

A Survey of Distributed Hypertext Systems

Joachim Feise
Institute for Software Research
University of California, Irvine
Irvine, CA 92627-3425, USA
jfeise@ics.uci.edu

ISR Technical Report # UCI-ISR-05-4
April 2005

Contents

1	Introduction	1
2	Early Hypermedia Systems	2
2.1	Xanadu	2
2.2	Augment	3
2.3	ABC	4
2.4	KMS	5
2.5	Virtual Notebook System	6
2.6	Sun Link Service	6
3	Second-Generation Hypermedia Systems	7
3.1	Microcosm TNG	7
3.2	HyperDisco	8
3.3	JPernLite	9
3.4	Callimachus	10
3.5	Hyper-G	10
3.6	Chimera	11
4	Web-based Hypermedia Systems	13
4.1	World Wide Web	13
4.2	Frontpage	14
4.3	WebDAV Web Extensions	14
4.4	DeltaV Web Extensions	14
5	Classification Framework	15
5.1	Links	15
5.1.1	Embedded Links	15
5.1.2	Uni-Directional Links	16
5.1.3	N-ary Links	16
5.1.4	Anchor Style	16
5.2	Document Storage	16
5.2.1	Hyperbase and Linkserver Combinations	16
5.2.2	Replication and Caching	16
5.3	Concurrency Control	17
5.3.1	Locking	17
5.3.2	Change History	17
5.3.3	Metadata	17
5.3.4	Access Control	17
5.4	Naming and Locating	18
5.5	Communication and Interoperability	18
5.6	Framework matrix	18
5.6.1	Links	18
5.6.2	Document Storage	20
5.6.3	Concurrency Control	21
5.6.4	Naming and Communication	22
6	Conclusion	24
7	Acknowledgements	25

List of Figures

1	Xanadu Document Architecture	3
2	Augment Architecture	3
3	ABC Distributed Graph Server Architecture	4
4	KMS Architecture	5
5	VNS Server Architecture	6
6	Sun Link Service Architecture	6
7	The Microcosm TNG Architecture	7
8	The HyperDisco Architecture	8
9	JPernLite Transaction Server Architecture	9
10	The Callimachus Architecture	10
11	The Hyper-G Architecture	11
12	Chimera 2.0 Architecture	12
13	WebDAV Architecture	14
14	DeltaV Architecture	15

List of Tables

1	Link aspects	19
2	Document storage aspects	20
3	Concurrency control aspects	21
4	Naming and communication aspects	22

1 Introduction

In 1945, Vannevar Bush observed that documents do not exist in isolation [12]. Rather, there exist a number of relationships between documents. These relationships can be explicit, for example references in academic articles or in books; other references can be implicit, like similarity in style or content. Hypertext systems help capture, represent, and navigate these relationships.

The term *hypertext* was coined by Ted Nelson [51] to denote the relationships between textual documents. The emergence of non-textual documents, from spreadsheets and drawings to audio, video and multimedia resulted in a gradual shift from the hypertext term to the more general *hypermedia* term (which was first used by Nelson in [52].)

The relationships between documents usually stretch beyond the data storage of one person, just as an author of an academic paper refers to academic articles of other authors and not just his or her own. Unlike in the domain of academic papers, documents in the hypermedia domain tend to be dynamic: users edit documents, modifying the relationships between them.

In particular, there is no single authority or clearinghouse through which document changes have to get approved. Authors of documents are free to change their data as they see fit, without consideration of the relationship of their particular documents to other authors' documents.

While the need to manage distributed documents was recognized early by both Nelson and Doug Engelbart [22], most early hypermedia system implementations were limited to single machines. Rather, the focus was on the representational and navigational aspects of hypermedia.

The few early hypermedia systems that took distribution issues into consideration were targeted towards specific application domains which had a history of collaboration between the users. For example, the Virtual Notebook System [62] aimed to support collaboration in a biomedical setting [29] by acting as an electronic analogue of a scientist's workbook. Another early hypermedia system, the Sun Link Service [57], arose out of the insights into networked environments.

The second-generation hypermedia systems also largely ignored issues of distribution. The rise of the World Wide Web, however, highlighted the problems of navigating in a distributed hypermedia environment. As a consequence, the developers of single user, single workstation hypermedia systems retrofitted their systems to allow the management of documents distributed across a network. Developers of hypermedia systems that already provided access to distributed documents saw the rise of the Web as an opportunity to widen the reach of their distributed models, and started to incorporate the use of the Web into their systems.

The focus on navigation in a document space as exemplified by the Web represents one view of hypermedia common among researchers. The other common view holds that navigation is but one feature provided by hypermedia. The research on open hypermedia systems emphasizes the latter view [56]. The feature set of these systems can include authoring and changing documents, as well as creating and maintaining links between documents.

Distributed hypermedia systems provide the ability for collaboration of a possibly arbitrary large group of people. These people can be located in physically separate locations, in different organizations, with possibly intermittent network connectivity, and with possibly high latencies for data transfer.

