Chimera: Hypermedia for Heterogeneous Software Development Environments

Kenneth M. Anderson

Department of Computer Science
University of Colorado, Boulder
ECOT 717, Campus Box 430
Boulder, CO 80309-0430
FAX +1.303-492-2844
e-mail: kena@cs.colorado.edu

Richard N. Taylor, E. James Whitehead, Jr.

Department of Information
and Computer Science
University of California, Irvine
Irvine, California 92697-3425
FAX +1.949-824-1715
e-mail: {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 hypermedia services in this heterogeneous setting.[1] 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 which may be stored in distinct object bases; viewers may be implemented in different programming languages; and, hypermedia services are provided to multiple, concurrently active, viewers. The paper describes the approach, supporting architecture and lessons learned. Related work in the areas of supporting heterogeneity and hypermedia data modeling is discussed. The system has been employed in a variety of contexts including research, development, and education.

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

Additional Key Words and phrases: heterogeneous hypermedia, hypermedia system architectures, open hypermedia systems, link servers, software development environments.

1 Introduction

Software development environments (SDEs) are used to develop and maintain a diverse collection of highly interrelated software objects [9, 22, 36, 58] in a distributed, multi-user context. Large software systems may, for example, consist of multiple versions of requirements specifications, designs, prototypes, source code, test information, scripts, and documentation. Establishing and exploring the many and complex connections between these com-

1. This article is a major revision and expansion of "Chimera: Hypertext for Heterogeneous Software Environments," which appeared in the proceedings of ECHT'94 [5].

ponents are major tasks of development, program understanding, and maintenance. In 1986 Delisle and Schwartz suggested that the attributes of hypermedia made it a promising technology for capturing and visualizing such relations in CAD environments (a closely related area) [18]. Since then several research groups have explored the idea of supporting navigation and access to SDE artifacts with hypermedia technology [15, 23, 25, 47, 48]. The objectives have typically included allowing an engineer to freely associate objects without regard to the type of those objects or where they are stored and, subsequently, to support their navigational access through a convenient user interface. This paper describes the approach employed by the Chimera open hypermedia system to address this research area, and also serves as an archival reference for Chimera's implementation techniques.[2] The primary purpose of this paper is to communicate how various techniques can be used in tandem to make environment-wide heterogeneity manageable and to provide enough information to enable researchers outside of the open hypermedia field to apply these techniques in new domains.

1.1 Technical Requirements

The design of the Chimera open hypermedia system evolved from our research in the software engineering and user-interface development domains [36, 57]. In particular, the following technical requirements are important for supporting and managing large-scale heterogeneity within these domains:

1. **Heterogeneous object editor & viewer support.** SDEs contain a wide variety of tools for developing and manipulating objects. SDEs also increasingly include multiple viewers of single objects, where each viewer presents different aspects of the object, perhaps using different depiction styles. We feel it is unlikely that software development teams will give up their favorite tools in exchange for hypermedia functionality. Ideally all editors and viewers in an environment should be able to use hypermedia services and respond to hypermedia events.

2. **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.[3] For example, program source code may be viewed as text, as a control-flow graph, or as a data-flow graph, with separate anchors on each view.

3. **Hypermedia information separate from object contents.** Applications unaware of hypermedia services, such as commercial compilers, must be capable of using their objects without change. Similarly the number and type of views of an object is commonly subject to unpredictable change over the life of a project. These requirements strongly suggest that hypermedia information be stored separately from the objects to which they pertain.

4. **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

---

2. According to Merriam-Webster's 9th Collegiate Dictionary, a chimera is "an individual, organ, or part consisting of tissues of diverse genetic constitution…"

3. In this section, we refer to intuitive notions of objects, viewers, views, anchors, and links. See Section 2.1 for more details.

views to be present simultaneously, where several views may be of the same object, and where actions in views may be autonomous and concurrent.

5. **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 essential to be able to establish links between objects that are managed by different repositories.

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

7. ***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 hypermedia support for SDE applications should provide such capabilities in the form of *n*-ary links.

Other concerns can be addressed by hypermedia systems that support SDEs. In fact, the hypermedia community has produced multiple sets of requirements for real-world hypermedia such as the fifteen assumptions for hypermedia-in-the-large [37] and the requirements set forth for industrial-strength hypermedia [38, 50]. Among these concerns are issues such as scripting, collaboration, versioning, and integration with the World Wide Web (WWW). We have explored the latter two issues in current research [2, 63]. However, in the initial design of Chimera we chose to focus our support on heterogeneity and providing core hypermedia services while keeping the entry barrier to use as low as possible.

This paper describes a set of concepts and techniques implemented by a system employed by users in real-world contexts. The notion of viewers of objects is at the heart of the conceptualization. We postulate an environment of many types of objects where display or editing of an object requires use of a viewer. Not all viewers are of the same type. How viewers manage their display and what services they provide is totally under the control of the software developer creating the viewer. We have developed a set of interfaces whereby a viewer announces to the hypermedia system the anchors it defines for its view of its object(s). These view-specific anchors can then participate in (many) links. Links are 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. Since heterogeneous environments are most often multilingual (i.e. multiple programming and scripting languages) and distributed, the generic architecture and our implementation is client-server based and a multilingual interprocess communication mechanism (Q [43]) is utilized in our implementation.

Example. A simple example demonstrates the concepts discussed so far (See Figure 1). Three separate applications utilize hypermedia services to provide a simple SDE. An integrated text editor (XEmacs) allows anchors to be created over ASCII source code while an integrated document processor (FrameMaker) provides the same functionality for design and requirements documents. A developer of an integrated flight simulator creates links to explicitly capture the implicit relationships between a gauge, its source code, and its requirements document. These

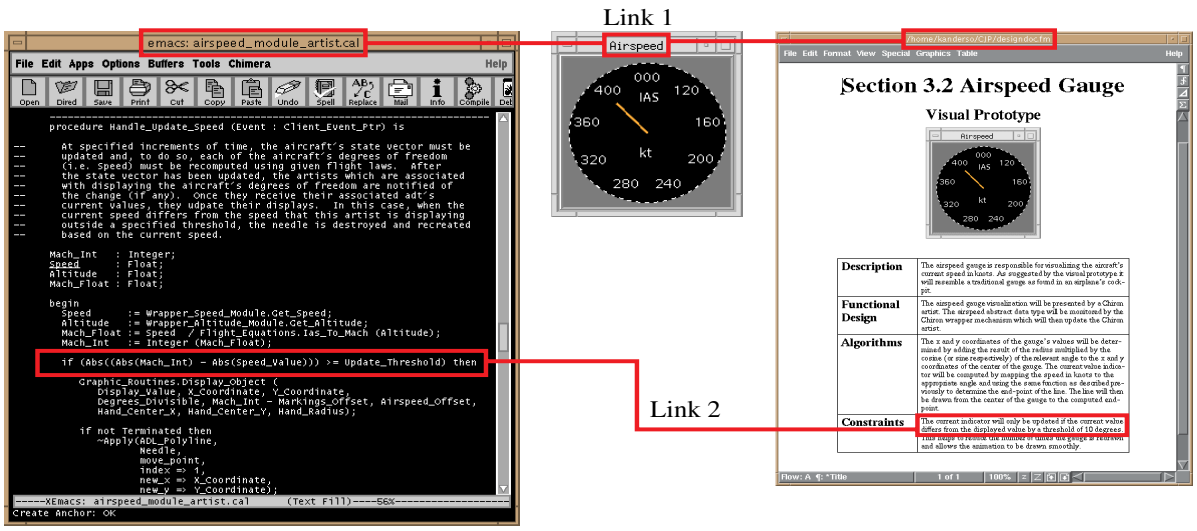

Figure 1. The essence of hypermedia support for software development environments. An airspeed gauge from an active flight simulator is linked (Link 1) to its source code on the left and its requirements document on the right. A more specific link (Link 2) relates a design requirement to the line of code which implements the specified constraint.

navigational relationships can be used by developers to quickly perform requirements traceability tasks with a single mouse click.

The FrameMaker documents and the ASCII source code are considered Chimera objects, as is the value for the flight simulator's airspeed. XEmacs, FrameMaker, and the airspeed gauge are Chimera viewers. The windows displayed by the editors and the visual depiction of the airspeed gauge are Chimera views. Anchors can be defined on each of these views. XEmacs provides anchor granularity at the row/column level, while FrameMaker maps anchors to specific paragraphs. The airspeed gauge associates only a single anchor with the entire gauge (as opposed to supporting multiple anchors referring to specific regions of the gauge). The straightforward mapping of the elements of the example into Chimera concepts demonstrates Chimera's flexibility.

This example justifies the existence of the technical requirements above. The presence of the flight simulator, FrameMaker, and XEmacs demonstrates the need for support of heterogeneous viewers (requirement 1) with multiple-view, concurrent, and active displays (requirement 4). Without satisfying these requirements, even simple scenarios such as this example could not be supported. All three clients have a different user-interface for manipulating and accessing anchors, demonstrating the need for view-specific anchors (requirement 2); the fact that each client stores persistent information in different formats requires the ability to link information across heterogeneous object managers (requirement 5). FrameMaker and XEmacs store anchor information separately from the original document. Since these documents are not modified to add support for hypermedia, they are available for processing by other tools, such as a FrameMaker plug-in or a compiler (requirement 3). In addition, *n*-ary links were required in order to simultaneously link the running flight simulator to both its requirements document and source code (requirement 7). Finally, while not specifically addressed by this example, the need for composable links or "links to

links" [31] (requirement 6), can be imagined by considering the use of "configuration management" links that enable traversals between sets of links instantiated for the gauge across multiple versions of the source code, requirements documents, and flight simulator.

1.2 Design Assumptions

Chimera's primary design goal is the accommodation of existing SDEs comprised of heterogeneous applications, data objects, and repositories. Designed to augment an existing SDE with hypermedia services without requiring modifications to existing clients, objects, or object stores, Chimera reduces the entry barrier for adoption. In this accommodationist approach, Chimera is aligned with existing open hypermedia systems [49] that also share the goal of increasing the value of existing environments, rather than drastically altering them.

