Chimera: Hypertext for Heterogeneous Software Environments

Kenneth M. Anderson, Richard N. Taylor, E. James Whitehead, Jr.*
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92717-3425
FAX +1.714-856-4056
e-mail: {kanderso,taylor,ejw}@ics.uci.edu

ABSTRACT

Emerging software development environments are characterized by heterogeneity: they are composed of diverse object stores, user interfaces, and tools. This paper presents an approach for providing hypertext services in this heterogeneous setting. Central notions of the approach include the following. Anchors are established with respect to interactive *views* of objects, rather than the objects themselves. Composable, *n*-ary links can be established between anchors on different views of objects stored in distinct object bases. Viewers (and objects) may be implemented in different programming languages afforded by a client-server architecture. Multiple, concurrently active viewers enable multimedia hypertext services. The paper describes the approach and presents an architecture which supports it. Experience with the Chimera prototype and its relationship to other systems is described.

Categories and Subject Descriptors:
H.5.1 [Multimedia information systems]
D.2.2 [Software Engineering]: Tools and techniques;
I.7.2 [Document preparation] Hypertext/hypermedia;
General Terms: Design, Experimentation
Additional Key Words and phrases: heterogeneous hypertext, hypertext system architectures, link servers, separation of concerns, software development environments.

1 Introduction

Software development environments (SDEs) are used to develop and maintain a diverse collection of highly interrelated software objects [2, 8, 20, 35]. Large software systems may, for example, consist of multiple versions of requirements specifications, designs, prototypes, code, test information, scripts, documentation, and so on. The connections between these components are many and complex. Establishing and exploring these connections are major tasks of development, program understanding, and maintenance. Researchers have recognized that the size, complexity, and interconnectedness of these systems place a severe cognitive load on software engineers which often leads to errors at high levels of system abstraction, such as requirements and design [1, 17].

It has been suggested that the attributes of hypertext make it an excellent technology for capturing and visualizing relations in a SDE [7]. Providing hypertext capabilities in a SDE allows an engineer to freely associate objects without regard to the type of those objects or where they are stored. These relationships can then be accessed at a later time through a convenient user interface which allows the engineer to easily navigate them [30][1]. Yet while some excellent work has taken place in this area [3, 4, 7, 9, 10, 11, 22, 24, 28, 29, 31], it is clear that no single system to date has effectively addressed all the technical challenges posed by this task.

We believe that the following technical features are among those which need to be present in hypertext systems intended to support SDE activities[2].

Heterogeneous object editor & viewer support. SDEs contain a wide variety of tools for developing and manipulating objects. Different kinds of editors are used for different types of objects. SDEs also increasingly include multiple viewers of single objects, where each viewer presents different aspects of the object, perhaps

*This material is based upon work sponsored by the Advanced Research Projects Agency under Grant Number MDA972-91-J-1010. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

[1]We recognize that some in the software engineering community would argue that more conventional object management systems are the appropriate technology for capturing relations and that free object association should not be allowed. The purpose of this paper is not to argue this point. Our perspective is that given some engineers prefer the hypertext approach, we examine how the approach may be supported.

[2]There is some overlap with our list and the fifteen assumptions listed by Leggett and Schnase when discussing hypermedia-in-the-large [21]. Since we arrived at our list independently, we view this as an indication our approach and contributions have significant value outside the domain of software development environments.

using different depiction styles. It is unlikely that software development teams will give up their favorite tools in exchange for hypertext functionality. Thus a monolithic approach to providing hypertext services to a SDE would be ineffective. Ideally all editors and viewers[3] in an environment should be able to use hypertext services and respond to hypertext events.

Anchors specialized to particular views. Given that different viewers of a single object may present strikingly different depictions, or that one viewer may present a depiction of information synthesized from several separate objects, anchors seem more naturally — or necessarily — associated with views, rather than objects.

Multiple-view, concurrent, and active displays. Since a software developer is typically engaged in examining and changing many different related objects "at once" it is most supportive to provide a system which enables many views to be present simultaneously, where several views may be of the same object, and where actions in views may be autonomous and concurrent.

Links across heterogeneous object managers. SDEs manage such a wide variety of objects, of different legacies, types, and possessing different object management constraints, that large scale SDEs are now beginning to support multiple, heterogeneous object managers. It is nonetheless essential to be able to establish links between objects managed by different repositories.

Action specifications on both anchors and links. Given that many different users, of different abilities and training, may be collaborating on a project using a SDE, it seems useful to provide programmable actions on both anchors and links so that actions could, for example, be determined as a function of who selected an anchor in a particular view, or how a particular link traversal was requested.

Scalable (composable) links. Hierarchy and abstraction are two key tools that engineers employ in undertaking large-scale problems. Hypertext support for SDEs must similarly provide such capabilities for dealing with large, complex, aggregations of information.

***n*-ary links.** Software development often involves situations where several pieces of information jointly represent a single concept or are in some sense "grouped." We claim therefore that hypertext support for SDE applications should provide such capabilities in the form of *n*-ary links.

Multiple contexts. The process of software development involves many stages (often including cycles) where different types of information are more important than others. Often software team members have many different roles both between and within these different stages. As such, hypertext support for SDEs should be able to take into account the current stage of the software development process and enable engineers to quickly access critical information.

This paper describes a set of concepts which satisfy this set of requirements, and a prototype which implements the described concepts. The notion of viewers of objects is at the heart of the conceptualization. We postulate an environment of many types of objects; display or editing of an object requires use of a viewer. Not all viewers are of the same type; how they manage their display is their decision. We have developed a set of interfaces whereby a viewer announces to the hypertext system the anchors it defines for its view of its object(s). These view-specific anchors can then participate in (many) links. Links may be considered objects in their own right, and may thus have viewers associated with them which can define yet additional anchors. These anchors can participate in other links, and in so doing provide hierarchical composition.

This approach brings along with it some limitations and requirements. In order for our techniques and interfaces to be of value, the viewers in the SDE must be programmed to utilize the hypertext system's application program interface (API). The viewers are also responsible for maintaining (over time) the associations they make between the anchors they announce to the hypertext system and the objects or process artifacts (e.g. a button in a window) that they display in their views.

Since heterogeneous environments are most often multilingual and distributed, the generic architecture and our implementation is client-server based and a multilingual remote procedure call (RPC) mechanism (Q [23]) is utilized. Our prototype implementation, Chimera[4], utilizes the Pleiades object management system from the University of Massachusetts [32] for persistence of the server's data structures. To illustrate the concepts and the Chimera system, we discuss an application in which graphical views of a flight simulator's instrument panel are hyperlinked to statements in a requirements document maintained by FrameMaker® [5]. The Chiron user interface development and management system (Chiron) from the University of California, Irvine [33, 34] is used as the graphical viewer of the instrument panel.

The remainder of the paper is organized as follows. The next four sections present the basic concepts, the conceptual architecture, our particular implementation, and our future plans. We then discuss related work and conclude.

[3] From now on we will use the term "viewer" to denote tools capable of visually depicting an object and which may include interactive editing capabilities.

[4] According to Merriam-Webster's 9th Collegiate Dictionary, "an individual, organ, or part consisting of tissues of diverse genetic constitution..."

