood about the ls/hb
unachieve
undelete

E.2 FIPA's Agent Communication Language

FIPA's Agent Communication Language (ACL) distinguishes more clearly between agent management and speech acts. Therefore, some of the functionality is part of the agent management (e.g. register) and some is part of ACL.

FIPA Communicative Acts:

accept-proposal the action of accepting a previously submitted proposal to perform an action
agree the action of agreeing to perform some action

cancel canceling some previously requested action

cfp call for proposals to perform a given action

confirm sender confirms to receiver that a proposition is true

disconfirm sender informs receiver that a proposition is false

failure I tried it but it didn't work

inform sender tells receiver that a proposition is true (It's raining today)

inform-if (macro-act) tell me whether Paris is in France

inform-ref (macro-act) tell me the current Prime Minister of the UK

not-understood I'd like to do it but I don't understand it

propose submitting a proposal to perform a certain action

query-if asking the receiver whether a proposition is true

query-ref asking another agent for the object referred to by an expression. E.g. ask for available services

refuse the action of refusing to perform a given action, and explains why

reject-proposal the action of rejecting a proposal

request sender requests the receiver to perform some action, e.g. open a file

request-when sender requests the receiver to perform an action when event occurs

request-whenever sender requests the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.

subscribe request notifications whenever the referenced object changes

Appendix F

FOHM SoFAR Ontology

F.1 The Definition Language

The SoFAR agent framework (described in Section 8.2) communicates using sets of related *Predicates* (or truth statements), these are in term made up of *Literals*, atomic values such as numbers or strings, and *Terms* collections of literals and terms that are in turn used by predicates. These sets of predicates are known as ontologies and represent a subset of the statements you can make about the world.

Before writing agents it is necessary to create an ontology with which they will communicate. This is done by writing an XML file that is then parsed by the SoFAR ontology compiler. The XML defines a hierarchy of predicate objects and includes comments on each field that is converted to Javadoc in the final Java source files.