Three control choices constrain the Chimera system architecture. Chimera controls the hypermedia structure—the links and anchors that express relationships in a SDE—but does not control the storage of the objects being related, nor the user interface used to display and manipulate them. This characterizes Chimera as a link server system according to the taxonomy of open hypermedia systems in [49]. An object manager, such as a file system, database, or configuration management system, controls object storage; Chimera accommodates the object manager(s) used by a given SDE. Each application controls its own user interface and provides link creation and traversal capabilities. In contrast, on the Web [7] the user interface is controlled via the browser. As discussed in [62], Chimera's control choices are similar to those made by other open hypermedia systems.

The choice of not providing object storage services distinguishes Chimera from open hyperbase systems (for instance [28, 46, 67]), the other main class of open hypermedia systems, which do provide storage for objects, along with anchors and links. Open hyperbase systems are similar to the research on object management systems performed in the software engineering community, in which SDE artifacts and relationships are stored in an object manager. Both open hyperbase and object management systems require existing tools to be integrated with the repository for storage of persistent data. Though open hyperbase systems do allow data to be stored outside the repository, applications that use the repository typically receive a higher level of service than those which do not [67]. Since integrating applications with a new storage repository requires additional integration effort over that required for integrating anchor and link creation and link traversal functionality, Chimera opted for the approach of accommodating existing repositories. This choice helps reduce integration, hence adoption, effort.

The Chimera system assumes its users are all located on the same local area network, and have access to the same object store, such as a network file system. Consistency maintenance of a hyperweb motivates this design choice. By centralizing hypermedia structure storage, the impact of a single change to the structure can be identified, and consistency maintenance steps can be performed, thus avoiding dangling links. By avoiding potentially high Internet latency, Chimera operations perform quickly, though limiting scale to local users. We recognize other systems address issues of relationship management across a wide area network (for instance, [34, 68]), and have ourselves addressed this in recent work [2].

As part of its strategy for accommodating existing SDEs, Chimera creates an internal model of the users, tools, objects and views used in the SDE. Chimera then connects a hypertext network to this model using links com-

prised of anchors defined on views. Chimera receives its knowledge about the external world from its clients, which directly manipulate and query the model using remote procedure calls (RPC). In reverse, Chimera updates clients when its model changes using event notifications that are also sent using the RPC mechanism.

The remainder of the paper is organized as follows: the next three sections present our conceptual foundations, implementation issues, and future plans; we then discuss related work, use in an industrial setting, system limitations, and conclude.

2 Conceptual Foundations

The primary components of an open hypermedia system are its hypermedia data model and its architectural abstractions. Chimera's data model consists of a flexible set of hypermedia concepts well suited for the software engineering domain. Chimera's architecture enables the provision of hypermedia services to a heterogeneous set of applications. We present details of both aspects below.

2.1 Hypermedia Concepts

Chimera's data model consists of the following elements (See Figure 2):

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

Viewers V*iewers* are named active entities that display objects. The operations provided by a viewer are specific to the viewer and the type of object(s) 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. For instance, a spreadsheet can be displayed in a traditional "rows and columns" view but can also be displayed as a chart. In addition, objects can be composed of other objects, thus multiple objects can be displayed in a single view. Views (and similarly anchors) do not necessarily have to be visual displays. For instance, this would be the case when an autonomous agent manipulates Chimera concepts via an application program interface (API).

Anchors A*nchors* 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.

Links A *link* is a set of anchors. All anchors in a link are equally related to each other. Thus a link traversal can begin at any member of a link and lead to any other member or members. This provides engineers with support for the complex relationships found in software projects in which one artifact is simultaneously related to many others. In addition, empty links and singleton links are supported. Unlike HTML links [7], Chimera links are not embedded in a data object. Furthermore, 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," [31] which supports construction of large-scale networks of links.

Attribute-Value Pairs To associate metadata with each instance of a Chimera hypermedia concept, an arbitrary number of *attribute-value* pairs can be defined on each instance. An attribute-value pair consists of two strings, one its name, the other its value. Hypermedia systems commonly use attribute-value pairs to provide run-time semantics or behavior for hypermedia entities [13]. Chimera uses attributes for naming links, recording anchor locations, providing access control information, filtering, etc.

Clients *Clients* contain one or more viewers. The client concept allows the accurate modeling of multi-viewer applications by separating per-client and per-viewer responsibilities.

Users *Users* interact with the views displayed by viewers (which are contained in clients). The user concept allows Chimera to track active users and their clients.

Hyperwebs A collection of objects, viewers, views, anchors, links, clients, and users, along with their attributes, is a Chimera *hyperweb*. A hyperweb is similar to Leggett's *hypermedia* [37] and Halasz's *hypertext* [30]. Some hypermedia systems provide support for contexts (for instance, [53, 67]) which are a partitioning and work isolation mechanism for hyperwebs [19]. Chimera does not provide a first class context mechanism, but does provide filtering on concepts by user as a form of limited context support.

A common question asked of all hypermedia models is "Who points at me?" or more generally, how are objects related to one another? Chimera's hypermedia concepts can answer this question in a straightforward fashion. A view represents a visualization of some object, and it contains a set of anchors, each of which can be included in many links. In order to answer the above question, a client would iterate over each anchor contained in the view and query for the set of links that contain it. The client would then iterate over the set of links to retrieve the other anchors that are included in each link. Each of these anchors belong to views that are related to the starting view, and each of these related views have an associated object. Once this process is complete, the client will have a list of objects that are related to the object associated with the source view. An obvious concern is performance; our implementation can perform this computation with acceptable speed. See Section 6 for more information.

Another common question asked of hypermedia models is related to consistency. That is, if some element is deleted, how is the model updated to remain in a consistent state. The answer for Chimera's model is also straightforward. Links can be deleted freely without impacting any other aspect of the model. If an anchor is deleted, it must be removed from all links that contain it. If a view is deleted, each of its anchors must be deleted. If either a viewer or an object is deleted, then all views in which that concept participates must be deleted. Deleting a viewer, object, or view, instigates a cascade in which potentially many elements are removed or updated. Thus, removing a viewer, causes all

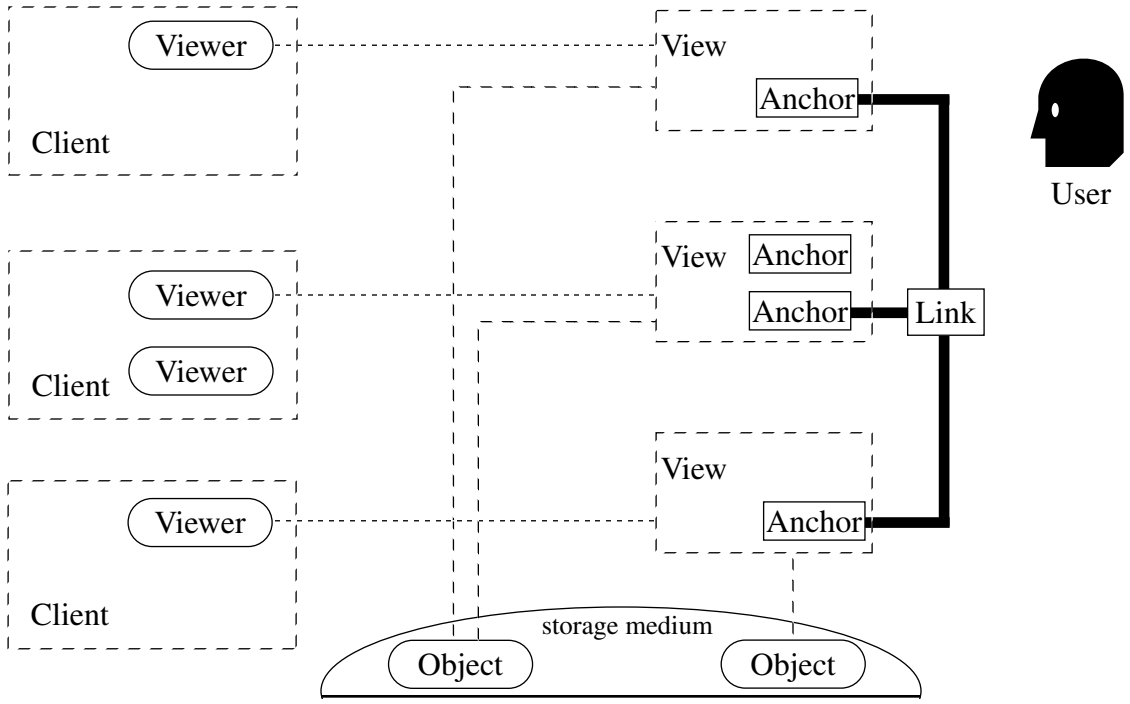

Figure 2. **Chimera's hypermedia concepts.** A user interacts with one or more views, generated by viewers. Each client has at least one viewer active within it. Viewers access objects via a storage mechanism (e.g. file system, database, object management system) and create one or more graphical views depicting them. Anchors can be created on these views either automatically by the viewer or explicitly by the user. Links define relationships between these anchors. Chimera's links are modeled as sets of anchors with each anchor in the set accessible by link traversal to the others. Attributes can be associated with each concept, but are not shown on the diagram to avoid clutter. A hyperweb consists of all of these elements.

of its views to be deleted, which in turn requires that all of their anchors be deleted, which in turn requires that each anchor be removed from any links that contain it. We continue our discussion of Chimera's concepts below.

2.1.1 Dexter Comparison

The hypertext community held four workshops in the late 80's to develop a standard reference model for hypertext systems, known as the Dexter hypertext reference model (Dexter) [30]. Dexter defines a model that spans both conceptual and architectural concerns. In this section, we map Chimera's hypermedia concepts into Dexter to help clarify their roles with respect to the standard model and to highlight a few differences that occur.