Tools designed to facilitate collaboration under these circumstances have to be able to handle a variety of problems:

Access Permissions: Not all users should necessarily be allowed to add or modify documents at specific locations. Users may not be allowed to link to specific documents or navigate to them. A distributed hypermedia system needs to provide mechanisms to specify the access permissions for documents accordingly.

Link Maintenance: Whenever documents in a hypermedia system are modified, links to these documents may become invalid. Hypermedia systems employ a variety of mechanisms to detect and possibly repair invalid links. In a distributed setting, these tasks can become much more complicated, since links can be distributed as well.

Document Consistency: In collaborative environments, it is entirely possible that several users simultaneously want to change the same document. Without a means to serialize document access, changes to documents may get lost.

Document Metadata: Since document modification in a distributed setting is no longer the task of a single person, it becomes important to provide information with the document to allow determination of who changed which document when. Such a document history log is only one use of document metadata. Metadata can be used for a wide variety of purposes, for example to facilitate workflow tasks.

Replication: In a distributed environment, especially in a large-scale environment like the Internet, network connectivity can be disrupted for a variety of reasons. Network latency can fluctuate widely, with the result of rendering some remote documents offline for some amount of time. A distributed hypertext system should be able to provide access to remote documents even if the remote site is temporarily offline. Document replication or caching, in combination with synchronization when the remote site becomes available again are common strategies to provide for high document availability in distributed environments.

The systems included in this survey all claim to provide some support for operation in a distributed environment. The majority of hypermedia systems were developed as single-user systems or centralized multi-user systems. Such systems are excluded here. However, systems that started life as single-user systems or centralized systems, but were later adapted to a distributed environment are included in this survey. This particularly affects several hypermedia systems developed in the 1990s, in parallel with the appearance of the Web. Several of these systems were modified to take advantage of the infrastructure provided by the Web and to operate within a Web-based distributed environment.

2 Early Hypermedia Systems

2.1 Xanadu

Ted Nelson, who coined the term Hypertext, started his project Xanadu [54] in 1960. While it has never been fully implemented, the design of Xanadu exhibited several groundbreaking ideas, from versioning to complex document structures, i.e., compound documents, to distributed storage to micropayment concepts.

Here, I focus on the distributed storage aspect of Xanadu, or the *docuverse*, as Nelson calls it. Documents in Xanadu are usually not closed, but rather built upon each other. Documents provide references, windows to other documents, as shown in figure 1. Every one of the referred documents can be located on a different machine.

Figure 1: Xanadu Document Architecture (from [54])

Since Nelson envisioned Xanadu to grow without limits and eventually provide access to all available literature, the necessity of storing the documents on a network of computers became obvious [53].

The goal was to unite the network of computers in such a way that the user experience does not change, i.e., it should become irrelevant where a particular document is stored.

To achieve this goal, Nelson envisioned dynamic replication of documents throughout the network, depending on demand. He also put emphasis on the need to have changes to any document be instantly known in the whole network.

Nelson described as “most important thing” and the hardest part the problem of response time. He described as his goal that “all documents become a single instantaneous whole”, which requires response times in the order of the times required to set up telephone connections (in the 1993 edition of [54] he advocates high-capacity connectivity like satellite links between the networked machines.)

Of course, such a distributed storage concept has a big influence on other important aspects of the usability of the system.

For example, versioning and concurrency issues had to be addressed in the system. Nelson’s solution was the introduction of write-once storage. New versions of documents contain both new content and links to the old document, allowing access to the parts of the document that didn’t change.

This concept has the side-effect of making concurrency issues irrelevant, since users don’t modify documents, but rather always create new ones. A namespace definition inspired by the Dewey Decimal system allows the addressing and localization of servers, user accounts, document versions down to every byte in a document.

2.2 Augment

Augment [22] was developed in the 1960s by Doug Engelbart to support group collaboration in a networked multi-user environment. In particular, one explicit goal of Augment was to support “the development, production and control of complex technical documentation – through the whole cycle of gathering information, planning, creating, collaborating, reviewing, editing, controlling versions, designing layout, and producing the final documents.” [22] Engelbart provided an architectural overview of Augment in [21] (see figure 2).

Figure 2: Augment Architecture (from [21])

Augment represents data in arbitrary windows on the screen. The data displayed can be text, graphics, or other arbitrary data. The ability to show multiple such windows provides for cross-file editing.

The data is stored in structured files, with nodes of up to 2000 characters of text, a graphic or other data, organized in a hierarchy. For textual data, a node often contains a section or paragraph, which is a natural way of structuring text.

Augment has multiple ways to identify data in a file. Nodes have unique identifiers that are assigned when the nodes are created. These identifiers are immutable. Each

node also has a structural statement number, assigned by the system, that identifies a node within the structure of a document. As a document is changed, the structural statement numbers of the document nodes are updated to reflect the changes. Document authors can also label nodes, and it is possible to place markers in arbitrary positions in a document. Finally, positions in a document can be specified with a relative address, anchored at an arbitrary fixed node.

These flexible addressing schemes allow Augment to address any item in a document, from chapter, section, paragraph down to individual words and even individual characters.

Links using the Augment addressing schemes can therefore point to arbitrary data within the documents in Augment. The links in Augment are unary, there is no back-linking facility. Augment treats links just like any other text. The links are embedded within the Augment documents.