F.2 The FOHM XML Definition

Listed below is the latest version of the FOHM SoFAR ontology. This version is almost complete but does not yet include the context objects described in Section 7.4.3.

```
<!-- Last edited 29-11-00 -->

<ontology name="Fohm">
<package>sofar.users.dem97r.ontology.fohm</package>
<import>sofar.ontology.base.*</import>
<import>sofar.ontology.web.*</import>
<import>sofar.ontology.multimedia.*</import>
<import>sofar.ontology.actions.*</import>
```

```
<url>http://www.sofar.ecs.soton.ac.uk/</url>
<version>1.0</version>
<author>Dave Millard</author>

<comment>The Fundamental Open Hypertext Model (FOHM) is
an attempt to define a common model that captures all the
functionality of three different hypertext domains.
Navigational, Spatial and Taxonomic Hypertext. </comment>

<vector name="BindingVector"   type="Binding"/>
<vector name="StringVector"    type="String"/>

<term name="ReferencableObject" extends="Predicate" abstract="yes">
<comment>These are objects that may be Referenced by an
Association (via an ObjectRef) </comment>
</term>

<term name="Association" extends="ReferencableObject">
<comment>An Association is FOHM's way of expressing a
link. It represents a relationship between 0 or more
objects (other ReferencableObjects). It contains a
featurespace - a list of attributes for each
member of the Association must provide a value. For
example the feature space of a Navigational Link is a
single attribute "direction".</comment>
<field type="StorageID" name="id">
<comment>the id that locates this object
(could be UndefinedID)</comment>
</field>
<field type="String" name="relationshiptype">
<comment>the type (i.e. "Supports", "Explains")</comment>
</field>
<field type="String" name="description">
<comment>a description of this association
(i.e. "Published Papers")</comment>
</field>
<field type="String" name="structuretype">
<comment>the structure (i.e. "List", "DSet")</comment>
</field>
```

```
<field type="BindingVector" name="bindings">
<comment>list of (Binding)s in this association</comment>
</field>
<field type="StringVector" name="featurespace">
<comment>ordered list of (String)s defining
binding requirements</comment>
</field>
</term>

<term name="Data" extends="ReferencableObject">
<comment>In FOHM all forms of data have to be wrapped
by a Data object. This can be a file identified by a
unique ID or URL but also might be the content of the
Data itself. </comment>
<field type="StorageID" name="id">
<comment>the id that locates this object
(could be UndefinedID)</comment>
</field>
<field type="MediaObject" name="content">
<comment>The Media itself (can be content or
a reference)</comment>
</field>
</term>

<term name="UndefinedReferencableObject"
      extends="ReferencableObject">
<comment>This is used when stating that a Reference
points at 'any' ReferencableObject</comment>
</term>

<term name="Reference" abstract="yes" extends="Predicate"/>

<term name="ObjectRef" extends="Reference">
<comment>In FOHM the endpoint of a link is always
fixed on an object called a Reference.  An ObjectRef
refers to a ReferencableObject
 and may optionally point into that
object. (i.e. the third paragraph of a text document,
or a particular region of an image)</comment>
```

```
<field type="StorageID" name="id">
<comment>the id that locates this object
(could be UndefinedID)</comment>
</field>
<field type="ReferencableObject" name="target">
<comment>the (ReferencableObject) it is
referencing</comment>
</field>
<field type="LocSpec" name="locspec">
<comment>the LocSpec used (could be an
UndefinedLocSpec)</comment>
</field>
</term>

<term name="IDRef" extends="Reference">
<comment>This refers
to an object that is identified by ID. To retrieve the
object becomes the responsibility of the recipient.
(The ID cannot be guaranteed to reference a
ReferencableObject, but should do to be valid).
</comment>

<field type="StorageID" name="id">
<comment>the id that locates this object
(could be UndefinedID)</comment>
</field>
<field type="ReferencableStorageID" name="target">
<comment>the id of the target of this
Reference</comment>
</field>
<field type="LocSpec" name="locspec">
<comment>the LocSpec used (could be an
UndefinedLocSpec)</comment>
</field>
</term>

<term name="Binding" extends="Term">
<comment>A Binding is an object that binds a
particular Reference to a particular Association.  It
```

```
includes a feature vector that makes values to the
feature space of the Association (E.g. in a
navigational link it would map either "source",
"destination" or "bi-directional" to the feature
"direction").</comment>

<field type="StringVector" name="featurevalues">
<comment>ordered list of (String)s </comment>
</field>
<field type="Reference" name="reference">
<comment>the (Reference) bound</comment>
</field>
</term>

<term name="StorageID" abstract="yes" extends="Term"/>
<term name="ReferencableStorageID" abstract="yes"
      extends="StorageID"/>

<term name="UniqueID" extends="ReferencableStorageID">
<comment>A StorageID is an object that represents a
storage locator for a hypermedia object.  The UniqueID
is the first type of StorageID. It contains a globally
unique identifier that can identifies an
object.</comment>

<field type="String" name="idvalue">
<comment>the unique id </comment>
</field>
</term>

<term name="UndefinedID" extends="StorageID">
<comment>A StorageID is an object that represents a
storage locator for a hypermedia object.  The
UndefinedID is the second type of StorageID. An
UndefinedID inside an object says that the object
is not stored anywhere (it was probably generated
dynamically).</comment>
</term>
```

```
<term name="LocalID" extends="ReferencableStorageID">
<comment>A StorageID is an object that represents a
storage locator for a hypermedia object. A LocalID
specifies a name that is unique only to this
communication. This allows dynamic content that is
not stored anyway to be referenced by ID within a
single communication.</comment>
<field type="String" name="localvalue">
<comment>the local id </comment>
</field>
</term>

<term name="LocSpec" abstract="yes" extends="Term"/>

<term name="NameLoc" extends="LocSpec">
<comment>A LocSpec is an object that represents a
selection within an object. The NameLoc identifies a
selection by name. For example in a CAD system it
could refer to specific objects within a file (Data
object).</comment>

<field type="String" name="name">
<comment>the referenced name</comment>
</field>
</term>

<term name="RegionLoc" extends="LocSpec">
<comment>A RegionLoc is an object that represents a
region within an object. The RegionLoc identifies a
selection by defining zero or more axes and then
defining selections on those axes.  For example for an
image it might define two axes x and y and then points
on those axes would select a polygon.</comment>

<field type="RegionContent" name="regioncontent">
<comment>the content of the loc</comment>
</field>
<field type="Region" name="region">
<comment>the region of the loc</comment>
```

```
</field>
</term>

<term name="UndefinedLoc" extends="LocSpec">
<comment>A LocSpec is an object that represents a
region within an object. However it is possible to
select an entire object rather than a region. In
these cases an UndefinedLoc is used.</comment>
</term>

<term name="Region" abstract="yes" extends="Term"/>

<term name="TimeRegion" extends="Region">
<comment>A TimeRegion defines a region via a start
and end point (given in secs)</comment>

<field type="String" name="start">
<comment>the start time in seconds (offset
from the beginning of the media)</comment>
</field>
<field type="String" name="end">
<comment>the end time in seconds (offset
from the beginning of the media)</comment>
</field>
</term>

<term name="TextRegion" extends="Region">
<comment>A TextRegion defines a region via a start
and end character</comment>

<field type="String" name="start">
<comment>the start character (offset from
the beginning of the file)</comment>
</field>
<field type="String" name="end">
<comment>the end character (offset from
the beginning of the file)</comment>
</field>
</term>
```

```
<term name="ImageRegion" extends="Region">
<comment>An ImageRegion defines a region via two
x,y coordinate pairs</comment>

<field type="String" name="x1">
<comment>the x value of the upper left
co-ordinate</comment>
</field>
<field type="String" name="y1">
<comment>the y value of the upper left
co-ordinate</comment>
</field>
<field type="String" name="x2">
<comment>the x value of the lower right
co-ordinate</comment>
</field>
<field type="String" name="y2">
<comment>the y value of the lower right
co-ordinate</comment>
</field>
</term>

<term name="UndefinedRegion" extends="Region">
<comment>An UndefinedRegion is used to indicate
that a region has not been defined and that the
location depends on the RegionContent</comment>
</term>

<term name="RegionContent" abstract="yes" extends="Term"/>

<term name="DataRegionContent" extends="RegionContent">
<comment>A RegionContent is an object that
represents the contents of a selection within an
object.  The DataRegionContent holds that content
explicitly.</comment>

<field type="MediaObject" name="selection">
<comment>the media selected</comment>
```

```
</field>
</term>

<term name="UndefinedRegionContent" extends="RegionContent">
<comment>A RegionContent is an object that represents
the contents of a selection within an object.  The
UndefinedRegionContent is used when no regioncontent
has been specified explicitly.</comment>
</term>

<term name="StoreAssociation" extends="Action">
<field type="Association" name="assoc"/>
</term>

<term name="StoreData" extends="Action">
<field type="Data" name="data"/>
</term>

<term name="StoreReference" extends="Action">
<field type="Reference" name="ref"/>
</term>

</ontology>
```