[5] FrameMaker is a registered trademark of Frame Technology Corporation.

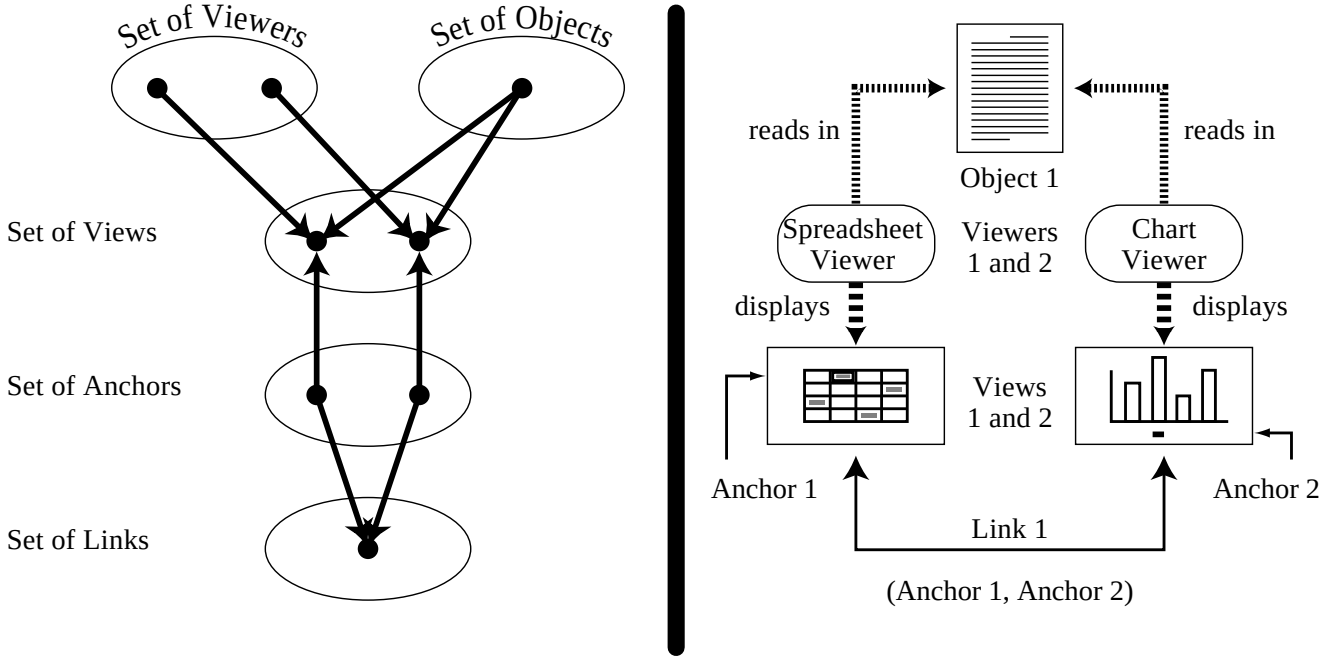

Figure 1: **Chimera Concept Example.** Chimera's hypertext concepts are shown on the left. Two viewers are combined with one object to produce two distinct views. An anchor is added to each view and then combined in one link. On the right, an example hyperweb presents a data file (stored as a file in the operating system) being displayed by two different viewers. One viewer displays the data as a spreadsheet, creating a spreadsheet view of the data file.The other viewer displays the data as a chart, creating a chart view of the same data. The two distinct anchors are indicated by a black box in the spreadsheet, and a black underline in the chart. The anchors are stored in the Chimera database, not in the data file. The two anchors are members of the link. Attribute-value pairs are not indicated to avoid visual clutter.

2 Hypertext Concepts

Chimera has a flexible set of hypertext concepts that map well into the domain of software development environments. Our concepts include objects, viewers, views, anchors, links, attribute-value pairs, and hyperwebs (See Figure 1). In the remainder of this section, we define each concept and provide an example which illustrates how they can be applied in a software development environment.

Objects *Objects* are named, persistent entities whose internal structure is unknown and irrelevant to Chimera.

Viewers *Viewers* are named active entities that display objects. The operations provided by a viewer are specific to the viewer and the type of objects it displays. Typically viewers provide browsing, creation, and editing functionality on objects within their domain.

Views *Views* denote a pair (v, o) where v is a viewer for an object o. Note that an object may be displayed by more than one viewer, and thus participate in multiple views. Views contain anchors.

Anchors *Anchors* are defined and managed by viewers in the context of a view. An anchor tags some portion of a view as an item of interest. Anchors are tailored by a viewer to the particular view of the object being displayed. Unlike hypertext systems which require direct anchor to object mappings [12], with anchors often embedded in the objects themselves [25], Chimera anchors may represent some purely visual component, such as a button or other interface element. Even when creating anchors on the view of an object, the underlying object may be left unmodified, still usable by tools unaware of Chimera's existence.

Links A *link* is a set of anchors. Links relate portions of views. Links are first-class objects in Chimera and a viewer can be constructed to display them. Anchors may be created on these link views and included in other links. In this manner Chimera supports "links to links," an important abstraction which supports construction of large, complex hyperwebs.

Attribute-Value Pairs. Each instance of a Chimera hypertext concept can have an arbitrary number of *attribute-value pairs* associated with it. An attribute-value pair consists of two associated strings where one string con-

tains the attribute's name, the other its value. Attribute-value pairs are commonly used in hypertext systems to provide run-time semantics or behavior for hypertext entities [3]. Example uses of attributes include providing access to an anchor only to the user who created it, or link viewers filtering their displays to show only certain types of links.

Hyperwebs A collection of objects, viewers, views, anchors, and links along with their attributes is a Chimera *hyperweb*. A hyperweb is similar to Leggett's hypermedia [21] and Halasz's hypertext [13].

Example. Several of the Chimera hypertext concepts are demonstrated by a term project from a senior level software engineering project class. For this project, students augmented a flight simulator program distributed with Chiron so that its design and requirements documents, both created with FrameMaker, are efficiently accessed via link traversal.

At the heart of the flight simulator are abstract data types (ADTs) representing the state of the aircraft, including the aircraft's pitch, roll, heading, altitude, and speed. Gauges in the flight simulator's cockpit visualize information from these ADTs as they are updated by the simulator's flight equations. Chiron performs this visualization. In this application the ADTs are considered Chimera objects, Chiron is considered a Chimera viewer, and the gauges that Chiron produces are Chimera views.

The flight simulator requirements and design documents were written using FrameMaker. Both documents are considered Chimera objects. FrameMaker allows users to access and edit these documents, hence FrameMaker is a (Chimera) viewer. The display of the requirements document represents a Chimera view, as does the display of the design document. Anchors are created on the display of the section titles within these documents.

Students added two buttons to each flight simulator gauge, marked "Requirements" and "Design" respectively. These buttons do not visualize any portion of the ADT represented by each gauge, rather they are the visual indication of two Chimera anchors created on the gauge/view. For each gauge, the Requirements anchor was put in a link along with an anchor on the top of the requirements document section describing that gauge. Thus, a single click of the requirements button takes the engineer from the running ("flying") simulator to the requirements for that gauge in the requirements document. The Design anchor was similarly linked to the design document.