Objects. Chimera's object concept corresponds to Dexter's atomic component. More specifically, it corresponds to an atomic component's metadata (referred to as "component info" in [30]). Dexter's atomic component consists of content and metadata, and both are stored in Dexter's storage layer. In Chimera's storage layer, only metadata is stored; the content of an object can be stored anywhere and the knowledge for accessing that information is localized in viewers, thereby supporting environments with heterogeneous object managers. SDE artifacts come in many forms and are stored using a variety of storage mechanisms: requiring content to be stored in Chimera's storage

layer would restrict the range of types that can be used with Chimera and would negatively impact Chimera's ability to scale to large information environments.

In addition, there is a difference between the metadata stored in an atomic component and the information associated with a Chimera object. An atomic component's metadata consists of an arbitrary set of attributes, a presentation specification (pSpec), and a set of anchors. A Chimera object, on the other hand, stores only references to its associated content, and a set of attributes. Chimera objects have no need of an atomic component's pSpec since presentation details are handled by Chimera views, and anchors are associated with a view, not an object.

Viewers. Chimera viewers have no direct correspondence with any element in Dexter. Instead, Dexter defines a run-time layer model where components get instantiated. An instantiation is created by presenting a component to a user. This presentation is governed by two pSpecs: one pSpec is supplied at the time of instantiation, the other is retrieved from the component's metadata. One or both of these specifications selects the application that displays the component. This application, not explicitly modeled by Dexter, would correspond to Chimera's client concept. If this client had more than one viewer within it, then presumably Dexter's pspecs could also be used to select a relevant viewer during the instantiation process.

Chimera uses its viewer concept to accurately model one aspect of its external environment. Chimera knows what clients exist, the object types they can display (type information is stored as attributes on viewers), and the views in which they participate. Associating attributes with viewers allows Chimera to record viewer preferences along a variety of dimensions including, for example, whether a viewer can display more than one view at a time. Note, Chimera allows attributes to be associated with client concepts, however this capability is rarely used. In practice, associating attributes with viewers is more common since this information persists between sessions, whereas clients are modeled only at run-time. We anticipate making use of client attributes in the future, especially when Chimera's support for collaboration is enhanced (See Section 4.2). In particular, client attributes can be used to track characteristics such as the collaborative session a client is participating in, its coupling mode [20], etc.

Views. Chimera views also have no direct correspondence to any Dexter element. A view is essentially a static specification of possible instantiations that can occur in Dexter's run-time layer. A view records both the viewer and object involved in an instantiation and stores references to a set of anchors that have been associated with the instantiation. In Dexter, an instantiation consists of a base instantiation (the instantiation of the underlying component), a list of link markers (visual indications of the presence of a link), and a function that maps link markers into the underlying component's set of anchors. Dexter's link marker mapping function is not needed by a view, since that responsibility is handled by the anchor concept.

Anchors. In Chimera, anchors are associated with views, not objects, with the views acting as a level of indirection between the content of a component and the anchors that refer to that content. In Dexter's atomic components, this layer of indirection does not exist. Dexter anchors are stored in a component's metadata and maintain indexes into the component's content. This information is stored in Dexter's within-component layer. The indexes associated with a Dexter anchor are opaque to Dexter's storage layer and are interpreted only by an application directly aware of the format of the component's content.

Chimera attempts to raise the level of abstraction that is used to create these indexes. By associating anchors with views, the index that is stored in a Chimera anchor is defined in terms of the abstractions of the view, not in terms of the format of the underlying content. Consider anchors associated with two views of a spreadsheet object. One presents a typical "rows and columns" presentation of the spreadsheet, while the second presents a graphical pie chart view. Anchors on the former can make use of the row and column abstractions to specify the content to which they refer, while anchors on the latter can refer to individual pie slices on the pie char, referring to data synthesized from many different cells in the spreadsheet. While Dexter does not explicitly contain a view concept, it's notion of a composite can be used to provide similar capabilities. A composite component has the same structure as an atomic component except that its content can include one or more atomic components. In this instance, the anchors associated with a composite would be referring to the atomic components contained within the composite and would thus be at a similar level of abstraction as Chimera's anchors.

Links. A Dexter link is a special type of component that consists of a set of two or more component specifiers. Dexter requires that all links be at least a binary relationship and that all members of the link refer to existing components. No singleton or empty links are allowed. This decision has generally been viewed as too restrictive and the Dexter model has, by general consensus, relaxed these constraints [28, 37]. Each specifier in a Dexter link identifies a component, an anchor, a link direction, and a pSpec. The component and anchor are used to uniquely identify the information that the link is relating. A specifier's direction can contain the following values: NONE, TO, FROM, and BIDIRECT.

Chimera links contain references to zero or more anchors. Each anchor is assumed to have a directionality of BIDIRECT. Thus, a Chimera link traversal can start at any anchor and will lead to all other anchors in the set. Attributes take the place of Dexter's pSpecs for link traversal and can be used to achieve a variety of link traversal behaviors including, for instance, guided tours [13]. Recent work [2] uses anchor attributes to select link traversal behavior.

Missing Elements. The composite component is a Dexter conceptual element that does not have a Chimera counterpart. One way of addressing this drawback in Chimera is the use of views comprised of multiple objects, allowing anchors to be defined on composite views. Alternately, Chimera clients can construct an object reference that specifies multiple external objects. The problem with this workaround is that Chimera provides no way to associate anchors with subobjects, requiring viewers to employ ad hoc procedures to achieve this functionality. Recent work has extended Chimera's object concept to be able to contain subobjects explicitly. This extension provides an abstraction known as a static composite [31]; virtual composites [26], on the other hand, are not supported.

2.2 Conceptual Architecture

In this section, we define a conceptual architecture for our open hypermedia approach and discuss the requirements placed on each component of the architecture. In Section 3, we discuss how our implementation of Chimera meets the requirements defined in this section and consider their consequences.

A client-server approach is employed to meet the challenges of a heterogeneous environment, shown in Figure 3. This approach allows multiple users on different machines to access the sole active hyperweb from a

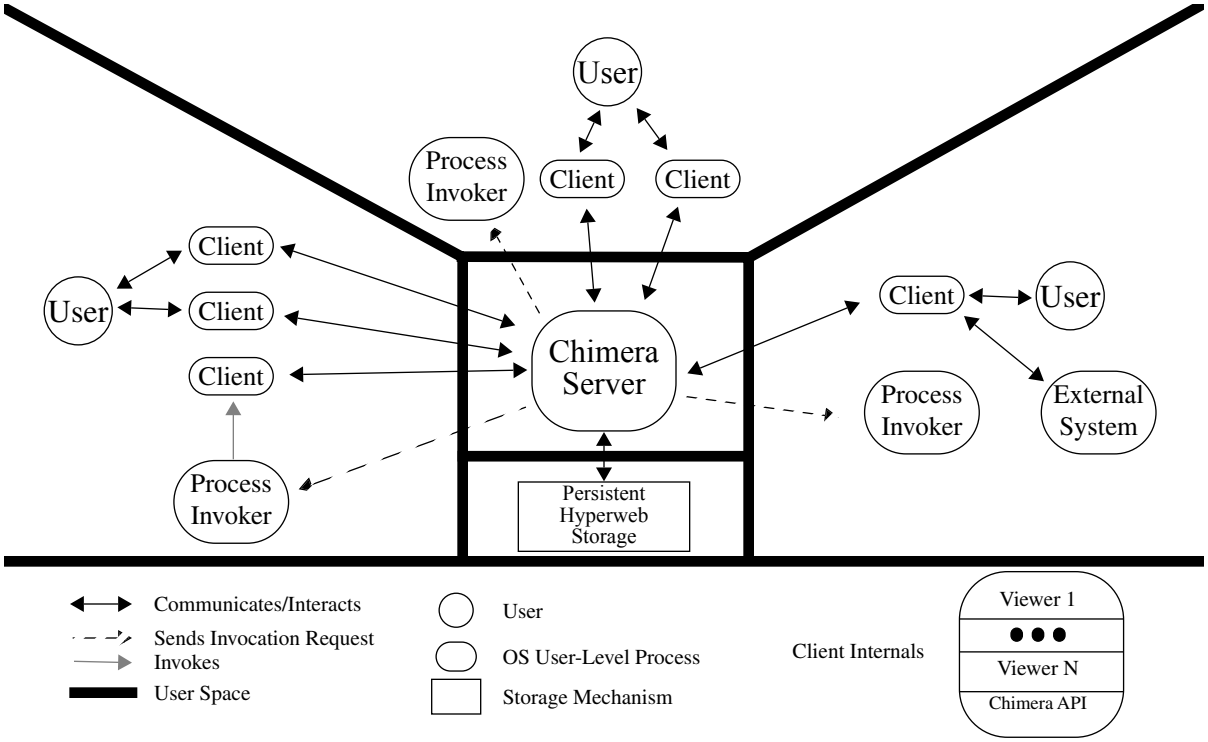

Figure 3. **Chimera's Conceptual Architecture.** Chimera uses a client-server architecture. Each user has a set of Chimera clients running in a distinct user space. These clients are typically interactive, although that is not a requirement, and may consist of multiple viewers running in separate threads. A client communicates with the Chimera server through the use of an API. Clients may also interact with other systems in a user's environment. Each user may run a process invoker which can activate clients needed to complete link traversals. The process invoker receives an invocation request from the server and maps the requested client to an executable in the user's environment.

dynamically changing set of viewers; hypermedia messages originate in one viewer and propagate to (potentially many) others via the server. Reaping a standard benefit of client-server systems, the use of a machine and language independent on-the-wire data representation allows clients to be written in different programming languages, each accessing the server via a language-specific API.

We now discuss the architecture's components: the server, clients, API, and process invoker.

2.2.1 Chimera Server

The *Chimera server* is the central element of the conceptual architecture. Its primary responsibility is to provide hypermedia services to its clients; these services include the generation of hypermedia messages, the management and persistence of hyperwebs, and the tracking of active clients.