References

1997 (Oct.). *Open Document Management API Version 1.5*. Tech. rept. The Association for Information and Image Management (AIIM). Available as http://www.aiim.org/odma/odma15.htm (last accessed Oct 2000).

2000. *XML-RPC*. UserLand Software, Inc. Available as http://lcweb.loc.gov/marc/naming.html (last accessed Oct 2000).

Akscyn, Robert, McCracken, Donald L., & Yoder, Elise A. 1988. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*, **31**(July), 820–835.

Anderson, Kenneth M. 1997. A Critique of the Open Hypermedia Protocol. *Pages 11–17 of: Proceedings of the 3rd Workshop on Open Hypermedia Systems, ACM Hypertext '97, Southampton, UK, April 6-11. Available as Report No. SR-97-01 from the Danish National Centre for IT Research, 8000 Aarhus C, Denmark.*

Anderson, Kenneth M. 1998. Client-Side Services for Open Hypermedia - Getting past the "foo". *Pages 15–21 of: Proceedings of the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98 Conference, Pittsburgh, PA, June 20-24. Available as Report No. CS-98-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Anderson, Kenneth M., Taylor, Richard N., & Whitehead, E. James. 1994. Chimera: Hypertext for Heterogeneous Software Environments. *Pages 94–197 of: ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK.*

Anderson, Kenneth M., Taylor, Richard N., & Whitehead, E. James. 1997. A Critique of the Open Hypermedia Protocol. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems*, **1**(2).

Austin, J. L. 1975. *How to do things with Words*. Oxford: Clarendon Press. first published 1962.

Berners-Lee, Tim. 1994 (June). *Universal Resource Identifiers in WWW. A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web*. Tech. rept. Internet RFC 1630.

Berners-Lee, Tim, Cailliau, Robert, Luotonen, A., Nielsen, Henrik Frystyk, & Secret, A. 1994. The World Wide Web. *Communications of the ACM*, **37**(8), 76–82.

Blackburn, Steven G., & De Roure, David C. 1998 (Sept.). A Tool for Content Based Navigation of Music. *Pages 361–368 of: Multimedia '98, Bristol, UK*.

Boy, G. A. 1991. Indexing Hypertext Documents in Context. *Pages 51–61 of: Proceedings of the '91 ACM Conference on Hypertext, Dec. 15-18, 1991, San Antonio, TX*.

Brusilovsky, Peter. 1996. Methods and techniques of adaptive hypermedia. *Pages 87–129 of: User Modelling and User-Adapted Interaction : Special Issue on adaptive hypertext and hypermedia*, vol. 6. Berlin/Heidelberg/New York: Kluwer academic publishers.

Bush, Vannevar. 1945. As We May Think. *The Atlantic Monthly*, **176**(1), 101–108.

Campbell, Brad, & Goodmann, Joseph M. 1988. HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM*, **31**(7), 856–861.

Casanova, M. A., & Tucherman, L. 1991. The Nested Context Model for Hyperdocuments. *Pages 193–201 of: Proceedings of the '91 ACM Conference on Hypertext, Dec. 15-18, 1991, San Antonio, TX*.

Christodoulakis, Stavros, & Triantafillou, Peter. 1995. Research and development issues for large-scale multimedia information systems. *ACM Computing Surveys*, **27**(4), 576–579.

Christophides, Vassilis, & Rizk, Antoine. 1994. Querying Structured Documents with Hypertext Links using OODBMS. *Pages 186–197 of: ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK*.

Conklin, Jeff. 1987. Hypertext: An Introduction and Survey. *IEEE Computer*, **20**(Sept.), 17–41.

Crowder, Richard, Wills, Gary, Heath, Ian, & Hall, Wendy. 1997. The Application of Hypermedia in the Factory Information Environment. *Pages 411–415 of: IEE International Conference Factory 2000, Cambridge*.

Davis, Hugh. 1995. *Data Integrity Problems in an Open Hypermedia Link Service*. Ph.D. thesis, University of Southampton.