Augment's addressing schemes are not limited to single files. So-called composite addresses combine the address of a specific file with an address within that file to arrive at unique system-wide addresses. To support distribution, the file address itself can be a composite address combining a compute name with a file name and a directory name where the file is located.

To support collaborative work on documents, Augment maintains change records for each node in each file. These records contain information about the date, time and the author of the creation of nodes or the last change of nodes. Augment has provisions to view and filter these records.

Augment also has the ability to mark documents as immutable, allowing the creation of a permanent record of documents marked this way.

2.3 ABC

The Artifact-Based Collaboration system (ABC) [64] was one of the early hypermedia developments providing a distributed environment for collaborative work.

ABC stores documents and links in a hyperbase management system called a Graph Server. The documents are represented as graph structures, with links providing connections between nodes in different subgraphs.

The graph server acts as a single logical system, but the data can in fact be distributed among several servers. Figure 3 outlines the architecture of the graph server. This distribution is facilitated by the use of graph structures to represent the data. Each server can maintain a single subgraph.

Figure 3: ABC Distributed Graph Server Architecture (from [64])

To manage this distributed graph structure, a distributed graph storage system was developed. Items in this storage system are identified by unique object ids. These ids consist of an entry specifying the server the data is stored on, and an entry identifying the actual data within the server. The server is not directly identified, but rather an indirection through a lookup table is used. This improves location independence of the data.

Browsers and other client applications like text editors, drawing programs, etc. utilize the graph server interface to browse and manipulate the data. Links can be uni-directional and bi-directional.

The system supports two kinds of links, structural links and hyperlinks. Structural links define the form of the graph structure. Constraints on the structural links con-

form with the graph structure. For example, in a tree structure, no node can have more than one incoming link. Hyperlinks, on the other hand, can provide a link between any two nodes, regardless of the structure.

The browsers within ABC are specific to particular graph structures, i.e., there are browsers for general graphs, trees, lists, etc. The browsers enforce the constraints on the particular graph structures they support. A tree browser can not be used to manipulate structural links when the change would violate the integrity of a tree.

ABC also supports applications operating on the document contents, such as text editors, drawing programs, or spreadsheet programs. The authors acknowledge the problems of maintaining link integrity when modifying the actual data, and propose to use wrappers communicating with browsers and servers to provide for link integrity.

Since the graph storage system used by ABC needs to support multiple collaborating users, the storage system has to provide a means to control concurrent access to objects. The graph storage system does so by providing access modes that allow different subsets of operations and prevent concurrent changes. These access modes are reminiscent of standard file system access modes, e.g., multiple read operations can occur simultaneously, but only one write operation at a time is allowed.

Finally, access to different parts of the graph structure is governed by the use of access control lists, stored with each object. The access control lists map users or groups of users to the operations that these users are allowed to perform on particular objects. ABC defines two authorization categories. Access authorizations allow the use, or access of particular objects, while administer authorizations allow users to change the access rights of objects.

2.4 KMS

The KMS system [1], a large-scale, distributed hypermedia system for use in a collaborative work environment, was developed in the 80ies at Carnegie-Mellon.

It presents data in fixed size frames, which can contain text and graphical items. The frames fill the whole screen. Items in frames can be linked to point to other frames. The items can also be used to start external programs. Figure 4 illustrates the system architecture.

Figure 4: KMS Architecture (from [1])

The data for these frames can be distributed across multiple file servers. KMS presents the data as a single logical database, so that the physical location of the data is transparent to the users.

Since links can only point to other frames and not to another item within a frame, links in KMS are unidirectional. A sort of backlink, pointing from an item on the second frame back to the first frame is obviously possible, but seems to be rather rare in actual KMS deployments. Conklin [19] provides an illustrative example of a possible KMS database structure. The paper uses this example to discuss the “strong hierarchical orientation of most KMS databases.”

KMS uses two different link types to distinguish between structural relationships such as lower-level frames in a hierarchical structure and associate relationships such as comments, annotations and cross-references.

Since KMS was intended for a collaborative work environment, users in KMS can modify each frame as they see fit. This of course introduces the problem of concurrent access. KMS uses an optimistic concurrency control mechanism. It does not lock

frames that are worked on by the users. To prevent users from overwriting each other's changes, KMS refuses to write a user's changes when the document has changed. The justification for this mechanism is that the unit of change, the frame, is small enough to make collisions unlikely. The authors suggest out-of-band, informal locking conventions for the cases when several users modify the same set of frames.

Frames in KMS can be frozen, which has the effect that subsequent changes to the frozen frames results in the creation of new versions. Additional frames are automatically created to provide access to the previous, frozen data.

2.5 Virtual Notebook System

The Virtual Notebook System (VNS) [62, 29], is a distributed hypertext system to support collaborative work in a biomedical setting, developed in the 80ies. It functions as an electronic analogue to a scientist's notebook.

The user interacts with VNS through a number of pages on a screen. Similar to KMS, pages can contain text and images. Pages can contain links that point to other pages in the system.