There are three types of hypermedia messages: requests, responses, and events. Chimera employs a standard RPC communication model where clients generate requests and the server generates responses in reply. In response to changes in the hyperweb or to user actions, the server can generate events and send them asynchronously to inter-

ested clients. For instance, requests initiating a link traversal will generate link traversal events targeted at the appropriate destination clients.

The server persistently stores instances of its hypermedia concepts in an object management system. However, there is a distinction between Chimera's anchors, links, and views (where the server holds the master copy) and viewers and objects (which simply store a reference to some external entity). The distinction lies in the difference between the concepts that enable hypermedia services and concepts which directly model Chimera's external environment. Users can organize this information into multiple hyperwebs; when a concept is created it is assigned to the active hyperweb.

To facilitate rapid link traversal, the server tracks the active clients of all users interacting with a hyperweb, allowing it to know which clients are ready for link traversal events. The server also tracks whether a user has invoked a process invoker to enable delayed link traversals (discussed below).

2.2.2 Process Invoker

The *process invoker* is responsible for activating clients requested by the server. This occurs when the server needs to send a link traversal event to an unavailable client. This is called a *delayed link traversal*. If the user has a process invoker running, the link traversal event is held in the server until the target client has been spawned by the process invoker. After client initialization, the server forwards the event to the client, completing the traversal.

Use of the process invoker is at the user's discretion. Situations in which a process invoker might not be wanted include authoring activities, where links are being created between a focused set of applications, or resource limitations, where a user's computer might not be able to handle the demand of multiple applications invoked simultaneously. There is no more than one process invoker active per user.

The process invoker maintains a mapping between viewer names and executable programs (clients). After receiving a viewer name from the server, the process invoker determines the associated program and invokes it via operating system services. This map is supplied and maintained by each user of Chimera and is akin to MIME type mapping. This allows for a limited form of end-user tailorability: for instance, if two Chimera-aware JPEG viewers are available, the user can specify which one to invoke. Note that other systems also employ a similar approach to our process invoker, for example, HyperDisco [67].

2.2.3 Chimera Client

A *Chimera client* is an executable program containing one or more viewers and an API. All OHSs place demands on their clients [17, 51, 61]; Chimera is designed to keep these demands to a minimum. This approach enables a low-entry barrier to use and facilitates the integration of a diverse set of applications.

A. Client Responsibilities

A client is responsible for establishing, maintaining, and relinquishing a connection with the Chimera server. Since all viewers in a client share a single connection to the server, a client must route hypermedia events to its viewers. To enable the use of single-threaded clients that cannot simultaneously process and receive events, when the server sends a hypermedia event to a client, the server marks the client as unready to receive additional events, queuing subsequent events until the client indicates it can accept another. Whereas in Chimera the client interacts directly

with the server, in Hyperform [65, 69] a tool integrator sits between a client and the Hyperform server. The tool integrator contains a scheme interpreter, allowing it to perform services on behalf of clients, such as reducing the interaction work required of clients and caching application objects.

B. Viewer Responsibilities

A viewer allows users to manipulate objects by performing actions within its views. A viewer may access any external system in order to accomplish this task. For instance, a viewer can make use of a user-interface management system to provide its user-interface, or an object management system to store its persistent information. The Chimera architecture in no way restricts access to external systems, in order to accommodate the heterogeneity of typical SDEs.

The primary responsibility of a Chimera viewer is the mapping of the concepts “object”, “view” and “anchor” into its application domain. Mapping issues include how a view is presented to the user, 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 the viewer supports. The viewer can choose to store information to facilitate this mapping in a variety of ways. For example, a viewer can store display information in an attribute of an anchor to help reconstruct the anchor’s visual presentation. It can also store hypermedia information in a file that is associated with the set of objects being displayed, or it can choose to compute the mapping each time without the use of persistent information.

The second responsibility of a Chimera viewer is to correctly utilize the hypermedia protocol of the Chimera server by making the appropriate calls on the Chimera API. This includes notifying the server when it is active, registering for hypermedia events, indicating its current set of views (this set will change over time based on user interaction), and handling hypermedia operations such as creating anchors and adding them to links.

The final responsibility of a Chimera viewer is the handling of hypermedia events. For instance, a viewer typically responds to a link traversal event by opening the target object and highlighting the destination anchor. Once a viewer has finished processing an event, it must notify the client, which then informs the server that it’s ready for more events.

The drawback to this approach is user-interface inconsistency. Since each viewer may choose to implement access to the hypermedia services in different ways, as dictated by its application domain, Chimera cannot guarantee a uniform interaction style. This is potentially troublesome since the user has to remember how this hypermedia functionality is invoked for each viewer [23]. This is a design trade-off involving ease-of-use, open systems, and customized interfaces. We believe a single, standard style is too restrictive and 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 simplify the task of writing viewers and promotes uniform authoring, display, and interaction styles.

2.2.4 Chimera API

The Chimera API is a language-independent set of routines allowing remote creation and manipulation of hypermedia concepts, management of server connections, and handling of hypermedia events. The API must contain

routines for creating, manipulating, and deleting Chimera concepts, along with a set of query routines for viewers to search information in the hyperweb. In addition to routines for establishing and terminating server connections, the API must also provide a mechanism to notify a client if a connection terminates unexpectedly. Finally, the API must provide a mechanism allowing a client to specify how it is notified of hypermedia events.

3 Implementation Issues

An initial prototype of Chimera was constructed in the summer of 1992. Since then, Chimera has evolved significantly. The hypermedia concepts discussed in Section 2.1 have been flexibly employed in a variety of client integrations. The conceptual architecture discussed in Section 2.2 has been realized in a concrete implementation. In this section, we describe our implementation choices, along with the issues encountered and the lessons learned in mapping our conceptual architecture into a robust system.

3.1 Chimera Server

The implementation of the Chimera server, described below, addresses the issues and implements the responsibilities discussed in Section 2.2.1.

3.1.1 Internal Architecture

The multi-threaded internal architecture of the Chimera server is shown in Figure 4. The server is divided into three conceptual tasks: handling client requests, managing the hypermedia network and external world model, and generating events.

A. Handling a Client Request

The Service Request Handler manages client connections, marshalling and unmarshalling of parameters for RPC requests, and routing requests to invocations on abstract data types (ADTs) in the World model. Client connections last for as long as the client is in operation. As an example, a query by the client for the number of views present in the active hyperweb results in request unmarshalling by the Service Request Handler, which then invokes the HOW_MANY_VIEWS routine in the Hyperweb ADT, located in the World model. The HOW_MANY_VIEWS routine queries the object management system for the correct result, which it hands back to the Service Request Handler for marshalling as a response message and transmission to the requesting client.

B. Hyperweb and World Models

The Object Manager and World Model data structures consist of nine ADTs each. The Object Manager ADTs, generated by Pleiades [56], model the state of the managed hyperweb and support query, manipulation, and persistence operations upon it. All of the server's persistent information is read into memory at server start-up and written to disk at server shut-down. This limitation (since new information can be lost if the server terminates unexpectedly) is due to historical constraints imposed by Pleiades, and has been addressed in a new version of Chimera that employs conventional relational database technology [2]. Each user selects which hyperweb they want to access through a resource file (.CHIMERA) located in their home directory, which the server reads on start-up.

The World Model ADTs track the state of the server's external world. This includes keeping track of all users interacting with the hyperweb, all connected clients, all active viewers, and all connected process invokers. The Chi-

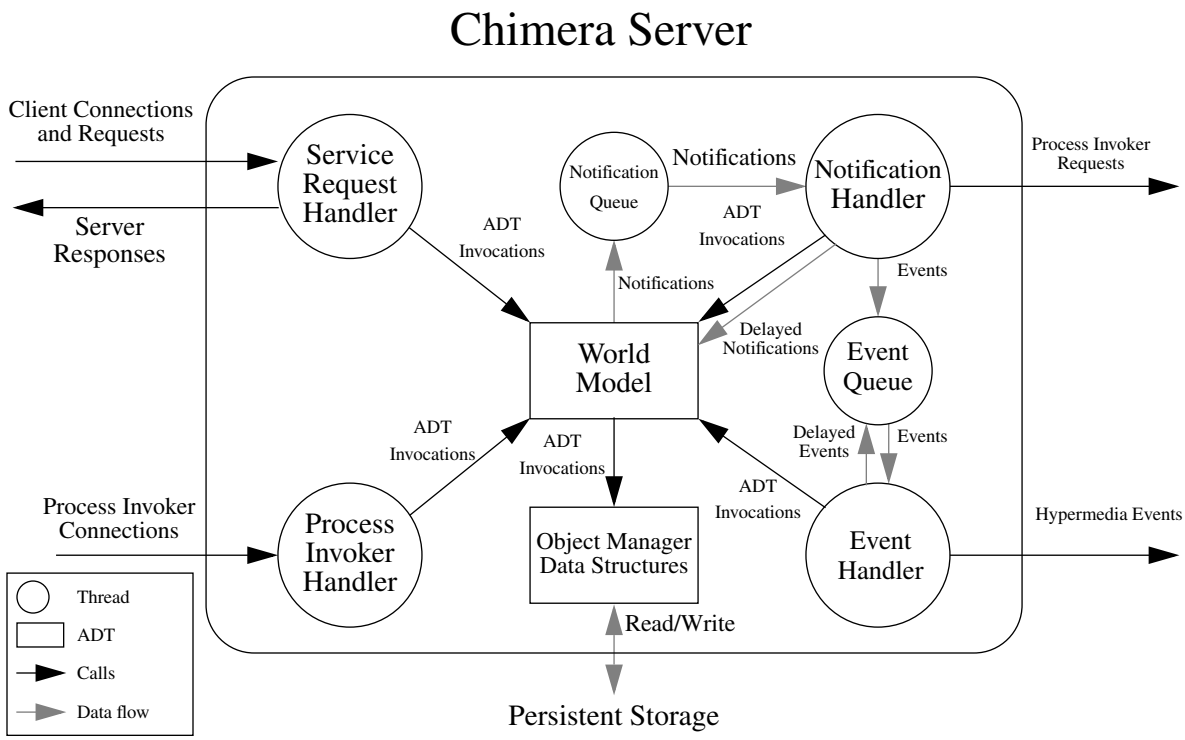

Figure 4. **Chimera Server's Internal Architecture.** The Chimera server's responsibilities are distributed over a set of threads and ADTs which handle incoming requests from both clients and process invokers, generate and handle hypermedia notifications/events, maintain a model of the external world, and update the persistent state of the managed hyperweb.

mera server stores the following information about a user: name, active clients, process invoker status, active link, and delayed link traversal events. For a connected client, the server maintains the client id, client status, client notification connection, host and display information, client filtering mode, and client viewers. For an active viewer, the server stores the id, views, and interests in events.