Davis, Hugh, Reich, Siegfried, & Millard, David. 1997. *A Proposal for a Common Navigational Hypertext Protocol*. Tech. rept. Dept. of Electronics and Computer Science. Presented at 3.5 Open Hypermedia System Working Group Meeting. Aarhus University, Denmark. September 8-11.

Davis, Hugh C., Hall, Wendy, Heath, Ian, Hill, Gary J., & Wilkins, Robert J. 1992. Towards an Integrated Information Environment with Open Hypermedia Systems. *Pages 181–190 of: ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy*.

Davis, Hugh C., Rizk, Antoine, & Lewis, Andy J. 1996. OHP: A Draft Proposal for a Standard Open Hypermedia Protocol. *Pages 27–53 of:* Wiil, Uffe Kock, & Demeyer, Serge (eds), *Proceedings of the 2nd Workshop on Open Hypermedia Systems, ACM Hypertext '96, Washington, D.C., March 16-20. Available as Report No. ICS-TR-96-10 from the Dept. of Information and Computer Science, University of California, Irvine*.

Davis, Hugh C., Millard, David E., Reich, Sigi, Bouvin, N., Grnbk, K., Nrnberg, P. J., Sloth, L., Wiil, U. K., & Anderson, K. M. 1999 (Feb.). Interoperability between hypermedia systems: The standardisation work of the ohswg. *Pages 201–202 of: Hypertext '99, the 10th acm conference on hypertext and hypermedia, darmstadt, february 21-25, 1999*. ACM.

Delisle, N., & Schwartz, M. 1986. Neptune: A hypertext system for CAD applications. *Pages 132–142 of: ACM SIGMOD '86, Washington D.C.*

Delisle, N. M., & Schwartz, M. D. 1987. Contexts - A Partitioning Concept For Hypertext. *ACM Transactions on Office Information Systems*, **5**(2), 168–186.

De Roure, David C., Hall, Wendy, Reich, Siegfried, Pikrakis, Aggelos, Hill, Gary J., & Stairmand, Mark. 1998. An Open Architecture for Supporting Collaboration on the Web. *Pages 90–95 of: WET ICE 98 — IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, June 17-19, Stanford University, California*.

Dervin, Brenda. 1997. Given a Context by any Other Name: Methodological Tools for Taming the Unruly Beast. *Pages 13–38 of: Information Seeking in Context*. Taylor Graham, London.

Dyke, Parunak H. Van. 1991. Don't Link Me In: Set-Based Hypermedia for Taxonomic Reasoning. *Pages 233–242 of: Proceedings of the '91 ACM Conference on Hypertext, Dec. 15-18, 1991, San Antonio, TX*.

Dyke, Parunak H. Van. 1993. Hypercubes Grow on Trees (and Other Observations from the Land of Hypersets). *Pages 73–81 of: Proceedings of the '93 ACM Conference on Hypertext, Nov. 14-18, 1993, Seattle, WA.*

Engelbart, Douglas C. 1962 (Oct.). *Augmenting human intellect: A conceptual framework*. Tech. rept. AFOSR-3223, Contract AF 49(638)-1024. Stanford Research Institute Technical Report, Palo Alto, CA.

Engelbart, Douglas C. 1963. *A Conceptual Framework for the Augmentation of Man's Intellect*. Vol. 1. London: Spartan Books. Pages 1–29.

Engelbart, Douglas C. 1990 (Oct.). Knowledge-Domain Interoperability and an open Hyperdocument System. *Pages 143–156 of: CSCW 90. Proceedings of the Conference on Computer-Supported Cooperative Work.*

FIPA. 1997 (Nov.). *FIPA 97 Specification, Part 2: Agent Communication Language*. Tech. rept. Foundation for Intelligent Physical Agents, Geneva, Switzerland.

Fountain, Andrew M., Hall, Wendy, Heath, Ian, & Davis, Hugh C. 1990. MICROCOSM: An Open Model for Hypermedia With Dynamic Linking. *Pages 298–311 of:* Rizk, A., Streitz, N., & André, J. (eds), *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)*. Cambridge University Press.

Furuta, R., & Stotts, P. 1990a. A functional meta–structure for hypertext models and systems. *Electronic Publishing—Origination, Dissemination and Design, November 1990*, **3**(4), 179–205.

Furuta, Richard, & Stotts, David P. 1990b. The Trellis Hypertext Reference Model. *Pages 144–155 of: Proceedings of the 17. International SIGGRAPH Conference on Computer Graphics and Interactive Techniques*.

Gibney, M., & Jennings, N. 1998. Dynamic Resource Allocation by Market-Based Routing in Telecommunications Networks. *Pages 102–117 of: Intelligent Agents for Telecommunications Applications 1998, Springer Verlag*.

Goose, Stuart, Dale, Jonathan, Hill, Gary J., De Roure, David C., & Hall, Wendy. 1996 (June). An open framework for integrating widely distributed hypermedia resources. *Pages 364–371 of: Ieee conference on multimedia and computing systems, hiroshima*.

Goose, Stuart, Lewis, Andy J., & Davis, Hugh C. 1997. OHRA: Towards an Open Hypermedia Reference Architecture and a Migration Path for Existing Systems. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems*, **1**(2).