In VNS, links are stored separate from the data, so every user can have different sets of links connecting different pages. Any web of pages connected by links is referred to in the system as notebook. Users can have many notebooks, some of which may be shared by other users and some may be private. Links in VNS are uni-directional.

VNS also provides a filtering mechanism to support directed navigation by the user. This mechanism is used to ease the task of finding items in the hypertext.

The data is stored in work group servers, connected through a network. Each server stores the hypertext and the links in a relational database. Users can access the data in their local servers directly through the VNS. Figure 5 provides an overview of the server architecture.

Figure 5: VNS Server Architecture (from [29])

References in a page that are not located in the local server can be accessed through a special gatekeeper computer. The gatekeeper resolves such remote references through a mapping of object names to work group servers.

To facilitate concurrency control and access rights, VNS relies on the capabilities of the relational database system used to store the data and links.

2.6 Sun Link Service

The Sun Link Service [57] is a hypermedia system that only maintains the links between documents. It was the first system to be described as *Open* hypermedia system, in the sense that it does not make any assumption about the data that is linked. This openness makes it possible for users of the Sun Link Service to connect to objects located on local or remote filesystems, and to even link to documents maintained by other hypermedia systems. Figure 6 shows the architecture of the service.

Figure 6: Sun Link Service Architecture (from [57])

Since the Link Service is independent of the applications used to view and edit documents, the Link Service does not have any control over the documents. The Link Service was designed to minimize the impact on the look and feel of the applications.

Obviously, for the Link Service to be able to operate, the applications to be used with the Link Service have to have the ability to interact with the Link Service. This is accomplished through the integration of a link command panel, which is the same across applications. Applications must provide a mechanism for selecting documents, text areas or points where link endpoints are anchored. Applications also must be able to provide a visual indication to denote links.

Since the Link Service only manages links, the issue of link consistency becomes important. The Link Service provides an implicit and an explicit mechanism for maintaining link consistency. The implicit mechanism is invoked when a user follows a link which doesn't have a valid endpoint anymore. The Link Service informs the user about this problem, and suggests the deletion of the link. Of course, if both link endpoints become invalid, the link can not be discovered. The explicit link management is akin to a garbage collection mechanism. When invoked, all links are traversed to check their validity, and invalid links are then removed.

Links in the Sun Link Service are n-ary. When a link with multiple endpoints is selected, the user is presented with a dialog box to allow selection of the desired endpoint. Link service links don't have directionality.

Links in this service can cross machine boundaries, and multiple users can use the service in parallel. The Link Service does not address the problems of simultaneous changes to documents that arise in such multi-user environments. The Link server storing the links is a centralized database. However, the developers acknowledge these issues in a paragraph in their paper.

3 Second-Generation Hypermedia Systems

3.1 Microcosm TNG

Microcosm TNG [36] is an extension of the Microcosm system developed at the University of Southampton in the UK. It extends the Microcosm model to allow the distribution of data and processes across a network. It retains the core Microcosm functionality. Figure 7 shows the architecture of a typical Microcosm TNG system.

Figure 7: The Microcosm TNG Architecture (from [40])

Microcosm provides access to documents and the links structuring the data through a hyperbase maintained by itself. This ensures link validity, but requires that document viewers have to be written specifically for Microcosm. Depending on the flexibility of third-party document editors and viewers, it may be possible to integrate some of them with Microcosm.

Microcosm provides a Document Control System that is the sole point of interaction with the document viewers. It in turn interacts with a Filter Management System that allows processing of viewer requests through a linear chain of filters.

It was determined that using such a linear chain of filters with messages passing sequentially through all filters was inefficient, since all messages had to be routed through all active filters, even though not all filters were able to process the messages. While this inefficiency didn't significantly impair the performance of the original standalone Microcosm system, delivering messages across a network to remote filters would have caused a significant performance reduction, so the filter management was modified for Microcosm TNG. In Microcosm TNG, filters register their services with the

management system, so that the management system can decide which filters to invoke for specific messages.

This allowed the developers to provide specific filters that connect to remote servers to provide link services and retrieve data from remote servers. Since the filters providing this service are just another kind of filters, the remote access is completely transparent to the users.

The namesystem for Microcosm TNG uses a variant of the URL format [10], with a private protocol name. This allows easy extension to support other protocols, e.g., http or ftp.

Links in Microcosm and Microcosm TNG are bidirectional, n-ary links.

3.2 HyperDisco

Like Microcosm TNG, HyperDisco [79, 80] is a hypermedia system that not only provides support for linking, but also implements a hyperbase management system to store the hypermedia documents.

The HyperDisco infrastructure consists of HyperDisco enabled tools that use tool integrators to access hypermedia documents and links provided by one or more hyperbases, called workspaces in HyperDisco (see the high-level architecture in figure 8.)

Figure 8: The HyperDisco Architecture (from [79])

A workspace provides access to documents stored in the filesystem or in the hyperbase management system itself. The workspaces also manage linking services, i.e., anchoring and linking, as well as access controls, and support for collaboration, like locking and versioning.

Links in HyperDisco have a direction, but can be traversed in both directions. They are provided in multi-headed form, i.e., they are n-n links. The links are first-class objects, i.e., they exist independently from the hypermedia documents.