The server stores the existence of a process invoker and its connection information with the process invoker's user record. This information is updated by the Process Invoker Handler. A process invoker contacts the Process Invoker Handler once at start-up and shut-down, allowing the server to determine if a particular user can receive delayed link traversals.

C. Generating Hypermedia Events

Certain changes to the World Model cause the generation of internal notifications. Each notification is processed by the server's Notification Handler to determine if a client has registered interest in the notification, and, if so, it generates one or more hypermedia events from the notification. Chimera currently defines fourteen events which are described in Figure 5. Events sit in a separate monitored queue until the Event Handler determines (via the World Model) if a client is ready to receive an event. If the client is ready, it sends the event, if not, the event is placed back on the event queue. All events for a particular notification are handled before the events of a subsequent notifi-

Event Name	Associated Information	Explanation
Server Disconnected	None	Unexpected loss of connection
Viewer_Registered	Viewer Identifier	Viewer is added to persistent store
Viewer_Unregistered	Viewer Identifier	Viewer is removed from persistent store
Object_Registered	Object Identifier	Object is added to persistent store
Object_Unregistered	Object Identifier	Object is removed from persistent store
View_Registered	View Identifier	View is added to persistent store
View_Unregistered	View Identifier	View is removed from persistent store
Anchor_Registered	Anchor Identifier	Anchor is added to persistent store
Anchor_Unregistered	Anchor Identifier	Anchor is removed from persistent store
Link_Registered	Link Identifier	Link is added to persistent store
Link_Unregistered	Link Identifier	Link is removed from persistent store
Link_Modified	Link Identifier	The link's set of anchors has been changed
Link_Traversed	Anchor Identifier	A link traversal has led to this anchor
Link_Activated	Link Identifier	The link is now the active link for the user

Figure 5. **Chimera's hypermedia events.** The events are notifications of actions taking place in the server. Clients register interest in events; the server sends these events when they occur. The two most common events are LINK_TRAVERSED and LINK_ACTIVATED. The former is used to notify a client of a link traversal; the latter is used to notify a client that the active link of its user has changed. The SERVER_DISCONNECTED event is different from all others since it is generated by the API rather than the Chimera server. It is used to notify the client when the connection to the Chimera server unexpectedly terminates.

cation, thus a partial ordering is maintained by the event queue. This has the drawback of forcing all clients to wait for the slowest one to catch-up (assuming it ever does) before new events are received.

3.1.2 Link Traversal

Since the most basic functionality of any open hypermedia system is the ability to traverse links, we now describe Chimera's link traversal process. A link traversal begins when a user selects an anchor in an active view and performs the link traversal operation on it. The viewer invokes the TRAVERSE_LINK subroutine of the API, passing the identifier of the selected (e.g. source) anchor. If the anchor is valid (e.g., registered in the active hyperweb), a LINK_TRAVERSED notification is placed on the notification queue.

The Notification Handler retrieves every link which contains the indicated anchor. It iterates through this list and for each link retrieves its anchors. It then iterates through this list of anchors and for each anchor that is not equal to the source anchor, it determines if the anchor's viewer is currently active. If the desired viewer is not running, the Notification Handler queries the World Model to see if the user has a process invoker running and, if so, generates a message to invoke the desired client. If a user does not have an active process invoker, the traversal ends.

Otherwise, when the desired viewer is running, the Notification Handler checks to see if it has registered interest in LINK_TRAVERSED events. If so, a LINK_TRAVERSED event is created, parameterized by the destination anchor and destination viewer, and placed on the event queue. If not, since the client is uninterested in link traversal events (as is the case with a client that can only generate requests, but cannot receive events), the traversal to this particular anchor ends and will not be completed.

Eventually, each event is removed from the event queue by the Event Handler which checks to see if the destination client is ready to receive a hypermedia event, and, if so, sends it the LINK_TRAVERSED event. If the client is not ready, the event is placed back on the event queue where it will be processed at a later time. The link traversal ends when all destination clients have received their LINK_TRAVERSED events. The client forwards the event to the indicated viewer which examines the destination anchor and displays this anchor to the user. The specific steps executed are different for each viewer but in general involve determining the view associated with this anchor, determining the object associated with this view, opening the object, displaying the view, and highlighting the destination anchor.

3.1.3 Additional Server Features

In addition to fulfilling all of the requirements described in Section 2.2.1, the server implements the additional features of filtering, active links, and invocation policies.

A. Filtering.

Filtering is used to provide different hypermedia perspectives of the same view. Based on a filtering level for each client and information about a client's user, anchors and links can be filtered such that different sets can be provided to different users for the same view. 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 limited support for multiple contexts within a hyperweb.

B. Active Links.

Active links are a mechanism provided by the Chimera server to allow modeless link creation. One active link is maintained for each user connected to the Chimera server and it acts as a pointer to an existing link within the active hyperweb. In most hypermedia 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, as well as in some other systems [28, 46, 67], a user can create several empty links, select one of these links to be the active link, 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. Thus, the active link is a level of indirection which allows the sharing of a single link between a user's clients.

C. Client Invocation.

With respect to client invocation, our experience has distinguished a difference in the styles required by two types of viewers. Some viewers are written to support the display of multiple views. When this type of viewer receives a LINK_TRAVERSED event to an undisplayed view, a new window is typically created to display the desired view. Thus, only one invocation of this viewer is needed to display multiple views, and this is indicated to the Chimera server by specifying a value of ONCE_ONLY for the INVOCATION_POLICY attribute, a pre-defined attribute on all viewers. Other viewers are not as sophisticated and do not have the ability to switch views after they have been invoked. Thus, multiple instances of this viewer are needed to display multiple views simultaneously to the user. This type of viewer is indicated to the Chimera server by assigning a value of EVERY_TIME to the viewer's INVOCATION_POLICY attribute.

3.1.4 Server Metrics

The server is implemented as a multi-threaded Ada application. It consists of approximately 81,000 lines of commented source code. However approximately 60,000 lines of that code (74%) was either generated or reused. Specifically Pleiades, our object management system, generated 36,000 lines of code used to implement the Chimera ADTs which model the structure of a hyperweb at run-time and manage its persistence. The 24,000 lines of reused code consists of ADTs such as linked lists, binary search trees, and hash tables from a library of software components produced by the University of Massachusetts, Amherst [56].

3.2 Process Invoker

The process invoker relieves the user from anticipating the applications needed for a particular session; if a link traversal leads to a new application, the process invoker activates it. Once started, a process invoker contacts the server to announce its existence, reads its map file (as described in Section 2.2.2), and then enters a loop waiting for service requests. The process invoker map is stored in a user's home directory and read once at start-up. The limitations implicit in this will be fixed in future implementations by providing tools to manage a process invoker's map at run-time or by switching to a dedicated tool server.

When the server requests the process invoker to spawn a new process, it sends a message containing the name of the desired viewer and the name of the display upon which the viewer should place its output. The process invoker searches for the specified viewer name in its association list and invokes the associated client, if found. Since the process invoker allows a remote process, the server, to invoke a program on a user's machine, some security risks occur. These risks are limited, however, since the map file restricts the applications that can be run, use typically occurs within a LAN setting, and a process invoker only interacts with a known server.

3.3 Chimera Client

A client is any entity interacting with the server via the API. This definition covers a wide range of potential architectures including hypermedia-unaware applications being controlled by Chimera-aware wrappers, single-threaded C programs, and multi-threaded Java Applets. We have developed and integrated an extensive variety of clients including text editors (XEmacs and xvi), image, sound, and video browsers, Chimera-specific tools (hyperweb editors and link viewers), and wrappers for third-party tools (FrameMaker). As an example, the integration of XEmacs with Chimera is shown in Figure 6. In the remainder of this section, we examine the consequences of the responsibilities placed on Chimera clients and viewers in Section 2.2.3.

3.3.1 Impact on Legacy Systems

Integrating Chimera with an application while it is being written is easy. By including Chimera support from the start, the integration can occur at a deep level and provide sophisticated hypermedia services. Unfortunately, new applications are vastly outnumbered by legacy systems.

Legacy systems cannot typically achieve the level of hypermedia support new applications provide. In general, they are large, complex, programs not designed with hypermedia in mind. They do not often provide an API which would allow them to be controlled by external applications. Thus, an important issue for OHSs is incorporating hypermedia functionality into these systems in a minimally intrusive fashion [6, 8].