Grice, H. P. 1975. Logic and conversation. *Syntax and Semantics, Speech Acts. P. Cole and J. Morgan, Eds. Academic Press, New York*, **3**.

Grønbæk, Kaj. 1998. Interoperability - issues beyond the protocol. *Pages 33–38 of: Proceedings of the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98 Conference, Pittsburgh, PA, June 20-24. Available as Report No. CS-98-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Grønbæk, Kaj, & Sloth, Lennert. 1999. Supporting interchange of open hypermedia structures and contents. *Pages 34–37 of:* Wiil, Uffe Kock (ed), *Proceedings of the 5th Workshop on Open Hypermedia Systems, ACM Hypertext '99 Conference, Darmstadt, Germany, February 21-25. Available as Report No. CS-99-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Grønbæk, Kaj, & Trigg, Randall H. 1994. Design issues for a Dexter-based hypermedia system. *Communications of the ACM*, **37**(3), 40–49.

Grønbæk, Kaj, & Trigg, Randy. 1996. Toward a Dexter-based Model for Open Hypermedia: Unifying Embedded References and Link Objects. *Pages 149–160 of: Proceedings of the '96 ACM Conference on Hypertext, March 16-20, 1996, Washington, D.C.*

Grønbæk, Kaj, & Wiil, Uffe K. 1997. Towards a Reference Architecture for Open Hypermedia. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems*, **1**(2).

Grønbæk, Kaj, Bouvin, Niels Olof, & Sloth, Lennert. 1997. Designing Dexter-based hypermedia services for the World Wide Web. *Pages 146–156 of: Proceedings of the '97 ACM Conference on Hypertext, April 6-11, 1997, Southampton, UK*.

Guenther, Rebecca. 1999. *Naming Conventions for Digital Resources*. Tech. rept. Library of Congress. Available as http://lcweb.loc.gov/marc/naming.html.

Haake, Anja. 1992. CoVer: A Contextual Version Server for Hypertext Applications. *Pages 43–52 of: ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy*.

Haake, Jörg M., & Wang, Weigang. 1998. Collaboration Support in Open Hypermedia Environments. *Pages 39–44 of: Proceedings of the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98 Conference, Pittsburgh, PA, June 20-24. Available as Report No. CS-98-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Halasz, Frank. 1988. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, **31**(7), 836–852.

Halasz, Frank. 1991. Seven Issues: Revisited. The Hypertext '91 Closing Plenary. *Proceedings of the '91 ACM Conference on Hypertext, Dec. 15-18, 1991, San Antonio, TX*, Dec.

Halasz, Frank, & Schwartz, Mayer. 1994. The Dexter Hypertext Reference Model. *Communications of the ACM*, **37**(2), 30–39.

Hardman, L., Bulterman, D. C. A., & van Rossum, G. 1994. The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *Communications of the ACM*, **37**, 50 – 63.

Hardman, Linda, & Bulterman, Dick C.A. 1995 (Nov.). Using the Amsterdam Hypermedia Model for Abstracting Presentation Behavior. *In: Electronic Proceedings of the ACM Workshop on Effective Abstractions in Multimedia, 1995, San Francisco, CA, USA.*

Hardman, Linda, Bulterman, Dick C.A., & van Russum, Guido. 1993. Links in Hypermedia: the Requirement for Context. *Pages 183–191 of: Proceedings of the '93 ACM Conference on Hypertext, Nov. 14-18, 1993, Seattle, WA.*

Hill, Gary, & Hall, Wendy. 1994. Extending the Microcosm Model to a Distributed Environment. *Pages 32–40 of: ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK.*

Hirata, Kyoji, Hara, Yoshinori, Shibata, Naoki, & Hirabyashi, Fusako. 1993. Media-based Navigation for Hypermedia Systems. *Pages 159–173 of: Proceedings of the '93 ACM Conference on Hypertext, Nov. 14-18, 1993, Seattle, WA.*

Kappe, F., Maurer, H., & Scherbakov, N. 1993. Hyper-G – A Universal Hypermedia System. *J.EMH (Journal of Educational Multimedia and Hypermedia)* , **2**(1), 39–66.

Labrou, Yannis, & Finin, Tim. 1997 (Feb.). *A Proposal for a new KQML Specification*. Tech. rept. TR CS-97-03. Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250.

Lesser, V. 1991. A Retrospective View of FA/C Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed AI.*, **21**(6), 1347–1362.

Lewis, Paul H., Davis, Hugh C., Griffiths, Steve R., Hall, Wendy, & Wilkins, Robert J. 1996 (Mar.). Media-based Navigation with Generic Links. *Pages 215–223 of: Proceedings of the '96 ACM Conference on Hypertext, March 16-20, 1996, Washington, D.C.*

Lowe, David, & Hall, Wendy. 1999. *Hypermedia & the Web. An Engineering Approach*. Chichester: John Wiley & Sons.

Malcolm, Kathryn C., Poltrock, Steven E., & Schuler, Douglas. 1991. Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise. *Pages 13–24 of: Proceedings of the '91 ACM Conference on Hypertext, Dec. 15-18, 1991, San Antonio, TX*.