The original HyperDisco system, though, did not allow links to cross workspaces. For a distributed hypermedia system, though, it is crucial to provide the ability of linking to remote locations.

The distributed extension to HyperDisco therefore introduced the ability to specify the workspace name in the links. Link endpoints in HyperDisco are a triple containing a workspace name, a node identifier and an anchor identifier.

To maintain integrity of links involving multiple workspaces, HyperDisco introduced several safeguards. First, links are replicated across all workspaces the links point to. Second, HyperDisco prevents creation and destruction of links if one of the link endpoints is not reachable.

This limitation does not exist for link traversal, since link traversal does not have any effect on link integrity. If some workspaces can not be reached during link traversal, the affected endpoints are ignored. This behavior, together with the link replication, promotes autonomy of each workspace.

When the user follows links to remote workspaces, the appropriate hypermedia documents have to be retrieved from the remote site. The HyperDisco tool integrator was extended to provide the functionality to retrieve and store remote documents.

To locate remote documents, HyperDisco implements a two-level mapping system inspired by the URL specification. It starts by mapping workspace names to Internet host names and ports. This mapping is not automated, and instead maintained

locally by each user. This is a deviation from the centralized name service of the non-distributed version of HyperDisco.

Hypermedia files are uniquely identified by their path and filename within a workspace and the workspace name.

To facilitate collaboration, HyperDisco provides locking, versioning, and access control mechanisms.

The locks are fine-grained and operate on attributes. Granted locks are stored persistently in the hyperbase.

Documents in the hyperbase are versioned whenever a tool explicitly requests a new version. At that point, the old version becomes immutable. The literature did not go into details about the extent of support for versioning of the link structure, other than mentioning that the facilities for it are available, but the direct support for it is not.

HyperDisco support access controls on the node, link, and composite (aggregation of nodes and links) level to manage read, write, annotate, and delete privileges.

3.3 JPernLite

JPernLite [82] isn't a full hypermedia system, but rather provides a means to use the World Wide Web as a substrate for a transaction server.

The developers of JPernLite noticed that most web servers lack the features required for maintaining data consistency and the correct semantics of user-initiated operations. Especially, they identified the need for event notifications, user-controlled locking, persistency, and transaction support. These issues were also raised in greater detail by Fielding et. al. in [25] and [26].

To enable these features, JPernLite implements a middleware solution, an external transaction server. Figure 9 provides an overview of the transaction server architecture.

Figure 9: JPernLite Transaction Server Architecture (from [82])

Use of such a transaction server has the advantages that no changes to existing web servers or web clients are required unless they want to exploit the features of the transaction server. Users who just want to browse can continue to use their familiar web browsers. Another advantage is that the transaction server can be tailored to apply specific concurrency control features. The transaction server, with the help of data access plugins, can also access other data sources, for example hyperbases or Corba servers. It obviously also has significant limitations. It has to interoperate with a wide variety of web servers supporting a variety of concurrency control mechanisms, e.g., WebDAV [35] or DeltaV [16]. The most serious limitation is the separation of the transaction server from the web servers. Because the transaction server is independent from the web servers, it does not have control over the data. This is a limitation similar to the one present in link-service based hypermedia systems.

JPernLite can be accessed by clients using the standard HTTP protocol extended with several JPernLite-specific headers. These headers specify the operation to be performed as well as additional information like parameters and return values of the operation. JPernLite maintains objects representing the data on the web servers it has access to. Client programs can also directly access the web servers, but to be able to use concurrency controls, have to ask JPernLite for permission first. These concurrency controls are implemented by JPernLite through locks and timestamps.

The transaction support in JPernLite is limited to the objects that JPernLite itself manages, i.e., locks, timestamps and other attributes available within JPernLite. The data itself continues to be maintained by web servers outside of JPernLite, and it is therefore possible that data on a web server changes even while a user is in the middle of a transaction through JPernLite.

The designers of JPernLite provided for extensibility of the system. A callback mechanism allows the implementation of complex concurrency control and transaction mechanisms. For example, two-phase locking and lock expiration (a la WebDAV) were implemented by the designers through the extension mechanism.

Event notification can also be implemented this way.

3.4 Callimachus

The Callimachus system [69] provides access to distributed hyperbase management systems. The hyperbases, named contexts, are similar to the workspaces in Hyper-Disco, providing the basic hypermedia object of documents, nodes, links and anchors in its data model.

Applications using Callimachus utilize a client-server model to access hypermedia in multiple contexts. A high-level architectural view of Callimachus with a focus on structural aspects is provided in [70] (see figure 10.)

Figure 10: The Callimachus Architecture (from [70])

To support cross-context linking, Callimachus provides its own naming system, dubbed Context Name Service (CNS) [68]. The CNS provides a mapping of arbitrary names to attributes of anchors and services. A service is specified by the address and port of the computer hosting the service. An anchor is specified by an object identifier and the address and port of the context server that manages the anchor. This mapping allows the client applications to abstract from the actual location of the data, links, and anchors.

The mapping is dynamic, i.e., the actual mapping is performed only when an application wishes to resolve an anchor name. Client applications do need to know the top-level addresses of the contexts they are trying to reach.