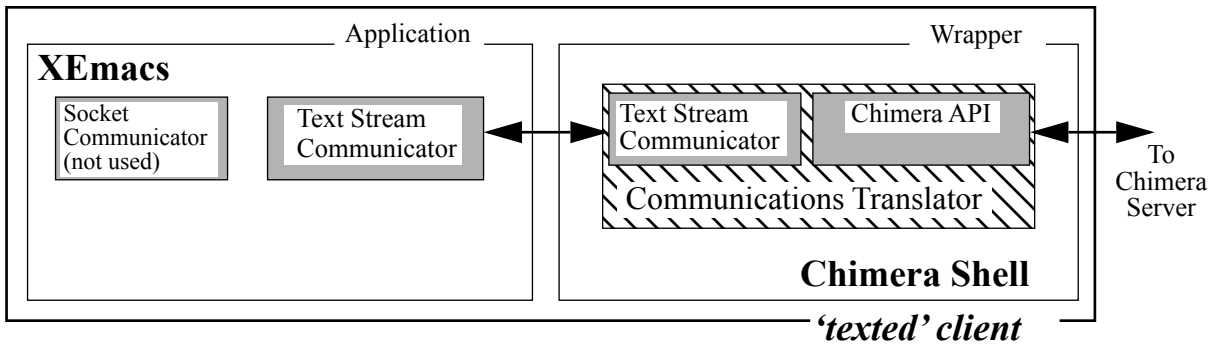

Figure 6. **XEmacs Integration.** The Chimera Shell contains a communications translator which converts between the text stream communication of XEmacs and the API-based communication of Chimera. XEmacs and the Chimera Shell together implement the Chimera 'texted' (text editor) client. XEmacs possesses the ability to create sockets for use in inter-process communication, but the XEmacs/Chimera integration does not use this capability.

If the legacy system has an API or an extension language, a wrapper can be written and assume the Chimera responsibilities for the legacy system. If these mechanisms do not exist then the source code for the legacy system must be modified to fulfill these responsibilities. If the source code is unavailable then the best option is to attempt a launch-only integration with the legacy system, a technique also employed in [17, 28, 68].

3.3.2 Handling Service Interruption

Since Chimera has a client-server architecture, Chimera clients must be prepared to deal with the issue of service interruption. After a connection has been established, the API automatically monitors (via its underlying RPC mechanism, Q) the connection to detect an unexpected termination of the connection, generating a SERVER_DISCONNECTED event if the connection is dropped. A client is free to do anything at this point, ranging from terminating execution to using a dialog box to inform the user of the suspension of hypermedia functionality. While clients can attempt to reconnect to the server, this process would essentially involve an inefficient polling process. Most developers opt to deactivate the hypermedia capabilities of their clients, placing the burden of discovering when access to the server is restored on the end-user. Clients which take this approach often provide a way (via a menu item or button) to re-establish contact with the server.

3.4 Chimera API

The Chimera API provides the foundation upon which applications in a user's environment are easily integrated with the Chimera system. The API consists of eighty-one entry points. Due to Chimera's use of Q, which employs the External Data Representation (XDR) portable data transmission standard, it is possible to write Chimera APIs which are independent of implementation language and machine byte ordering. Chimera supports clients written in Ada, C, and Java, with APIs for all three languages. Several clients have been written using each of these APIs.

The Ada API creates two Ada tasks per viewer. One task handles sending messages to the Chimera server; the second task handles receiving hypermedia 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 user-interface management system, 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 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, six separate C programs have been written which are simultaneously Chimera and X clients within a single Unix process.

The Java API is a set of Java classes allowing access to Chimera services within multi-threaded Java applications (and applets). The Java API creates two threads per viewer similar to the Ada API; one thread sends messages to the server and the other receives events from the server. Java applications implement an interface defined by the Java API so that hypermedia events can be routed to them.

4 Future Work

While our experience to date with Chimera has been very positive, there are several areas in which advancements would increase the value of the system. Three are particularly notable in the SDE context: support for versioning, support for collaborative hypermedia, and the integration of open hypermedia systems with the WWW; they are discussed below. Also important, but not discussed here, are (a) easing the difficulty of adding hypermedia capabilities to existing tools and (b) supporting computed links. The first topic is being addressed in an associated project [61]. Computational links are already supported in a variety of other projects, for instance [32, 53]. It remains for us to determine the most effective way of adding this functionality to Chimera.

4.1 Versioning

Version control mechanisms are very important for any hypermedia system that wishes to support software engineering activities in a non-trivial fashion [11, 19, 29, 31, 33]. Chimera is no exception, and a mechanism for versioning is a research priority [64]. Since Chimera intentionally offers no support for storing application objects within its hypermedia database, version control responsibility must be shared between Chimera, its client applications, and external object managers. For example, when making links between source code files, version control responsibility for the files will rest with a revision control system such as RCS [59], while responsibility for versioning the *relations* between the files will reside with Chimera.

We envision each Chimera concept becoming capable of multiple versions. This raises 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 maintaining associations between versions maintained by an external versioning system and versions maintained by Chimera.

4.2 Collaborative Hyperweb Construction

Chimera does not support collaborative building of hyperwebs in real-time by multiple users. Limiting factors include the lack of transactions over the hyperweb (two users modifying the same portion of a hyperweb can overwrite each other's changes), as provided in WebPern [70] and HB/SPn [53], and the lack of awareness mechanisms, such as in Sepia [55]. Thus, Chimera currently relies on its users to coordinate their actions in the shared information space. In recent work [2], a simple locking mechanism has been added to the Chimera server as a step towards fully supporting collaboration, however a transaction mechanism is still pending. For a detailed look at issues related to concurrency control in collaborative hypermedia systems, the reader is referred to [66].

4.3 Integration with the WWW

The WWW [7] offers significant potential for software development by globally-distributed software teams [24]. The Web's advantages include extensible, standard data formats and access protocols, low-entry barrier to use, and a high degree of distribution in terms of work and data. Open hypermedia systems can significantly benefit by leveraging these features through integration since they would be able to provide their services to a wider range of clients and over a larger amount of data than currently possible. This integration would also provide benefits to the WWW, since some of its current weaknesses—link consistency management, modeling arbitrary relationships among disparate data types, and modeling the links currently embedded in HTML—can be addressed by the abilities of open hypermedia systems. Recent work has addressed these issues in Chimera [2] and, in addition, this important topic has recently received a significant amount of attention [10, 12, 27, 32].

5 Related Work

A wide range of research has been conducted in the area of hypermedia systems and their application to software environments. Hypermedia systems have evolved from black-box (monolithic) applications to hyperbase and linkbase systems to open hypermedia systems. Monolithic hypermedia systems store all information (both content and hypermedia structures) internally. No external applications or information can participate or be included in the hyperweb. These systems (such as KMS [1] or Apple's HyperCard) excelled at producing self-contained hyperwebs on a variety of topics, however they did not lend themselves well to software development due to a lack of support for heterogeneity. In particular, they do not support webs distributed over multiple heterogeneous object managers nor is the set of tools which can manipulate the web open.

We have argued that support for heterogeneity is essential to support the software development problem. As such, we now examine related work among the subsequent forms of hypermedia systems, comparing and contrasting their support for heterogeneity with Chimera's. In addition, we briefly discuss elements contained in two additional hypermedia models and compare them to Chimera's set of hypermedia concepts.

5.1 Heterogeneity

In Section 1, we argued that hypermedia is suitably powerful and flexible to manage the complex relationships inherent in large, diverse software projects. A key characteristic of such large software projects is their heterogeneity: in the data types of software development artifacts, in the application programs used to support the

development, and in the platforms and implementation languages employed during development. For application domains characterized by such heterogeneity, the approach embodied by the Chimera system offers strong support in addressing this heterogeneity. This support derives from fulfilling the hypermedia requirements described in Section 1.1 along with specific implementation techniques employed with heterogeneity in mind. Existing hypermedia systems typically address some, but not all, of these aspects of heterogeneity either by failing to meet some of our requirements or by utilizing implementation techniques best suited for a homogeneous environment. Hypermedia database (hyperbase) systems inadequately handle the diversity of application tools, while existing hypermedia link servers are too heavily tied to a single implementation language or platform. These characterizations are detailed below.

Work in hyperbase systems has a long research history. This work can roughly be divided into closed and open hyperbase systems. An incomplete, but representative, sample of closed hyperbase systems includes NLS/Augment [21], Neptune [18], HAM [11], HB1 [52], HyperWeb [23], and SEPIA [55]. Closed hyperbase systems employ a database to store both application content and hypermedia structures. While hyperbase systems store and retrieve a wide range of data types (such as text, graphics, and sound), the set of types is dependent on the underlying database and cannot easily be extended by the end-user. This hinders support for heterogeneity since new types cannot be added to the hyperbase in a straightforward fashion. Closed hyperbase systems cannot easily integrate new tools since a tool must be modified to store all of its persistent information in the hyperbase. Since a closed hyperbase can only provide hypermedia services over the data stored within it, no external information can be linked into the system.

A representative sample of open hyperbase systems includes DHM [28], HOSS [46], Hyperform [69] HyperDisco [67], and SP3 [37, 53]. These systems differ from closed hyperbases in that application's can choose to store their content outside the hyperbase. However the quality of service provided to these applications is typically lower than applications which use the hyperbase for all their storage needs [67]. The reason for this difference in service lies in the fact that the internal mechanisms of the hyperbase enable some features, such as versioning, to be provided automatically for content stored in the hyperbase. In addition, some open hyperbase systems have extensible data models [67, 69], allowing the set of data types supported by the hyperbase to remain open.

Link servers [44] are an alternative approach to open hypermedia systems [49]. Example systems include Chimera [5], Microcosm [16, 32], Sun's Link Service [51], and PROXHY [35]. In contrast to hyperbase systems, link servers focus on storing only hypermedia structures in their database. Applications are free to store their persistent data anywhere. This directly addresses the heterogeneity of data types, since the link server is kept unaware of the set of data types used by its clients. With respect to the heterogeneity of clients supported by the above three types of systems, both link servers and open hyperbase systems can support a wider range of clients than closed hyperbase systems, since they do not place restrictions on where an application stores its data. In addition, it is easier to achieve a complete client integration [17, 61] with a link server since a complete integration with an open hyperbase system requires that an application store its data in the hyperbase. On the other hand, the level of service provided to a fully integrated client by an open hyperbase system is higher than a link server, since link server systems do not provide direct support for versioning.