The artificial horizon gauge demonstrates the value of the Chimera view concept. This gauge produces a visualization which is a synthesis of information from two objects, the pitch ADT and the roll ADT. This gauge represents a distinct view from the gauges/views that just show the values of the two ADTs separately. Students were able to add their anchor buttons to all three views. Thus while there is no object in the application which directly corresponds to the artificial horizon view, the engineer is nonetheless able to link the gauge to its appropriate requirements document. Moreover since Chimera anchors are defined on a view rather than an object, anchors can be added to multiple simultaneous views of the same object.

3 A Conceptual Architecture

Section 2 outlined Chimera's hypertext concepts and gave a short example mapping them into a reasonably complex software development project. This section sketches a conceptual architecture which supports these concepts. This architecture adopts a client-server approach to providing hypertext services. We term this the Chimera architecture (See Figure 2).

A client-server approach is adopted to help meet the challenges of a heterogeneous SDE in which there are many users. A client-server approach allows multiple users on different machines to access a hyperweb from a dynamically changing set of viewers; hypertext events originate in one viewer and propagate to (potentially many) others via the server. The use of a server supports a multilingual approach where clients can be written in different programming languages, each of them accessing the server through their language-specific API. The use of a server also keeps process and object file sizes down since code to manage a hyperweb is centralized in the server.

We now discuss the components of the Chimera architecture which are the Chimera server, the Chimera clients, the process invoker, and the external tools and systems that populate any SDE.

3.1 Chimera Server

The primary responsibilities of the Chimera server are as follows.

- Provide services which allow clients access to Chimera's hypertext concepts.
- Manage the persistence of a hyperweb. Through the use of an object manager, the Chimera server must map instances of Chimera's hypertext concepts into a persistent store. The Chimera server is not responsible for the persistent storage of client object data.
- Receive, route, and generate hypertext events[6]. The server must also provide a means for clients to register interest in a subset of these events.

[6]The set of hypertext events are left undefined in the conceptual architecture; only their existence is important at this point.

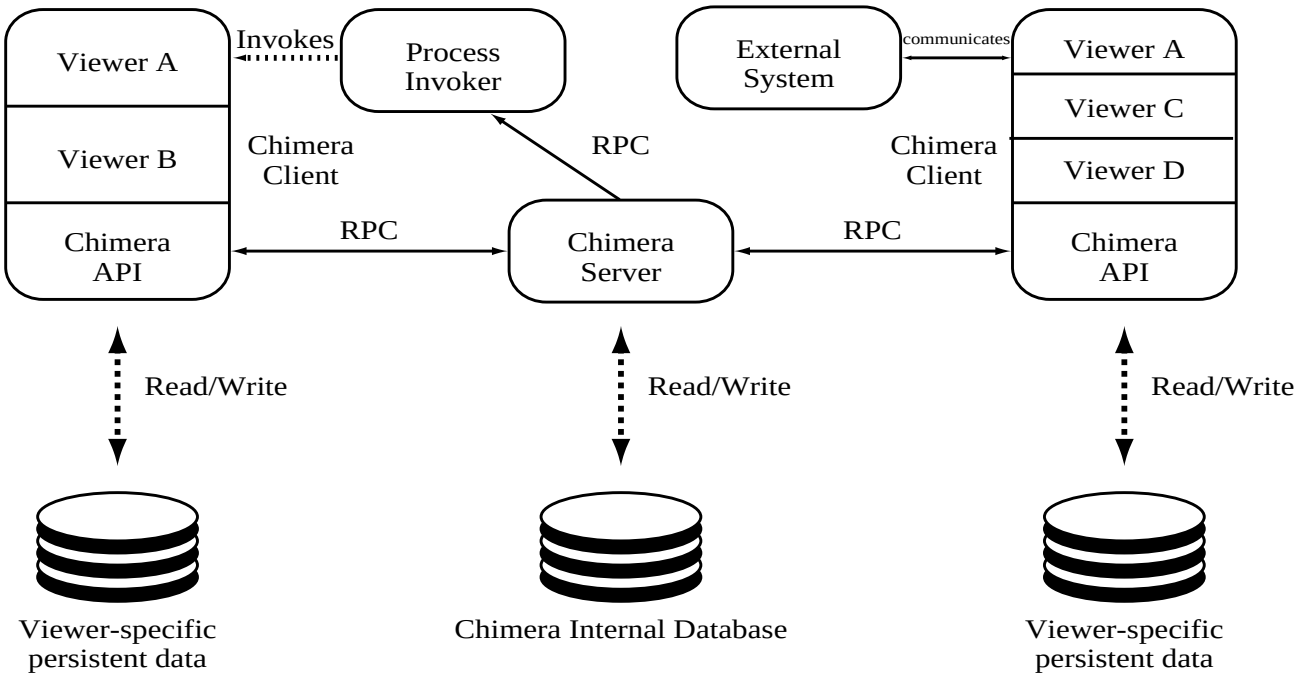

Figure 2: **Example instance of Chimera conceptual architecture.** Chimera clients consisting of one or more viewers access the Chimera server to provide hypertext services to their users. Note that there are no restrictions on the number of clients, the number of viewers per client, and that the same viewer can appear in multiple clients. The process invoker can invoke Chimera clients as directed by the Chimera server. Chimera clients can also access external systems in the environment, such as a graphics server or a sound server. All entities are separate Unix processes and can read/write information to a repository such as a file system or object manager.

- Manage each connected client. For instance, the Chimera server should know what viewers are running, what view each viewer is in, and if the viewer is ready to receive hypertext events.

3.2 Process Invoker

The process invoker is responsible for spinning up Chimera clients. This occurs when the Chimera server determines it needs to send a hypertext event to a viewer that is not running. The process invoker maintains a mapping between viewer names and executable programs[7]. When the Chimera server sends the process invoker a viewer name, the process invoker invokes the executable program via operating system services.

3.3 Chimera Client

A Chimera client includes one or more viewers and the libraries needed to communicate with a Chimera server. Each viewer in a client is responsible for the following.

- definition of the concepts "object" and "view". Each viewer must determine the precise meaning of these concepts in their own context. While this is typically straightforward, ambiguity may arise especially with respect to versions of objects.

- definition of the concept "anchor". This includes identifying what elements of a view can have anchors attached to them, how these anchors are created and deleted, how the presence of an anchor is indicated, and what attributes can be assigned to an anchor. Since each viewer may choose to implement this functionality in different ways, a uniform interaction style cannot be guaranteed[8]. This is one of many issues facing designers of open hypertext systems [4, 15, 25].

- a mapping function from an anchor identifier (received from the Chimera server at the time the anchor is created) into a specific region or object of its display.

[7] This map is supplied and maintained by the administrator of a Chimera hyperweb.

[8] This is potentially troublesome since the user has to remember how this hypertext functionality is invoked for each viewer [9]. This is a design trade-off involving ease-of-use, open systems, and customized interfaces. We believe requiring a single, standard style to be too restrictive: that would prevent many existing viewers from participating in Chimera. On the other hand, it is possible to provide a set of reusable components that developers can utilize which simultaneously simplifies the task of writing viewers and promotes uniform authoring, display, and interaction styles.