Marshall, Catherine C., & Irish, Peggy M. 1989. Guided Tours and On-Line Presentations: How Authors Make Existing Hypertext Intelligible for Readers. *Pages 15–26 of: Proceedings of the '89 ACM Conference on Hypertext, Nov. 5-9, 1989, Pittsburgh, PA.*

Marshall, Catherine C., & Shipman, Frank M. 1995. Spatial Hypertext: Designing for Change. *Communications of the ACM*, **38**, 88–97.

Marshall, Catherine C., & Shipman, Frank M. 1997. Spatial Hypertext and the Practice of Information Triage. *Pages 124–133 of: Proceedings of the '97 ACM Conference on Hypertext, April 6-11, 1997, Southampton, UK.*

Marshall, Catherine C., Shipman, Frank, & Coombs, James H. 1994. VIKI: Spatial hypertext supporting emergent structure. *Pages 13–23 of: ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK.*

McCracken, D., & Akscyn, R. 1984. Experience with the ZOG Human-Computer Interface System. *International Journal of Man-Machine Studies*, **21**, 293–310.

Millard, David, & Davis, Hugh. Navigating Spaces: The Semantics of Cross Domain Interoperability. *Pages 129–139 of:* Reich, Siegfried, & Anderson, Kenneth M. (eds), *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743).*

Millard, David, Davis, Hugh, & Moreau, Luc. Standardizing Hypertext: Where Next for OHP? *Pages 3–12 of:* Reich, Siegfried, & Anderson, Kenneth M. (eds), *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743)*.

Millard, David E., Reich, Siegfried, & Davis, Hugh C. 1998 (June). Reworking OHP: the Road to OHP-Nav. *Pages 48–53 of:* Wiil, Uffe Kock (ed), *Proceedings of the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98 Conference, Pittsburgh, PA, June 20-24. Available as Report No. CS-98-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark*.

Millard, David E., Reich, Siegfried, & Davis, Hugh C. 1999. Dynamic Service Discovery and Invocation in OHP. *Pages 38–42 of:* Wiil, Uffe Kock (ed), *Proceedings of the 5th Workshop on Open Hypermedia Systems, ACM Hypertext '99 Conference, Darmstadt, Germany, February 21-25. Available as Report No. CS-99-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark*.

Millard, David E., Moreau, Luc, Davis, Hugh C., & Reich, Siegfried. 2000. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. *Pages 93–102 of: Proceedings of the '00 ACM Conference on Hypertext, May 30 - June 3, San Antonio, TX*.

Moline, Judi, Benigni, Dan, & Baronas, Jean (eds). 1990 (Jan.). *Proceedings of the hypertext standardization workshop*. Nist Special Publication 500-178. National Institute of Standards and Technology.

Moore, Graham, & Moreau, Luc. From Metadata to Links. *Pages 77–86 of:* Reich, Siegfried, & Anderson, Kenneth M. (eds), *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743)*.

Moreau, Luc, Gibbins, Nick, DeRoure, David, El-Beltagy, Samhaa, Hall, Wendy, Hughes, Gareth, Joyce, Dan, Kim, Sanghee, Michaelides, Danius, Millard, Dave, Reich, Sigi, Tansley, Robert, & Weal, Mark. 2000 (Apr.). SoFAR with DIM Agents. An Agent Framework for Distributed Information Management. *In: The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, Monday April 10 - Wednesday April 12, 2000, Manchester, UK*.

Nelson, T. H. 1967. Getting it out of our system. *Pages 191–210 of:* Schechter, G. (ed), *Information Retrieval: A Critical Review*. Washington, D.C.: Thompson Books.

Nelson, Theodor Holm. 1987. *Literary Machines*. Published by the author. Mindful Press.

Neumüller, Moritz. Applying Computer Semiotics to Hypertext Theory and the World Wide Web. *Pages 58–65 of:* Reich, Siegfried, & Anderson, Kenneth M. (eds), *OHS*

6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743).

Newcomb, Steven R., Kipp, Neill A., & Newcomb, Victoria T. 1991. HyTime: the Hypermedia/Time-based Document Structuring Language. *Communications of the ACM*, **34**(11), 67–83.

Nürnberg, Peter J. 1997. *HOSS: An Environment to Support Structural Computing*. Ph.D. thesis, Department of Computer Science, Texas A&M University, College Station, TX.

Nürnberg, Peter J., & Ashman, Helen. 1999. What was the question? reconciling open hypermedia and world wide web research. *Pages 83–90 of: Proceedings of the '99 ACM Conference on Hypertext, February 21-25, 1999, Darmstadt, Germany*.

Nürnberg, Peter J., & Leggett, John J. 1997. A Vision for Open Hypermedia Systems. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems*, **1**(2).

Nürnberg, Peter J., Schneider, Erich R., & Leggett, John J. 1996. Designing digital libraries for the hyper-literate age. *Journal of Universal Computer Science*, **2**(9).