The WWW [7] addresses one issue of heterogeneity the proceeding systems, in general, ignore. The Web's approach of establishing a standard set of data formats and communication protocols directly supports the heterogeneity of computing platforms, allowing a hypermedia system to be widely distributed cross-platform. This coupled with the fact that the Web is scalable in terms of the amount of information which can be linked, extensible in terms of the capabilities of its protocols, and light weight in terms of its low-entry barrier to use makes the Web the single most widely-adopted hypermedia system to date. However, the Web has several weaknesses in which it can benefit from the lessons learned in open hypermedia systems. In the WWW, links are not first-class objects and thus must be embedded into the linked objects directly. Data types not suited for link embedding can be the destination of hypermedia traversals, but not the initiator of a link; the information must first go through a data conversion process in order for such data to become hypermedia-enabled. Likewise, the only method for analyzing links (and thus resource dependencies) in the Web is to actively traverse the hyperweb, either manually or with the assistance of software agents (e.g., web spiders). This limits the system's ability to maintain referential integrity when resources change, to assist the user in the navigation and visualization of the hyperweb, and to predict the effects of a change before it is made.

Section 4.3 contains pointers to research in open hypermedia systems that explore the integration of open hypermedia services into the Web. In addition, recent work [3, 4] has addressed issues of data scalability in open hypermedia systems, although it will be difficult for open hypermedia systems to attain the scalability achieved by the Web. Finally, it should be mentioned that various research groups are examining alternative architectures for providing hypermedia services over the Internet [42, 68]. In particular, HyperWave [42] adopted an approach of providing a new hypermedia server and client to replace existing WWW servers and clients. The new server and client provide features found in open hypermedia systems, such as maintaining consistency of the hypermedia model, and were compatible with existing Web protocols. Thus, a user of a standard Web browser can interact with a HyperWave server (although at a reduced quality of service) and HyperWave users can still access all of the information stored on the Web. HyperWave is not as scalable as the WWW, however, because of its goal of maintaining link consistency which requires a HyperWave server to inform all other HyperWave servers distributed across the globe when a particular entity (such as a link, anchor, or object) is deleted.

5.2 Hypermedia Models

Section 2.1.1 compared Chimera's hypermedia data model with the standard Dexter model. While this comparison is sufficient to characterize the conceptual power of Chimera's hypermedia concepts, there are additional features in other hypermedia models that do not appear in Dexter, and thus should be briefly described and compared with Chimera. The two features considered here are Microcosm's generic links and VIKI's end-user abstractions.

Generic Links. Microcosm [32] has a simple hypermedia model which provides three types of links: specific, local, and generic [16]. Chimera's links are equivalent to Microcosm's specific links, and local links can be modeled in Chimera using whole-component anchors [30]. Generic links, however, are not supported by Chimera. A generic link is a very powerful mechanism whereby a user can author a single link that can appear in multiple contexts. For instance, a user can specify that anytime a particular phrase appears in a document, it should be linked to

some other document. This mechanism requires a computational engine of some sort that examines the content of linked documents looking for matches to generic link specifications. Since Chimera is unaware of the content maintained by viewers—a design choice selected to increase Chimera's support for heterogeneity—it is difficult to add a generic link mechanism. Instead, recent work [2] allows users to dynamically load new link traversal semantics into the Chimera server. Clients can access this new functionality via the Chimera API. Since these new link traversal semantics can require different types of input than the traditional link traversal algorithm of Chimera (which accepts source anchors and maps them to destination anchors), a mechanism similar to generic links can be enabled. However clients must be modified to send the appropriate information, for instance a text string, to these new link traversal algorithms.

End-User Abstractions. Hypermedia systems vary in terms of the abstractions they present to a user. By this, we refer to the model that end-users form when working with a hypermedia system and the tools that are provided for manipulating that model. For instance, despite the full set of concepts described in Section 2.1, Chimera's end-users primarily think in terms of hyperwebs, anchors, and links. The remaining concepts and the architectural abstractions discussed in the rest of the paper are backgrounded, not really needed by end-users to complete their tasks. One end-user abstraction mechanism, often provided by hypermedia systems, is the ability to create typed links [60]. Chimera does not allows users to create new link types; instead, this power is given to client developers to create links which are tailored to a particular client or set of clients.

However some systems go beyond this simple form of abstraction to more powerful mechanisms. Examples include gIBIS [14], Aquanet [39] and VIKI [41]. For instance, VIKI is a spatial hypermedia system used to study, among other things, information triage [40]. Information triage is the process of determining how a set of documents can be best organized to support a particular task. VIKI is well suited to this task since it allows users to create task-specific hypermedia structures which use composition and spatial layout to help users organize and represent knowledge over a set of information. While these mechanisms are powerful, Chimera's goal of providing navigational hypermedia services in heterogeneous SDEs did not require the use of them, at least not initially. By keeping the hypermedia model simple, it is easier to provide support for it in a wide range of clients. In the example systems above, the end-user mechanisms are so powerful and specialized that the application which provides them is the hypertext system itself. For instance, VIKI's end-user abstraction mechanism are not available outside the VIKI browser. However, the desire to bring more powerful end-user abstraction mechanisms into open hypermedia environments, has contributed to the creation of a research direction known as structural computing [45], and new research in this area may develop techniques for integrating these mechanisms across a heterogeneous set of applications.

6 Use in an Industrial Setting

Chimera has been used in a variety of contexts, including software engineering classes, research environments, and as the hypermedia infrastructure in industrial technology demonstrations. For example, the Military Aircraft Systems Division of Northrop Grumman Corporation, as part of their contract with the Defense Advanced Projects Research Agency's Evolutionary Design of Complex Systems program, conducted a case study involving

Chimera. The case study developed a technology demonstration of a prototype next-generation development environment for avionics software. The demonstration covered multiple phases of a software development lifecycle, including requirements tracking, software testing, and architecture evolution. Chimera was used as the underlying hypermedia infrastructure both to demonstrate its use in navigating software relationships (such as links between running simulations, source code, design/requirements documents, and Web-based documentation) and in driving the demonstration itself (via Chimera's process invocation services).

The development group which implemented this demonstration received their first Chimera demonstration in February of 1997. They acquired Chimera and several off-the-shelf Chimera clients and began constructing their demonstration hyperweb in late March. Integration of Chimera with one of their own applications began in April and was completed in time for the July demonstration. Thus, Chimera was evaluated, adopted, and used in a large-scale technology demonstration within a period of six months. Not all of the demonstration preparation time was devoted to Chimera however—the demonstration integrated a variety of new technologies—so the actual time to learn and apply Chimera is significantly less than six months.

The off-the-shelf Chimera applications used in the technology demonstration were Netscape Navigator, XEmacs, Chimera hotlist, and XGif. Netscape and XEmacs allow HTML and ASCII (e.g. source code) documents to be linked into Chimera hyperwebs. The Chimera hotlist is a client which provides a function similar to the bookmark mechanism in WWW browsers. Essentially, the hotlist client stores a list of frequently accessed Chimera views, allowing users to return to the stored views rapidly. XGif is a public domain GIF image viewer modified to be a Chimera client. The industrial demonstration developers were able to quickly begin constructing their demo by simply downloading these clients and using them with their existing data which was already stored in formats accessible to these tools.

The developers took on the task of integrating one internal tool themselves. The tool was a crew-vehicle interface (CVI) which provided a simulated cockpit of an advanced aircraft. Its user-interface consists of a set of panels, gauges, and artificial horizon displays. This client, written in C++, made use of the C API. The developers reported that the API was extremely effective in enabling hypermedia services within the simulated cockpit (for instance linking a gauge to its associated requirements). They reported that a significant amount of time was spent on preparing the CVI to handle anchor selections; once anchor functionality was added, incorporating Chimera's hypermedia services occurred quickly. This is in line with research on client integration by the last author [61]. Only a handful of Chimera's API operations were required to achieve the final CVI integration. These included REGISTER_VIEWER, REGISTER_OBJECT, REGISTER_VIEW, REGISTER_ANCHOR, GET_ATTRIBUTE, SET_ATTRIBUTE, and ADD_ANCHOR_TO_ACTIVE_LINK. In addition, only three event handlers had to be implemented: ACTIVE_LINK, LINK_TRAVERSAL, and SERVER_DISCONNECT.

The final demonstration hyperweb was compact. The hyperweb was composed of seven viewers, fifty-seven objects, fifty-seven views (one view per object). Over this set of information seventy-eight anchors were created which were related by thirty-six links. While the size of this hyperweb seems small, it nevertheless captured all of the relationships required by the technology demonstration. Chimera has been tested on hyperwebs three orders of mag-

nitude higher (e.g. tens of thousands of concepts) but only in lab situations, not in a real-use environment. The demonstration team did not collect metrics on the speed of link traversal, but provided anecdotal evidence that link traversals between active viewers were extremely fast, i.e. no significant delay was experienced by the user. Delayed link traversals (See Section 2.2.2) take longer to complete since a user must wait for the destination client to be invoked by the operating system. This example demonstrates Chimera's ability to provide hypermedia services in a heterogeneous software development environment, our original goal.

7 Limitations of the Chimera Approach

The Chimera approach brings with it some limitations and requirements. Our choice of a centralized approach limits the applicability of our techniques to small workgroups distributed across a local area network (LAN). However, by avoiding issues of distribution, our research was better able to focus on issues of heterogeneity. In addition, recent work [2] has extended Chimera to address issues of distribution by leveraging aspects of the WWW.

Another limitation involves the need for modification of legacy applications in order to insert hypermedia functionality. In other words, in order for our services to be of value, viewers must be programmed to utilize the Chimera API. Some existing hypermedia systems have succeeded in avoiding this limitation by exploiting an aspect of homogeneity in their environment. For example, the ABC system [54] uses X-window system mechanisms to insert a hypermedia-aware parent window over an application's window, while Microcosm [32] employs a universal viewer that exploits aspects of Microsoft Windows to add a Microcosm menu to unaware applications. The Chimera approach opts not to use these solutions in order to develop techniques that work in the presence of heterogeneity.