- an event handler which will respond to hypertext events from the Chimera server.
- communicating with the Chimera server. This includes registering for hypertext events, indicating its current view (which may change if the viewer is asked by its user to display a different object), accessing the services related to Chimera's hypertext concepts, and letting the Chimera server know that it is ready to receive hypertext events (this provides time for a newly invoked client to initialize before the server sends it any hypertext events).
- providing a mechanism for persistent storage of object data.

3.4 External Systems

Viewers in a Chimera client may directly interface with the user, may require the use of external tools, or may use a user interface management system to present their interface. Chimera does not place any restrictions on what external systems are accessed by its clients or how these clients communicate with their users.

4 An Implementation Architecture

We have constructed a prototype of Chimera to validate the concepts presented in Section 2 and the conceptual architecture of Section 3. This prototype has been used to support the example described in Section 2 (as well as many other applications). Our prototype was constructed on Sun SPARCstations under the Unix operating system using the Ada and C programming languages. In this section, we describe the design choices that we made in mapping the conceptual architecture into a working prototype and discuss other issues concerning the prototype such as performance and size statistics.

4.1 Chimera Server

The Chimera server realizes and in some cases exceeds the goals set out for it in Section 3.1. We discuss each goal in roughly the order they were presented in Section 3.1 in the paragraphs below.

At the heart of the Chimera server lies a set of Ada packages which implement Chimera's hypertext concepts as ADTs. The Chimera server coordinates access to this set of ADTs by responding to remote procedure calls made by Chimera clients. These clients are invoked by end-users (or the process-invoker) and typically contain one viewer (although multiple viewers per client is allowable). The Chimera server can handle multiple clients run by multiple users at the same time.

Filtering information is maintained for each viewer connected to the Chimera server. Anchors and links can be filtered such that different sets of these concepts can be provided to different users for the same view. Users can select the level of filtering received if the viewer provides an interface to do so[9]. The default filtering level is none, i.e., all anchors and links for a particular view are accessible. The other filtering level is user_only, such that only those anchors and links created by a user on a particular view are accessible. This functionality allows Chimera to provide support for multiple contexts in a single hyperweb. Eventually we plan to implement a system of ownership and permissions modeled after Unix's file permissions. Thus, only those anchors and links which are readable by a user will be accessible for a particular view. We will then extend Chimera's support for multiple contexts by allowing a user to have different roles and thus gain access to different sets of anchors and links.

An active link is maintained for each user connected to the Chimera server. An active link is a mechanism provided by the Chimera server to allow modeless link creation. In typical hypertext systems, link creation occurs as a mode. The user indicates that a new link is desired, adds (typically two) anchors to this link and then continues working. In Chimera, a user can create several empty links, select one of these links to be active, and then add anchors to this active link at any time thereafter. The user can also at any time select another link to be the active link. Note that viewers are not required to use the active link mechanism. It is provided only as a convenience function in an attempt to foster common interaction styles between Chimera viewers. A viewer can forgo the active link mechanism, register its own links, and modify them independent of other viewers.

Hyperwebs are mapped into Unix directories where the Chimera server stores temporary run-time files along with the persistent state of its ADTs. The ADTs save their state information with the Pleiades object management system. Users select which hyperweb they want to access through a resource file (.chimerarc) located in their home directories which the Chimera server reads on start-up.

The Chimera server currently supports fourteen hypertext events and clients can register or deregister interest in any of them[10] (See Figure 3). The events range from reporting changes to the hyperweb, such as the addition or deletion of hypertext concepts, to link traversal events and active link events.

Finally, clients provide a variety of information about themselves to the server. This information includes whether each viewer in each client is ready to receive hypertext events, what hypertext events each viewer is interested in, and what view(s) each viewer is in. The server also knows how each

[9] Since Chimera is an open hypertext system, this functionality can not be guaranteed across all viewers. It will only exist if the viewer developer decides to include it in a particular viewer.

[10] A client is assumed to have no interest in any event when it first contacts the Chimera server.

Event Name	Associated Information
Server Disconnect	None
New Viewer	Viewer Identifier
Delete Viewer	Viewer Identifier
New Object	Object Identifier
Delete Object	Object Identifier
New View	View Identifier
Delete View	View Identifier
New Anchor	Anchor Identifier
Delete Anchor	Anchor Identifier
New Link	Link Identifier
Delete Link	Link Identifier
Modify Link	Link Identifier
Link Traversal	Anchor Identifier
Active Link	Link Identifier

Figure 3: **Chimera's Hypertext Events**

viewer would like to be invoked via an invocation policy attribute associated with each viewer. This policy is used when the server must send a link traversal event to a view not currently displayed by any connected viewer. A viewer with an invocation policy of "Once_Only" is only invoked once and all subsequent link traversal events are sent to it. This is for viewers which can display multiple views, perhaps by opening a new window for each one or by closing the previous view before opening the new one. An alternative invocation policy is "Every_Time" which causes the server to invoke (via the process invoker) a new instance of a viewer each time a link traversal occurs to an undisplayed view.

4.2 Process Invoker

The process invoker is an RPC server to which the Chimera server sends messages when it wants a particular viewer invoked. The Chimera server sends a viewer name which the process invoker maps into an executable program or shell script which it then invokes. This invocation is currently handled by having the process invoker use the Unix `fork` command to start a child process. This child process calls the Unix `execvp` command which replaces the child process with the specified executable program which then begins to run. The map that the process invoker uses to determine which executable program to run is read in once at start-up and is stored in the hyperweb that the user is accessing. The information in the map file must be entered manually via a text editor in the current implementation. (The limitations implicit in this will be fixed in future implementations. They can be overcome by providing tools to manage a process invoker's map file and altering the process invoker to detect changes to its map file at run-time.)

4.3 Chimera Client

A major benefit of the client-server architecture of Chimera is that its clients may be written in more than one language. An API to the Chimera server for a particular language is required before that language can be used to construct Chimera clients. An advantage of the API approach is that low-level RPC details of passing messages to the Chimera server are completely hidden from the clients which use the API. Instead clients invoke subprograms like `register_anchor` or `traverse_link` and the API converts these subprograms calls (and their parameters) into the appropriate RPC messages and ships them to the Chimera server. The API also passes back any return values from the server to the calling client. This conversion is straight forward and includes creating a new RPC buffer, marshaling the parameters into the buffer, making the actual RPC call, retrieving any return values from the buffer, and deallocating the RPC buffer.

Chimera supports clients written in Ada and C with APIs for both languages. Several clients have been written using each of these APIs. A C++ API is in the works but, as of this writing, is not yet complete. In addition, engineers at IBM Owego, New York, developed *tcl* bindings to Q with the result being that they can send RPC messages to the Chimera server. These *tcl* bindings do not represent a complete API to Chimera, however, since they cannot yet receive hypertext events from the Chimera server. Work is now in progress to develop a *tcl* API which is built on top of the C API.