3.5 Hyper-G

Hyper-G [41, 42] was designed as a large-scale, distributed hypermedia system. The developers aimed to address three problems associated with large-scale hypermedia systems: disorientation, authoring, and information distribution.

The Hyper-G approach to address disorientation relies on additional structuring and navigation facilities (collections and guided tours) and on enhanced support for searching. The authoring problem in this context does not refer to collaboration issues. The developers are rather concerned with the structure of hypermedia documents. The goal is, analogous to large-scale software development, to provide reusable hypertext modules with well-defined interfaces.

I focus here on the third problem identified by the Hyper-G developers, distribution. Hyper-G uses a client-server architecture. Client programs connect to a local Hyper-G server. The local server uses the Internet to retrieve data from other Hyper-G servers. Hyper-G can also interoperate with other hypermedia systems, like Gopher, WAIS, and the World Wide Web.

A Hyper-G server consists of a link server and a document server. A third server is used for interoperability purposes to Gopher or Web clients (see figure 11.)

Figure 11: The Hyper-G Architecture (from [42])

The central access point is the link server. Links are stored in the link server, independently from the documents. The link server stores more than just links, though. It assigns IDs to documents, and ensures that an ID uniquely describes a specific version of an document. Modified documents automatically receive new IDs. The link server also stores meta information about documents, such as title, author, creation date, etc. It also maintains the information needed to locate the document.

The link server also provides access control functionality, restricting access to documents and collections of documents to groups of users. Document creation and modification is also controlled by the link server.

The link server provides the functionality to search and navigate through the hypermedia space. To view or edit documents, they are retrieved from the document server. All requests for documents go to the local document server. If a document is located on a remote server, the request is forwarded to that server, and the local server retrieves the document from the remote server, caches it, and delivers it to the user. Subsequent requests for the document are then served from the cache in the local document server. To avoid the problem of stale cache copies, the unique document ID is used to determine if the cached copy is still valid. Modified documents get new IDs, so the cached stale versions of the documents are bypassed when such modified documents are requested.

3.6 Chimera

Chimera, developed by Ken Anderson at UCI [6, 4, 7], is a hypermedia system designed to handle the intrinsic heterogeneity of software development environments.

Users interact with Chimera through client programs that provide browsing and editing operations as well as hypermedia operations, like link traversal and anchor creation. These client programs contain one of more viewers, each one specializing in the display of a specific data type, and the Chimera API, which encapsulates the inter-process communication between the client program and the Chimera server, using the Chimera protocol.

Chimera does not store the hypermedia data, it employs the linkserver approach instead. Anderson describes the design decisions explored in Chimera as variations of a general *open vs. closed* theme. Open in this context means minimization of the assumptions made about the world outside the system. Closed systems, on the other hand, utilize requirements such as specific operating systems, communication protocols, or specific structures of the hypermedia model to achieve better quality of service as long as the requirements are met.

Chimera aims to be an open hypermedia system. In the trade-off between the consistency and increased service features of the hyperbase approach and the lower developmental effort of adapting applications to the hypermedia system in the linkserver approach, Chimera adopted the latter one. However, Chimera uses a closed hypermedia model. The use of a single hypermedia model, providing a fixed mapping of the abstract concepts of anchors, nodes and links to into the concrete representations used by the client applications enables the use of algorithms tuned to the model. The downside of this approach is that the model may not map appropriately to the needs of some applications. Chimera strives to minimize the disadvantages of a closed hypermedia

model, though. This is accomplished through extensions to the model that enable new concepts while maintaining existing functionality.

Chimera stores its hypermedia data in hyperwebs. A hyperweb contains a model of the user environment, and the links and anchors that define the relationships between the objects within the user environment. The Chimera server manages the hyperwebs. The server also implements the operations accessible through the Chimera API, and persists changes to the hyperweb made by the clients programs. To avoid inconsistencies, only one server at a time can manage a hyperweb. The server tracks the set of users accessing the hyperwebs it manages, the clients, and the viewers and views currently active within each client. This provides the server with the state information necessary to correctly react to requests from clients.

The Chimera server also has the ability to generate hypermedia events and route them to clients that have registered an interest in them.

Chimera allows multiple users to access a hyperweb simultaneously.

Links in Chimera are bi-directional and n-ary. When a link with multiple endpoints is traversed, the objects at all endpoints are displayed. Since links can refer to arbitrary data, it may be required to launch specific client programs to view the requested data. Chimera provides a process invoker to launch the appropriate client programs that are registered within the current hyperweb to be able to view the data. This results in a delay of the link traversal, until the newly invoked client has initialized. This situation is therefore known as *delayed link traversal*.

Chimera has been extended (and named Chimera 2.0) to explore integration possibilities between the World Wide Web and open hypermedia systems like Chimera. Chimera 2.0 explores the use of the Web to improve the hypermedia services it can provide. In particular, Chimera was originally not explicitly designed with distribution in mind. It's distribution support was limited to a local area network. While the components of the Chimera system, i.e., the server, process invoker and clients could be executed on different machines, they all required access to the same file system. The location of the server and the hyperwebs were specified using absolute filenames, making distribution past the local area network impossible. There also existed no means of communication between multiple Chimera servers.