Nürnberg, Peter J., Leggett, John J., & Schneider, Erich R. 1997. As We Should Have Thought. *Pages 96–101 of: Proceedings of the '97 ACM Conference on Hypertext, April 6-11, 1997, Southampton, UK*.

Nürnberg, Peter J., Leggett, John J., & Wiil, Uffe K. 1998. An Agenda for Open Hypermedia Research. *Pages 198–206 of: Proceedings of the '98 ACM Conference on Hypertext, June 20-24, 1998, Pittsburgh, PA.*

Nürnberg, Peter J., Grønbæk, Kaj, Bucka-Lassen, Dirk, Pedersen, Claus A., & Reinert, Olav. 1999. A component-based open hypermedia approach to integrating structure services. *New Review of Hypermedia and Multimedia*. Accepted for publication.

OMG. 1991 (Dec.). *The Common Object Request Broker: Architecture and Specification*. Tech. rept. Object Management Group (OMG), OMG Document Number 91.12.1, Revision 1.1.

Østerbye, Kasper, & Wiil, Uffe Kock. 1996. The Flag taxonomy of Open Hypermedia Systems. *Pages 129–139 of: Proceedings of the '96 ACM Conference on Hypertext, March 16-20, 1996, Washington, D.C.*

Østerbye, Kasper, & Wiil, Uffe Kock. 1998. Using the Flag Taxonomy to Study Hypermedia System Interoperability. *Pages 188–197 of: Proceedings of the '98 ACM Conference on Hypertext, June 20-24, 1998, Pittsburgh, PA.*

Pearl, Amy. 1989. Sun's Link Service: A Protocol for Open Linking. *Pages 137–146 of: Proceedings of the '89 ACM Conference on Hypertext, Nov. 5-9, 1989, Pittsburgh, PA.*

Pikrakis, Aggelos, Bitsikas, Tilemahos, Sfakianakis, Stelios, Hatzopoulos, Mike, De Roure, David C., Hall, Wendy, Reich, Siegfried, Hill, Gary J., & Stairmand,

Mark. 1998 (Mar.). MEMOIR — Software Agents for Finding Similar Users by Trails. *Pages 453–466 of: PAAM98. The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents. March 23-25, London, UK.*

Reich, Siegfried, Millard, David E., & Davis, Hugh C. 1999a. Naming in OHP. *Pages 43–47 of:* Wiil, Uffe Kock (ed), *Proceedings of the 5th Workshop on Open Hypermedia Systems, ACM Hypertext '99 Conference, Darmstadt, Germany, February 21-25. Available as Report No. CS-99-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Reich, Siegfried, Griffiths, Jon P., Millard, David E., & Davis, Hugh C. 1999b. Solent — a Platform for Distributed Open Hypermedia Applications. *Pages 802–811 of:* Bench-Capon, Trevor, Soda, Giovanni, & Tjoa, A Min (eds), *Database and Expert Systems Applications. 10th Intl. Conference, DEXA 99, Florence, Italy*. LNCS, vol. 1677. Berlin/Heidelberg/New York: Springer.

Reich, Siegfried, Wiil, Uffe K., Nürnberg, Peter J., Davis, Hugh C., Grønbæk, Kaj, Anderson, Kenneth M., Millard, David E., & Haake, Jörg M. 2000. Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol. *New Review of Hypermedia and Multimedia*, 207–243.

Reinert, Olav, Bucka-Lassen, Dirk, Pedersen, Claus A., & Nürnberg, Peter J. 1999 (Feb.). CAOS: A Collaborative and Open Spatial Structure Service Component with Incremental Spatial Parsing. *Pages 49–50 of: Proceedings of the '99 ACM Conference on Hypertext, February 21-25, 1999, Darmstadt, Germany*.

Rizk, Antoine, & Sauter, Louis. 1992. Multicard: An open hypermedia system. *Pages 4–10 of: ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy*.

Rosenberg, Jim. Domain Requirements for a Cybertext Authoring System. *Pages 98–107 of:* Reich, Siegfried, & Anderson, Kenneth M. (eds), *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743).*

Rui, Yong, Huang, Thomas S., & Chang, Shih-Fu. 1999. Image Retrieval: Current Techniques, Promising Directions, and Open Issues. *Journal of Visual Communication and Image Representation*, **10**, 39–62.

Saeed, John I. 1997. *Semantics*. Oxford: Blackwell Publishers Ltd.

Schnase, John L., Leggett, John L., Hicks, David L., Nuernberg, Peter J., & Sánchez, J. Alfredo. 1993. Design and implementation of the HB1 hyperbase management system. *Electronic Publishing—Origination Dissemination and Design*, **6**(1), 35–63.

Schutt, Helge A., & Streitz, Norbert A. 1990. HyperBase: A Hypermedia Engine Based on a Relational Data Base Management System. *In: European Conference on Hypertext Technology (ECHT) 1990.*

Shackelford, Douglas E., Smith, John B., & Smith, F. Donelson. 1993. The architecture and implementation of a distributed hypermedia storage system. *Pages 1–13 of: Proceedings of the '93 ACM Conference on Hypertext, Nov. 14-18, 1993, Seattle, WA.*

Sollins, K., & Masinter, L. 1994 (Dec.). *Functional requirements for uniform resource names*. Tech. rept. Internet RFC 1737.