The Ada API creates two Ada tasks per viewer. One task handles sending messages to the Chimera server; the second task handles receiving hypertext events from the server. These tasks operate independently and maintain separate Unix sockets. This allows multiple connections to the Chimera server within a single Unix process. The Ada API has proven to work successfully with other client-server systems, the most notable being a simultaneous connection by one viewer to a Chimera server, a Chiron server, and a sound server.

The C API allows C programs to use Chimera services within a single Unix process. Two sockets are maintained by the C API, requiring application writers to take responsibility for the scheduling of message transmission and event reception. Since many programs using the C API will also use X Windows to produce their user interface, support was added to receive Chimera events from within an Xt event loop. Chimera events are handled using a callback mechanism. To date, four separate C programs have been written which are simultaneously Chimera and X clients within a single Unix process.

4.4 External Systems

As of this writing, seven viewers for several media types have been integrated with the Chimera system. These viewers are briefly described below.

FrameMaker The FrameMaker program was integrated into the Chimera system *without modifying its executable image*. A wrapper program translates between Chimera hypertext concepts and FrameMaker's built-in hypertext concepts. Custom FrameMaker macros were written for anchor creation so link traversal messages could be received by the wrapper.

xvi The public-domain vi-clone 'xvi' was integrated with Chimera by a group of senior students as a class project for a software engineering project course. All existing vi editing functions work normally, with hypertext functionality accessed through a graphical control panel written using the Motif toolkit. Chimera services are accessed through the C API. Link traversals to xvi cause a different buffer to be opened for each new file referenced in the link.

Flight Simulator As described earlier, the flight simulator is a simulation of an aircraft. Written in Ada using Chiron, the flight simulator features each gauge in a separate thread of control. The Ada API easily allows multiple threads of hypertext functionality to operate independently within a single Unix process.

mpeg An ambitious term project for the software engineering project course, the mpeg viewer allows anchors to be defined on mpeg movies. Anchors may be defined on sections of each frame, and may have a duration from one frame to the entire movie. Authoring support allows anchors to be defined on a frame, then copied from frame to frame. Authors may then single-step through the movie and adjust anchors on individual frames for the best fit. Anchors may be added to the active link from any frame. The mpeg viewer was created by extending a public-domain mpeg player using the C API.

xgif Another result of the same course, the xgif viewer allows anchors to be defined on sections of a GIF[11] image. A public-domain GIF viewer was augmented using the C API to display a hypertext control panel written using the Motif toolkit. The control panel varies depending on whether the GIF viewer is in author mode or view mode. Link traversals to xgif cause a new xgif process to be invoked to view the linked-to GIF file.

Sound Player The sound player presents a list of sounds to which anchors are attached. Link traversals to a particular anchor causes the sound player to play the associated sound. The Sound Player uses Chiron for its user interface, the Ada API for its hypertext functionality, and a Sun sound server for sound capability, making it simultaneously a client of three separate servers.

Button The button is a simple viewer which displays a window containing only a button. An anchor is then associated with this button. This anchor may be a member of only one link (a restriction enforced by this particular viewer), and can be used as a placeholder for a section of a document. This viewer is noteworthy since its view is not associated with any underlying object, yet it can participate in a Chimera hyperweb.

[11] The Graphics Interchange Format © is the Copyright property of CompuServe Incorporated. GIF ℠ is a Service Mark property of CompuServe Incorporated.

4.5 Metrics

The Chimera API consists of approximately 90 entry points. The Ada API library is 296 kilobytes (K) in size. The C API library is 62 K. The Chimera server is 2.3 megabytes in size and uses approximately 5 megabytes of memory when executing.

Our metrics for the performance of Chimera is anecdotal. From a user's perspective, a link traversal between two running viewers occurs instantaneously. If a link traversal leads to a non-running viewer, there is a noticeable delay while Unix spins up the new process. After the newly invoked viewer has initialized the completion of the link traversal occurs quickly.

We conducted one performance experiment with the integrated FrameMaker client. We set up a link traversal between two FrameMaker documents using Chimera and FrameMaker's own internal hypertext functionality. The user noticed no difference between the time it took to complete either traversal despite the fact that the Chimera link traversal involved the passing of RPC messages across four Unix processes while the FrameMaker link traversal was handled completely within FrameMaker.

5 Future Work

5.1 Versioning

Version control mechanisms are very important for any hypertext system that wishes to support software engineering activities in a non-trivial fashion [14, 18]. Chimera is no exception, and a mechanism for versioning is a research priority. A brief discussion of our preliminary approach is given below.

Since Chimera intentionally offers no support for storing application objects within its hypertext data repository, version control responsibility must be shared between Chimera and its client applications. As an example, Chimera will undoubtedly be used to create relations between source code files. Version control responsibility for the files alone will rest with an existing revision control system such as RCS [36]. Responsibility for versioning the relations between the files will reside with Chimera.

Each concept within Chimera shall be capable of multiple versions. This raises significant issues of consistency maintenance when a concept instance is changed. For example,

when an anchor is deleted, it must also be removed from any links to which it belongs, requiring a new version of those links. Another issue is consistency maintenance between versions maintained by an external versioning system and versions maintained by Chimera. Resolving this issue and providing a mechanism for flexible multi-concept version naming requires a new first-class concept in Chimera, the *configuration*. A configuration will contain a snapshot of the current versions of a hyperweb subset. External object versions can then be mapped to a configuration.

It is anticipated that parallel version paths will be supported by Chimera using the first-class configuration concept. While explicit mechanisms will need to exist to support the creation of new branches, once a new branch has been created its use should be mostly transparent. Merging of paths will require the use of a special utility which will resolve conflicts, prompting the user for guidance, if necessary.

5.2 Collaborative Hyperweb Construction

Chimera does not currently support the collaborative building of linked hyperwebs in real-time by multiple users where each user is made aware of other user actions[12]. This is due to limitations in the conceptual architecture that must be addressed, as well as the current implementation's restriction of one Chimera server per hyperweb. The major limitation of the current conceptual architecture is that the Chimera server must maintain the data stored in a hyperweb along with managing any Chimera clients that connect to it. What is needed is a separation of concerns where a new component, the *hyperweb server*, is added to the conceptual architecture. This new component relieves the Chimera server from its data management activities. This involves moving the ADTs which implement Chimera's hypertext concepts from the Chimera server to the hyperweb server. The Chimera server component is then free to concentrate on supporting client access to multiple hyperweb servers (and thus multiple hyperwebs). We postulate an environment in which there is one hyperweb server per hyperweb with multiple Chimera servers managing collaborative user access in real-time to these hyperweb servers. The two sets of servers would need to work together to handle links created between hyperwebs and also to ensure that notification of changes made to the information space is propagated to all connected users. We also anticipate that the Chimera server will support access to distributed objects by employing a Universal Resource Locater (URL) style object naming mechanism and an existing transfer protocol to access remote objects.

[12] As of this writing, Chimera supports the collaborative building, in a limited manner, of a single hyperweb, since all the viewers participating in the collaboration can register for the appropriate hypertext events from the Chimera server managing that particular hyperweb. It is limited in the fact that two or more users cannot modify the same link at the same time and the notification of other user's actions occurs after those actions have taken place.