The Chimera 2.0 architecture (see figure 12) leverages the protocols and mechanisms of the Web to incorporate support for the global distribution of hyperwebs.

Figure 12: Chimera 2.0 Architecture (from [4])

For Chimera 2.0, the Chimera server no longer manages the hyperwebs. Instead, a new Hyperweb server was introduced. The hyperweb structure itself was also improved. Hyperwebs can be organized hierarchically, and are associated with URLs. The hyperweb manager uses a dedicated Internet port number registered with the Internet Assigned Number Authority. This ensures that hyperweb managers can be easily located and addressed with the HTTP protocol, using the Internet's Domain Name Service and the assigned port number, in the form of `<http://some.domain:4009/>`. The hyperweb manager exposes the Chimera API through the HTTP protocol.

The separation of the hyperweb server from the Chimera server provides the Chimera server with the ability to connect to multiple hyperweb servers and to allow the establishment of links that span multiple hyperwebs.

Chimera's hyperwebs can be accessed and navigated from within Web browsers with the help of cgi-scripts on Chimera-enhanced websites. Java applets can act as Chimera clients to provide access to Chimera's services.

Because Chimera is a link-service based hypermedia system, it does not have control over the actual documents to be viewed or modified, which can result in dangling or invalid links. Chimera does not address the dangling link problems.

While Chimera 2.0 addresses the global distribution consideration through the integration with the Web, it does not address the problem of simultaneous access. In particular, Chimera lacks locking mechanisms to prevent simultaneous changes to particular entries in its hyperwebs. Multiple users manipulating the same anchors or links can overwrite their respective changes. As a link-service based system, Chimera can obviously not provide locking mechanisms for documents.

4 Web-based Hypermedia Systems

4.1 World Wide Web

The World Wide Web is the most widely used distributed hypermedia system in use today. It has, however, serious flaws that severely restrict its usability as a full-fledged hypermedia system.

At the core of the Web is the HTTP protocol [24] which allows arbitrary client programs to request resources from arbitrary server programs, as long as both understand the protocol. Historically, most resources on the Web were presented in the Hypertext Markup Language (HTML) [9, 58], which the client programs generally understand and can render on a screen. Rendering of other formats often requires client program extensions, usually called plugins.

The Web has its focus on the viewing and navigating aspect of hypermedia systems. While authoring using the Web is in principle possible, most Web servers do not implement the necessary protocol support for it or severely restrict its use. Even in the Web servers that do support authoring, there is no provision to provide support for concurrency controls. Resources are not locked, leading to the “lost update” phenomenon, i.e., the last person to update a resource causes the loss of all earlier updates. Newer Web servers provide a means to retrieve a unique value identifying a resource (the ETag), but this is only an advisory for client programs. The client programs are free to ignore this value (and most do.)

The second issue with the Web is the treatment of links. Links on the Web are not first-class objects. Rather, they are embedded in documents. Links on the Web are uni-directional. Client programs often provide a way to get back to the link starting point, but this is not an inherent link feature.

As a result, links on the Web are often out-of-date, resulting in the all-too-familiar “404” error pages. Link maintenance requires the manual task of following each link and observing if the link target is still the intended one.

On the other hand, this second-class nature of links, in particular the lack of a back-link feature, made the growth of the Web possible in the first place. Authors of Web pages are free to create links to any other Web page, without the necessity of changing the linked-to Web page or asking the author of the linked-to Web page to add a back link.

Due to the lack of authoring support in current Web servers, authoring of Web documents requires out-of-band activity. Once finished, documents are usually copied to locations the Web servers can access, so that they can be linked to from other Web documents.

4.2 Frontpage

Frontpage [28] is a protocol to allow remote management of Web servers. It operates with server extensions to provide authoring capabilities.

Frontpage was originally developed by Vermeer Technologies and is now offered by Microsoft. The Frontpage protocol is proprietary, so information about it is scarce (some protocol information can be found in [20].)

Unlike WebDAV, Frontpage does not extend the HTTP protocol. The Frontpage client program uses the existing HTTP POST command to tunnel Frontpage-specific commands to the Frontpage server extensions.

While Frontpage provides remote authoring facilities, it lacks the ability to manage concurrency. Remote resources are not locked, which results in the possibility of lost updates.

The authentication capabilities of Frontpage are limited to the directory level, they don't extend to the file level.

4.3 WebDAV Web Extensions

WebDAV [35] is an extension to the HTTP protocol that forms the basis of the Web. WebDAV tries to overcome some of the problems inherent in the use of the Web as a distributed hypermedia system. Figure 13 provides a high-level overview.

Figure 13: WebDAV Architecture (from [73])

In particular, WebDAV provides authoring facilities for resources on the Web.

WebDAV-compliant Web servers provide the ability for server namespace management. Client programs can use WebDAV to organize resources on WebDAV-compliant servers in hierarchical fashion. WebDAV provides methods to create and manipulate collections of documents.