Stotts, P., & Furuta, R. 1989. Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*, **7**(1), 3–29.

Streitz, Norbert A., Haake, Jörg M., Hannemann, Jürg, Lemke, Andreas, Schuler, Wolfgang, Schütt, Helge, & Thüring, Manfred. 1992. SEPIA: a cooperative hypermedia authoring environment. *Pages 11–22 of: ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy*.

Tanenbaum, Andrew S. 1995. *Distributed Operating Systems*. Englewood Cliffs, New Jersey: Prentice-Hall International.

Thüring, Manfred, Hannemann, Jörg, & Haake, Jörg M. 1995. Hypermedia and Cognition: Designing for Comprehension. *Communications of the ACM*, **38**(8), 57–66.

Trigg, Randy. 1998. A Straw Model for Link Traversal in Open Hypermedia Systems. *Pages 59–62 of: Proceedings of the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98 Conference, Pittsburgh, PA, June 20-24. Available as Report No. CS-98-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Tzagarakis, Manolis, Vaitis, Michalis, Papadopoulos, Athamasios, & Christodoulakis, Dimitris. 1999 (Feb.). The Callimachus Approach to Distributed Hypermedia. *Pages 47–48 of: Proceedings of the '99 ACM Conference on Hypertext, February 21-25, 1999, Darmstadt, Germany*.

Tzagarakis, Manolis, Reich, Siegfried, Karousos, Nikos, & Christodoulakis, Dimitris. 2000 (May). Naming as a Fundamental Concept of Open Hypermedia Systems. *Pages 103–112 of: Proceedings of the '00 ACM Conference on Hypertext, May 30 - June 3, San Antonio, TX*.

Vaitis, Michalis, Papadopoulos, Athanasios, Tzagarakis, Manolis, & Christodoulakis, Dimitris. Towards Structure Specification for Open Hypermedia Systems. *Pages 160–169 of:* Reich, Siegfried, & Anderson, Kenneth M. (eds), *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743).*

van Dam, A. 1988. Hypertext '87 Keynote Address. *Communications of the ACM*, **31**(7), 887–895.

Waldo, Jim. 1999 (Jan.). *Jini Architecture Overview*. Tech. rept. Sun Microsystems, Inc., 901 San Antionio Road, Palo Alto, CA 94303 U.S.A. Available as http://www.sun.com/jini/whitepapers/architecture.html (last accessed Oct 2000).

Wang, Weigang, & Haake, Jörg M. 1999. Implementation Issues on OHS-Based Workflow Services. *Pages 52–56 of:* Wiil, Uffe Kock (ed), *Proceedings of the 5th Workshop on Open Hypermedia Systems, ACM Hypertext '99 Conference, Darmstadt, Germany, February 21-25. Available as Report No. CS-99-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Whitehead, E. James. 1999 (Feb.). Control Choices and Network Effects in Hypertext Systems. *Pages 75–82 of: Proceedings of the '99 ACM Conference on Hypertext, February 21-25, 1999, Darmstadt, Germany*.

Wiil, Uffe Kock, & Leggett, John J. 1996. The HyperDisco Approach to Open Hypermedia Systems. *Pages 140–148 of: Proceedings of the '96 ACM Conference on Hypertext, March 16-20, 1996, Washington, D.C.*

Wiil, Uffe Kock, & Nürnberg, Peter J. 1998. Collaboration in Open Hypermedia Environments. *Pages 74–80 of: Proceedings of the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98 Conference, Pittsburgh, PA, June 20-24. Available as Report No. CS-98-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark.*

Wiil, Uffe Kock, & Nürnberg, Peter J. 1999 (Feb.). Evolving Hypermedia Middleware Services: Lessons and Observations. *Pages 427–436 of: Proceedings of of the 1999 ACM Symposium on Applied Computing (SAC '99), San Antonio, TX*.

Wiil, Uffe Kock, & Whitehead, E. James. 1997. Interoperability and Open Hypermedia Systems. *Pages 137–145 of: Proceedings of the 3rd Workshop on Open Hypermedia Systems, ACM Hypertext '97, Southampton, UK, April 6-11. Available as Report No. SR-97-01 from the Danish National Centre for IT Research, 8000 Aarhus C, Denmark.*

Wooldridge, M., & Jennings, N. R. 1995. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, **10**(2).

Yankelovich, N., Meyrowitz, N., & van Dam., A. 1985. Reading and Writing the Electronic Book. *IEEE Computer*, **18**(10), 81–96.

Yankelovich, Nicole, Haan, Bernard J., Meyrowitz, Norman K., & Drucker, Steven M. 1988. Intermedia: The Concept and the Construction of a Seamless Information Environment. *IEEE Computer*, **21**(1), 81–96.

Young, L. De. 1990. Linking Considered Harmful. *Pages 238–249 of:* Rizk, A., Streitz, N., & André, J. (eds), *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)*. Cambridge University Press.