6 Related Work

There has been substantial evolution of hypertext functionality during the last decade and several significant efforts to apply hypertext to the software development problem (or similar). The systems described below are discussed in chronological order of appearance and were chosen either for their historical importance or because of their close relation to and impact upon the design of Chimera.

6.1 The Dexter Hypertext Reference Model

The Dexter Hypertext Reference Model (Dexter) is a framework for comparing hypertext systems developed as the result of two NIST workshops in 1988 [13]. It attempts to provide a standard hypertext terminology and also describe the important abstractions found in hypertext systems. At least one system has been developed based on Dexter (DeVise [12]) and Dexter itself has also been extended (AHM [16]).

Dexter defines three layers: the run-time layer, the storage layer, and the within-component layer. The storage layer sits in the middle and interfaces with the run-time layer via presentation specifications, and the within-component layer via anchors. The storage layer creates a network of components of three types: atomic, composite, and link. Atomic components contain data stored in the within-component layer and presented by tools in the run-time layer. In addition, atomic components contain anchors which index directly into the data and allow the data to be linked. Composite components allow scalable hyperwebs to be constructed. Link components contain specifiers which describe how atomic and composite components are linked together.

Chimera cannot be completely modeled in Dexter. For instance, Chimera can handle the presence of links with zero or one anchors (dangling links). Dexter's intolerance for such constructs has been widely criticized [12, 16, 21]. Also Chimera's notion of a view cannot be adequately modeled by a composite component, because a Chimera view contains information about the object being viewed and the viewer used to create the view. A composite component on the other hand only contains references to atomic components which only contain data. In Dexter, anchors are created directly on the data referenced by atomic components, whereas in Chimera, anchors are created on views not on the objects (i.e. data) themselves. This additional level of abstraction is not possible in Dexter. In fact, we believe that Dexter cannot model the case where the same object (atomic component) is displayed differently by two or more viewers with each viewer accessing a different set of anchors and links. Chimera's view concept handles this example easily while in Dexter there would be no way to specify it. Finally, a viewer is free to define its anchors with respect to anything in its view including objects which exist only at run-time e.g.

a graphical widget. We believe that Dexter would be unable to specify this type of anchor, since a graphical widget lies in the domain of Dexter's "presentation specifications" and anchors can not be created in Dexter with respect to these specifications.

6.2 Virtual Notebook System

The Virtual Notebook System (VNS) was built at the Baylor College of Medicine to support collaborative biomedical research via distributed hypertext services in a heterogeneous computing environment [28]. VNS is implemented by a set of work group servers (WGSs) distributed throughout a network. Each WGS is used to store text, graphics, and link information. Users typically store all their data with the WGS on their local machine but can also access information stored on another machine. The VNS Gatekeeper is used to integrate external tools with VNS, whereby information from these tools is copied and stored in a WGS. One interesting aspect of VNS is that while users may share data, they do not share links. Thus two users can see the same page but view different links. Link information for each user is stored separately from the data that makes up a page. After a page is constructed dynamically, a user's link information for that page is retrieved and displayed. Chimera is able to provide this functionality with its link filtering and can go one step further with its anchor filtering which allows users to see different anchors on a shared view.

VNS, in contrast to Chimera, requires that all system information be stored in a database under its control. Integration in VNS is concerned with providing the ability to copy information out of an external tool and into a VNS database. At that point, the external tool is taken out of the loop; VNS handles the display of the data from then on. Integration in Chimera is only concerned with getting a viewer to communicate with the Chimera server. Chimera makes no attempt to display a viewer's objects.

6.3 Sun's Link Service

Sun's Link Service (LS) was a commercial product which defined a protocol for an extensible and loosely coupled hypertext system [25]. An application integrates with the LS by loading in a library which implements the protocol. This library allows communication with the LS control process which facilitates communication between link-aware applications. Applications provide call-back procedures to the LS so that they can receive link-related messages. Links are binary and are stored in a database managed by the LS control process.

Chimera and the Link Service differ in three aspects. The LS promotes applications creating anchors on the underlying object via its data model. Chimera's anchors are created on a view of an object, not the object itself. This allows a Chimera viewer to store anchor information separately from the object (or objects) to which it refers. Links are hidden in the LS; that is, an application cannot retrieve links and manipulate them. This is not the case with Chimera, where links can be retrieved by an appropriate viewer and displayed in a variety of ways. This enables the creation of hierarchical hyperwebs. Finally Chimera's links are *n*-ary.

6.4 Hyperform

Hyperform is an approach to creating flexible hyperbase support based on the notion of extensibility. It is joint work of the University of Aalborg and the Hypermedia Research Laboratory [37]. The Hyperform system is a simple framework which provides several built-in classes that perform basic hyperbase features such as concurrency control, notification control, access control, version control, and search and query. A user of Hyperform takes this initial framework and extends the built-in classes to produce a hyperbase system tailored for the hypermedia applications in the user's environment. The object-oriented language used to implement the Hyperform server is the key to its extensibility and allows for new data objects and methods to be added at runtime which in turns produces an environment conducive to rapid prototyping.

The Chimera approach differs greatly from the Hyperform approach. The authors of Hyperform assert that a fixed hypermedia model hinders the development of new tools by forcing developers to make compromises in their applications to match the existing model [37]. Thus, the Hyperform approach has an implicit assumption that developers of hypermedia systems want to develop one system for a certain set of tools and another system for a different set of tools which require a different hypermedia model. After several iterations of this approach a user ends up with several hypermedia systems which we believe could be incompatible with each other. Thus, objects managed by one of these systems cannot be linked to objects in another of these systems, and presumably the effort required to update applications which use one data model to the other data model is non-trivial. This implicit assumption is simply not tenable in a software development environment which is already extremely heterogeneous. It seems counter-productive to introduce more diversity in such an environment with the addition of multiple potentially incompatible hypermedia systems. There is another assumption here that all of the objects in a hypermedia system are stored in a single database. This assumption is not valid in a software development environment in which multiple heterogeneous object management systems can exist and yet it is desirable to link objects stored in these diverse object repositories.

Thus Chimera provides a flexible hypermedia model which was developed specifically for the needs of tools in a software development environment. This greatly reduces the chance that a specific tool cannot use the model provided by Chimera

and not be able to participate in a Chimera hyperweb. In fact Chimera's hypermedia model was designed with the assumption that Chimera would enter an environment with many pre-existing tools which would eventually be integrated with it. Thus the model had to be as flexible as possible.

6.5 Microcosm

Microcosm is an open hypertext system developed at the University of Southampton [5, 6, 15]. It is a link service that attempts to keep all aspects of the system such as the hypertext model, the messages passed from applications to Microcosm, and Microcosm's response to such messages open and tailorable. Microcosm-aware applications send selections and action pairs to the Microcosm Document Control System (DCS). The DCS sends the message through a chain of filters which act on the message by blocking it or passing it on, perhaps modifying it along the way. A special type of filter is a linkbase which upon finding the source of a link in the message attaches the destination of the link to the message. The message emerges from the filter chain into the link dispatcher which presents to the user any actions contained in the resulting message. Microcosm can integrate non-aware Microcosm viewers through the use of a shared clipboard. However this is a worst-case situation that is rarely used as most applications that users want to use with Microcosm contain the hooks needed to integrate with Microcosm. Microcosm has been applied to situations involving several hundred megabytes of information and can handle multiple sets of links over the same set of data (by swapping in or out various linkbases in the filter chain).