To support concurrency control, the WebDAV protocol allows WebDAV client programs to lock resources or collections of resources. Such locks can be of the usual write lock variety, but it is also possible to create a so-called shared lock, which acts as an advisory that the lock owner has an interest in the locked documents or collections. To avoid long-term locking of documents, which would be detrimental to collaboration, WebDAV allows the use of time limits for locks.

WebDAV does not, however, provide a history of document changes. The WebDAV designers originally intended to provide versioning support (the “V” in the name still hints at that,) but later deferred this to another specification (DeltaV), in order to allow a timely release of the WebDAV specification.

Finally, WebDAV allows to assign arbitrary attribute-value pairs to documents and collections. This allows authors to specify metadata for documents, for example author names. WebDAV already provides a few fundamental metadata values, e.g., the date of last modification, the length of the document, a human readable name, etc.

4.4 DeltaV Web Extensions

Much like WebDAV is an extension of the HTTP protocol, DeltaV [16] is an extension of the WebDAV protocol. That is, DeltaV supports all the functionality of both the HTTP protocol and the WebDAV protocol.

DeltaV grew out of the deferred versioning part of WebDAV and allows the creation and management of version histories of documents and collections of documents (see figure 14 for a high-level overview.)

Figure 14: DeltaV Architecture (from [75])

It also supports workspace operations usually seen in version control systems such as Subversion ([18]) and configuration management systems like NUCM ([71]). With the locking mechanisms available through WebDAV, this provides strong support for collaborative authoring using the Web as the backbone.

DeltaV also provides additional metadata about the documents and their version histories, to better support client programs for collaborative authoring. Such metadata allows, for example, the association of authors to specific versions of documents.

5 Classification Framework

The hypermedia systems surveyed all provide some form of support for distribution. They each have different restrictions and functionalities that influence the suitability for distributed use. The particular functionalities of these systems is to a large extent influenced by the state of the art at the time the systems were developed, and by the application domains the systems in question target.

Some systems were designed for a collaborative environment from the onset, while others were developed as stand-alone, single user systems, and extended for distributed use later.

The framework developed in this section aims to provide a taxonomy in which to describe and compare properties of hypermedia systems that are relevant for the support of navigation of and authoring in a distributed environment.

Two sections of the framework in particular are important in large-scale distributed hypermedia systems:

- Concurrency control issues, like access control and locking.
These specify who can access or change documents, and help ensure serialization of document access to avoid unintentional overwrites.
- Replication and caching, including consistency of the replicated or cached data.
Replication and caching become a necessity when using Internet-scale distributed systems, to mitigate network latencies and network disruptions. Since multiple replicated copies of documents can get out of sync, distributed hypermedia systems should provide facilities to help the user synchronize documents.

5.1 Links

5.1.1 Embedded Links

Both systems that store links with the documents as well as systems that handle links separately from documents are found. Storing links independently from the documents they connect makes the use of a hypermedia system in a distributed setting easier, since such links can be stored and accessed in a different way than the documents. For example, it is possible to allow each user to have his own personal links, independent of other users' preferences.

5.1.2 Uni-Directional Links

Link maintenance is a big problem in all hypermedia systems. Systems with uni-directional links forego link maintenance for scalability. Systems that allow the linked documents to be changed outside of the hypermedia system also have problems maintaining link consistency. These systems trade familiarity of use of the client programs for link consistency.

5.1.3 N-ary Links

The use of n-ary links, i.e., links that can have multiple endpoints, exacerbates the link consistency concerns. Whenever any of the n linked documents are changed, the links have to be updated, which can put a heavy burden on the hypermedia system and is a detriment to scalability.

5.1.4 Anchor Style

Some systems provide coarse links that only point to documents. Links in finer-grained systems can point to sections within documents, in some cases down to characters in text or pixels in images.

5.2 Document Storage

5.2.1 Hyperbase and Linkserver Combinations

Hypermedia systems can provide storage for all documents available on the system. This facilitates the maintenance of link consistency, particularly in a distributed setting since such hypermedia systems only need to coordinate with their own remote counterparts. It obviously has the drawback that programs unaware of the hypermedia system cannot access the documents stored within the system. To adapt such programs to use the hypermedia system (if at all possible) or to persuade the program vendor to support the hypermedia system can be a costly endeavor. Requiring users to abandon their known tools and learn new specific tools to allow access to the hypermedia system is bound to meet resistance from the users.

On the other end of the spectrum is the exclusive use of a link base. Hypermedia systems that use a link base do not provide storage for the linked documents at all. Users can continue to use their familiar tools to maintain documents. The hypermedia functionality is overlaid on the document by the hypermedia system. The obvious drawback of this approach is that maintaining link consistency may not always be possible. Users can freely manipulate documents in such ways that link endpoints can no longer be found in the documents. Depending on the tool and the extent of integration with the hypermedia system, link maintenance is limited to a best effort approach.

It is also possible for a hypermedia system to provide storage for only some of the documents available on the system. Further, some documents may even be partially or completely generated dynamically, based on some specific user input or system state. An example is the dynamically generated content provided by Webservers through cgi scripts [60] or server-side scripting languages.

5.2.2 Replication and Caching

Of great importance in distributed hypermedia systems is of course how document storage is organized when the storage space can span multiple machines. In particular,