There are three distinctions between Chimera and Microcosm.

1. Microcosm uses a message-based API while Chimera uses a multi-lingual programmer's API. Microcosm messages are in a tagged ASCII format. Filters act on tags in the ASCII message that they recognize and ignore all other tags. In addition, each filter can introduce any tag and its associated data into any message. In contrast, the details of Chimera's message format are hidden from Chimera clients by the Chimera API and the Chimera server by a message ADT. This use of abstraction allows the Chimera developers to freely change the format of the messages without affecting the rest of the system. This allows the message format to be as simple (ASCII text) or as complex (an Ada variant record) as needed to effectively support the semantics of the Chimera API. This freedom to change message formats is not available in Microcosm but their approach can potentially integrate more tools into their system since they do not have to modify each participating tool to use a programmer's API. Chimera's approach to integrating non-aware viewers is to create a wrapper or proxy program which uses both the Chimera API and whatever mechanism the non-aware viewer has to communicate with external systems.

2. All Microcosm-aware applications must provide all hypertext functionality via a menu-based interface. While this promotes a common interaction style between viewers, it may also prevent some applications (such as those without a graphical user interface) in participating in the system. Chimera does not prescribe or restrict a viewer's interface in any way, with the idea that in a software development environment, developers will tolerate inconsistency in interface in return for using a powerful tool within a Chimera hyperweb.

3. Microcosm has no analogous concept for Chimera's view concept. Each document in Microcosm has a user-defined physical type. Each physical type is associated with one viewer. When a particular document is the destination of a link traversal, Microcosm invokes the associated viewer on the specified document. Chimera's view concept is independent of where a particular data element is stored. This allows Chimera greater flexibility in handling multiple views of a single object, and also handles easily the case where a single view consists of data accessed from multiple sources.

6.6 System Prototype 0, 1, 2, and 3

The Hypermedia Research Laboratory (HRL) at Texas A&M University has engaged in the building of several hypermedia systems (SP 0-3) since 1991 [19, 21]. At the same time, the HRL has also built a series of hyperbase systems (HB 0-3) to support the data storage requirements of these hypermedia systems [27, 26]. SP3 defines a flexible hypertext model. Applications manage components which have persistent selections created upon them. These persistent selections are attached to anchors which are then associated with links. These links create association sets that are modeled in the underlying hyperbase.

This matches Chimera's hypertext concepts fairly well. The only difference being that multiple persistent selections can be associated with a single anchor. In Chimera, if the viewer wanted to associate multiple regions of its view to one anchor it could do so, but this would necessarily make the mapping function between anchors and regions of the view more complex. Typically Chimera viewers map one region per anchor, which would be the equivalent of mapping one persistent selection to anchor in SP3.

A distinctive feature of the HRL work is that anchors and links are first-class executing programs that can provide a wide range of run-time semantics. It also allows user interaction with hypertext services to be handled by the anchor and link processes taking this burden off of the client applications. (This is handled by virtue of the fact that anchors and links are supplied as classes which have a default set of

behaviors which applications can subclass and modify as desired.) SP3 and HB3 provide a sophisticated environment that represents a first attempt at providing hypermedia-in-the-large.

The differences between Chimera and the HRL work are many. For instance, Chimera is not yet at a state where it can support multi-user collaboration on a shared hyperweb. A few additional differences are outlined below.

- Chimera has taken a different approach with respect to anchors and links. They are objects managed by viewers and the Chimera server respectively as opposed to being first-class processes. This allows for anchors to be specifically tailored to a viewer while placing a burden on viewer developers to implement this functionality for each new viewer. It allows links to be handled in a consistent manner at the price of implementing special link behaviors in the Chimera server.
- SP3 requires applications to use its object manager (HB3) to store application data; this allows SP3 to support versioning of application data and hypertext information. Chimera, in contrast, places no such restrictions on its viewers requiring viewer cooperation to provide reliable versioning.
- Chimera associates anchors with views, while SP3 associates anchors with persistent selections which refer directly to an application's data. This enables Chimera, when combined with a viewer mechanism such as Chiron, to provide greater flexibility in displaying an anchor, supporting the ability of several viewers (concurrently) providing different views of the same object, where the anchors and their presentation are view-specific (and this all separated by Chiron from any application code). This is similar, though, to SP3's notion of a context. In SP3, depending on the context, different sets of anchors and links will be made available to an application displaying the object. Contexts are handled in Chimera through a combined use of attribute-value pairs and the filtering of anchors and links by the Chimera server for a particular view.
- SP3 links are not "ordinary" objects and thus anchors cannot be defined upon them and thereby participate in hierarchical webs. Thus Chimera appears to have a scalability advantage.

7 Conclusions

In conclusion, Chimera makes a variety of contributions to software engineering environments and to hypertext technology, including the simultaneous satisfaction of the requirements described in the introduction. The essence of the contributions are key concepts and separations of concerns, provision of an effective set of server capabilities, and the demonstrated ability to simultaneously satisfy a wide variety of objectives in a single system.

The concepts and separations include view-specific anchors and separating object and anchor persistency from link persistency. Viewers define anchors and a hypertext server has responsibility for management of the links. Allowing viewers to define anchors permits a variety of types of anchors to be defined, and they may be implemented in non-invasive ways. Neither the database(s) of objects nor the user interface are part of Chimera or its concepts. The concepts are defined in a media-independent manner and such that scalability is supported.

The Chimera server interface supports multiple, concurrent clients written in multiple programming languages and demonstrates that commercial "black-box" tools can be integrated (provided they support minimal interprocess communication).

We have built a prototype system, Chimera, to validate both the concepts and architecture. In addition, we believe our approach has value outside of the SDE domain and can aid such tasks as ethnographic studies and the building of digital libraries.

Some open, and potentially troublesome, issues with this approach exist. Since viewers define anchors, and viewers may be heterogeneous, a lack of consistent user interface to the hypertext is more likely to occur than not. More troublesome from the SDE point of view, however, is the observation that the relations indicated by the hypertext links are in addition to whatever relations are maintained by the environment's object managers. This may yield a number of problems, including maintaining consistency in the face of change to the object stores. On the other hand it does not seem realistic to assume the existence of a single object manager which is responsible for maintaining all relations in an environment, whether they originate from quick, dynamic, and user-discretionary hypertext link creation, or careful specification and design of a complex project's master database of strongly-typed artifacts. The broad research issue, in a heterogeneous world, is determining how to maintain consistency between various relation/link managers. For a near-term partial solution, one approach we intend to pursue is the automatic creation (and maintenance) of hyperlinks from object manager relations; in such a case hypertext style navigation of an OM store would be enabled. It seems much more problematic to attempt to go the other direction, however (from hyperlinks to OM relations), because of the limitations of current OM systems. Additional key research activities include determining appropriate mechanisms for supporting access controls (so, e.g., a project's on-line personnel records are not accessible by those unauthorized), versioning, and support for collaborative hyperweb creation.

8 Acknowledgments

The authors would like to acknowledge Jonathan Grudin, Rebecca Grinter, Leysia Palen, and Hadar Ziv for reading an early draft of this paper and providing comments, the reviewers for their helpful suggestions, and Hugh Davis and Wendy Hall for providing detailed information about the Microcosm link service. In addition, the authors' gratitude is extended to the students in the software engineering project course whose experiences fine-tuned Chimera and helped explore issues of integration with existing software.

References

[1] V. R. Basili and B. T. Perricone. Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, 27(1):42–52, January 1984.

[2] G. Boudier, F. Gallo, R. Minot, and I. Thomas. An overview of pcte and pcte+. *SIGSOFT Software Engineering Notes*, 13(5), November 1988.

[3] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987.

[4] M. L. Creech, D. F. Freeze, and M. L. Gris. Using Hypertext in Selecting Reusable Software Components. In *Proceedings of Hypertext'91*, San Antonio, Texas, December 1991.

[5] H. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins. Towards an Integrated Information Environment with Open Hypermedia Systems. In *Proceedings of the ACM Conference on Hypertext*, Milano, Italy, November 1992.

[6] H. Davis, S. Knight, and W. Hall. Light Hypermedia Link Services: A Study of Third Party Application Integration. In *Proceedings of the ACM Conference on Hypertext*, Edinburgh, Scotland, September 1994.

[7] N. Delisle and M. Schwartz. Neptune: A hypertext system for CAD applications. In *Proceedings of the ACM SIGMOD'86*, pages 132–142, Washington, DC, May 1986.

[8] C. Fernström, K.-H. Närfelt, and L. Ohlsson. Software factory principles, architecture, and experiments. *IEEE Software*, 9(2):36–44, March 1992.

[9] J. C. Ferrans, D. W. Hurst, M. A. Sennett, B. M. Covnot, W. Ji, P. Kajka, and W. Ouyang. HyperWeb: A Framework for Hypermedia-Based Environments. In *Proceedings of ACM SIGSOFT '92: Fifth Symposium on Software Development Environments*, Washington D.C., December 1992.

[10] P. K. Garg and W. Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, 7(3):90–98, May 1990.

[11] F. Garzotto, P. Paolini, and D. Schwabe. HDM - A Model for the Design of Hypertext Applications. In *Proceedings of Hypertext'89*, Pittsburgh, Pennsylvania, November 1989.

[12] K. Grønbaek and R. H. Trigg. Design Issues for a Dexter-based Hypermedia System. *Communications of the ACM*, 37(2):40–49, February 1994.

[13] F. Halasz and S. Mayer. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, February 1994.

[14] F. G. Halasz. Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[15] W. Hall, G. Hill, and H. Davis. The Microcosm Link Service: A Technical Briefing. In *Proceedings of Hypertext'93*, Seattle, Washington, November 1993.

[16] L. Hardman, D. C. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *Communications of the ACM*, 37(2):50–62, February 1994.

[17] K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, SE-6(1):2–13, January 1980.

[18] D. L. Hicks, J. J. Leggett, and J. L. Schnase. Version Control in Hypertext Systems. Report TAMU HRL-91-004, Texas A&M University, July 1991.

[19] C. J. Kacmar and J. J. Leggett. PROXHY : A Process-Oriented Extensible Hypertext Architecture. *ACM Transactions on Information Systems*, 9(4):399–419, October 1991.

[20] R. Kadia. Issues encountered in building a flexible software development environment: Lessons learned from the Arcadia project. In *Proceedings of ACM SIGSOFT '92: Fifth Symposium on Software Development Environments*, Tyson's Corner, Virginia, December 1992.

[21] J. J. Leggett and J. L. Schnase. Viewing Dexter with Open Eyes. *Communications of the ACM*, 37(2):77–86, February 1994.

[22] D. Lucarella, S. Parisotto, and A. Zanzi. MORE: Multimedia Object Retrieval Environment. In *Proceedings of Hypertext'93*, Seattle, Washington, November 1993.

[23] M. J. Maybee, D. H. Heimbinger, D. L. Levine, and L. J. Osterweil. Q: A multi-lingual interprocess communications system for software environment implementation. Submitted for publication, 1992.

[24] J. Nielsen. *Hypertext and Hypermedia*. Academic Press, Inc., San Diego, California, 1990.

[25] A. Pearl. Sun's Link Service: A Protocol for Open Linking. In *Proceedings of Hypertext'89*, Pittsburgh, Pennsylvania, November 1989.

[26] J. L. Schnase. *HB2: A Hyperbase Management System for Open, Distributed Hypermedia System Architectures*. PhD thesis, Texas A&M University, College Station, Texas, August 1992.

[27] J. L. Schnase, J. J. Leggett, and D. L. Hicks. HB1: Initial Design and Implementation of a Hyperbase Management System. Technical Report TAMU-HRL 91-003, Hypertext Research Lab, Texas A&M University, October 1991.

[28] F. M. Shipman, III, R. J. Chaney, and G. A. Gorry. Distributed Hypertext for Collaborative Research: The Virtual Notebook System. In *Proceedings of Hypertext'89*, Pittsburgh, Pennsylvania, November 1989.

[29] J. B. Smith and F. D. Smith. ABC: A Hypermedia System for Artifact-Based Collaboration. In *Proceedings of Hypertext'91*, San Antonio, Texas, December 1991.

[30] D. Steinberg and H. Ziv. Software Visualization and Yosemite National Park. In *Proceedings of the Twenty-Fifth Annual Hawaii International Conference on System Sciences*, January 1992.

[31] N. Streitz, J. Haake, J. Hannemann, A. Lemke, W. Schuler, H. Schütt, and M. Thüring. SEPIA: A Cooperative Hypermedia Authoring Environment. In *Proceedings of the ACM Conference on Hypertext*, Milano, Italy, November 1992.

[32] P. Tarr and L. A. Clarke. Pleiades: An Object Management System for Software Engineering Environments. In *ACM SIGSOFT '93: Proceedings of the Symposium on the Foundations of Software Engineering*, Los Angeles, California, December 1993.

[33] R. N. Taylor and G. F. Johnson. Separations of concerns in the Chiron-1 user interface development and management system. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 367–374, Amsterdam, April 1993. Association for Computing Machinery.

[34] R. N. Taylor, K. A. Nies, G. A. Bolcer, C. A. MacFarlane, G. F. Johnson, and K. M. Anderson. Separations of concerns in the Chiron-1 user interface development and management system. UCI–ICS Technical Report TR-94-12, Department of Information and Computer Science, University of California, Irvine, March 1994.

[35] I. Thomas. Tool Integration in the Pact Environment. In *Proceedings of the Eleventh International Conference on Software Engineering*, Pittsburgh, PA, May 1989.

[36] W. F. Tichy. Design, implementation, and evaluation of a revision control system. In *Proceedings of the Sixth International Conference on Software Engineering*, pages 58–67, Tokyo, Japan, September 1982.

[37] U. K. Wiil and J. J. Leggett. Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems. In *Proceedings of the ACM Conference on Hypertext*, Milano, Italy, November 1992.