A third limitation of the Chimera approach is its reliance on integrated clients to provide information about the external world. For example, if a viewer decides to move one of its objects, it must notify the hypermedia system of the change. This allows Chimera to operate in environments where a variety of storage mechanisms are employed while not requiring any one of them. Again, this is consistent with our philosophy of supporting heterogeneity at all levels and, further, is necessary for supporting large-scale SDEs where complex configurations of storage mechanisms is the norm.

Chimera is limited to a single active hyperweb per session, with no ability to swap hyperwebs on the fly. This limits the use of hyperwebs as a grouping and abstraction mechanism. The new version of Chimera no longer relies on this mechanism; instead a graphical user-interface allows a user to specify a hyperweb via a URL [2].

The slow evolution of applications and data formats over time is another limitation to the Chimera approach. Since the external integration mechanisms provided by applications can change, a new revision can break Chimera integration code, requiring ongoing effort to accommodate the latest application revisions. Similarly, object data formats can change over time, sometimes to the point of being unreadable by later application revisions. This can cause older portions of long-lived hyperwebs to become unreadable if the data is not consciously migrated to newer formats.

8 Conclusions

In conclusion, Chimera makes a variety of contributions to SDEs and to hypermedia technology, including the satisfaction of all of the requirements described in Section 1. The essence of the contributions are key concepts and separations of concerns (architectural abstractions), provision of an effective set of server capabilities, and the demonstrated ability to simultaneously satisfy a wide variety of objectives in a single system.

Chimera's hypermedia concepts provide modeling power sufficient to map a diverse range of application-specific structures to their hypermedia counterparts. Neither the database(s) of objects nor the user interface of Chimera clients are part of Chimera or its concepts. The concepts are defined in a scalable and media-independent manner. Chimera's view concept in particular allows the modeling of the display capabilities of applications in which a single object can be displayed by more than one viewer or a single viewer can display more than one type of object. The view concept also provides the abstraction required to allow objects to be linked into a hyperweb without their modification. In addition, view-specific anchors enable a key separation of the conceptual architecture: object and anchor management from link management. Viewers manage anchors while Chimera manages 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. Chimera's conceptual architecture provides the abstractions necessary to enable the Chimera server to support clients built with diverse architectural styles ranging from single-threaded non-interactive link generators to multi-threaded highly-interactive end-user applications. This power comes from the distinction between clients and viewers and the clear demarcation of their responsibilities.

We have built an open hypermedia system, Chimera, to validate both the concepts and architecture. The Chimera server supports multiple, concurrent clients written in multiple programming languages, and the integration of commercial black-box tools (provided they support minimal interprocess communication) has been demonstrated. The implementation technique of providing equivalent APIs in multiple programming languages directly supports the heterogeneity found in SDEs and increases both the range and depth of client integrations.

Chimera's primary design goal is to provide hypermedia services to existing software development environments comprised of heterogeneous applications, data objects and repositories. This goal has been achieved, demonstrated by Chimera's use in multiple environments, including academic and industrial settings.

Acknowledgments

The authors would like to acknowledge Rebecca Grinter, Jonathan Grudin, Leysia Palen, Hadar Ziv, and the ECHT'94 reviewers for reading the conference version of this paper and providing comments. We would also like to acknowledge Roy Fielding for his help in understanding WWW-related issues. Uffe K. Wiil was kind enough to read an early draft of this paper and provide insightful comments while on sabbatical at the University of California, Irvine. We thank the reviewers for their careful critique of the initial submission. We also thank John Leggett for his extensive review of both the initial submission and subsequent revisions. His guidance was invaluable in preparing the paper for publication. In addition, the authors' gratitude is extended to the students in the software engineering project courses whose experiences fine-tuned Chimera and enabled the exploration of issues concerning the integra-

tion of legacy systems. Finally we wish to thank Alexander Wise, Yuzo Kanomata, and Joe Feise for their work integrating clients with Chimera.

The effort was sponsored by the Defense Advanced Research Projects Agency, and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

Source code for the released system is available on the Internet at <http://www.cs.colorado.edu/~kena/chimera/>.

References

1. Akscyn, R. M., McCracken, D. L., and Yoder, E. A. (1988). KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM,* 31(7)**:** 820-835.
2. Anderson, K. M. (1997). Integrating Open Hypermedia Systems with the World Wide Web. In *Proceedings of the Eighth ACM Conference on Hypertext*, pp. 157-166. Southampton, UK. April 6-11, 1997.
3. Anderson, K. M. (1999). Data Scalability in Open Hypermedia Systems. In *Proceedings of the Tenth ACM Conference on Hypertext*, pp. 27-36. Darmstadt, Germany. February 21-25, 1999.
4. Anderson, K. M. (1999). Supporting Industrial Hyperwebs: Lessons in Scalability. In *Proceedings of the 21st International Conference on Software Engineering*, pp. 573-582. Los Angeles, CA, USA. May 16-22, 1999.
5. Anderson, K. M., Taylor, R. N., and Whitehead, E. J., Jr. (1994). Chimera: Hypertext for Heterogeneous Software Environments. In *Proceedings of the Sixth ACM Conference on Hypertext*, pp. 94-107. Edinburgh, Scotland. September 18-23, 1994.
6. Ashman, H., Balasubramanian, V., Bieber, M., and Oinas-Kukkonen, H. (1996). The Second International Workshop on Incorporating Hypertext Functionality Into Software Systems, Washington D.C., USA.
7. Berners-Lee, T. (1996). WWW: Past, Present, and Future. *Computer,* 29(10)**:** 69-77.
8. Bieber, M. (1995). The First International Workshop on Incorporating Hypertext Functionality into Software Systems. Technical Report 95-10. New Jersey Institute of Technology.
9. Boudier, G., Gallo, F., Minot, R., and Thomas, I. (1988). An Overview of PCTE and PCTE+. In *Proceedings of the ACM SIGSOFT'88: Third Symposium on Software Development Environments*, pp. 248-257.
10. Bouvin, N. O. (1999). Unifying Strategies for Web Augmentation. In *Proceedings of the Tenth ACM Conference on Hypertext*, pp. 91-100. Darmstadt, Germany.
11. Campbell, B., and Goodman, J. M. (1988). HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM,* 31(7)**:** 856-861.
12. Carr, L. A., DeRoure, D. C., Hall, W., and Hill, G. J. (1995). The Distributed Link Service: A Tool for Publishers, Authors, and Readers. In *Proceedings of the Fourth International World Wide Web Conference*, pp. 647-656. Boston, MA, USA. December 1995. <http://www.staff.ecs.soton.ac.uk/~lac/dls/link_service.html>.
13. Conklin, J. (1987). Hypertext: An Introduction and Survey. *IEEE Computer,* 20(9)**:** 17-41.
14. Conklin, J., and Begeman, M. (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. In *Proceedings of the CSCW'88*, pp. 140-152. Portland, Oregon, USA.
15. Creech, M. L., Freeze, D. F., and Griss, M. L. (1991). Using Hypertext in Selecting Reusable Software Components. In *Proceedings of the Third ACM Conference on Hypertext*, pp. 25-38. San Antonio, Texas, USA. December 15-18, 1991.

16. Davis, H., Hall, W., Heath, I., Hill, G., and Wilkins, R. (1992). Towards an Integrated Information Environment with Open Hypermedia Systems. In *Proceedings of the Fourth ACM Conference on Hypertext*, pp. 181-190. Milano, Italy. November 30 - December 4, 1992.

17. Davis, H. C., Knight, S., and Hall, W. (1994). Light Hypermedia Link Services: A Study of Third Party Application Integration. In *Proceedings of the Sixth ACM Conference on Hypertext*, pp. 41-50. Edinburgh, Scotland. September 18-23, 1994.

18. Delisle, N. M., and Schwartz, M. D. (1986). Neptune: A Hypertext System for CAD Applications. In *Proceedings of the ACM SIGMOD'86*, pp. 132-142. Washington DC, USA. May 28-30, 1986.

19. Delisle, N. M., and Schwartz, M. D. (1987). Contexts—A Partitioning Concept for Hypertext. *ACM Transactions on Office Information Systems,* 5(2)**:** 168-186.

20. Dewan, P., and Choudhary, R. (1995). Coupling the User-Interfaces of a Multiuser Program. *ACM Transactions on Computer-Human Interaction,* 2(1)**:** 1-39.

21. Englebart, D. C. (1984). Authorship Provisions in AUGMENT. In *Proceedings of the COMPCON'84*, pp. 465-472. San Francisco, CA, USA. February 27 - March 1, 1984. <http://www.bootstrap.org/oad-2250.htm>.

22. Fernström, C., Närfelt, K.-H., and Ohlsson, L. (1992). Software Factory Principles, Architecture, and Experiments. *IEEE Software,* 9(2)**:** 36-44.

23. Ferrans, J. C., Hurst, D. W., Sennett, M. A., Covnot, B. M., Ji, W., Kajka, P., and Ouyang, W. (1992). Hyperweb: A Framework for Hypermedia-Based Environments. In *Proceedings of the ACM SIGSOFT'92: Fifth Symposium on Software Development Environments*, pp. 1-10. Washington D.C., USA.

24. Fielding, R. T., Whitehead, E. J., Jr., Anderson, K. M., Bolcer, G. A., Oreizy, P., and Taylor, R. N. (1998). Web-Based Development of Complex Information Products. *Communications of the ACM,* 41(8)**:** 84-92.

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

26. Grønbæk, K. (1994). Composites in a Dexter-Based Hypermedia Framework. In *Proceedings of the Sixth ACM Conference on Hypertext*, pp. 59-69. Edinburgh, Scotland. September 18-23, 1994.

27. Grønbæk, K., Bouvin, N. O., and Sloth, L. (1997). Designing Dexter-Based Hypermedia Services for the World Wide Web. In *Proceedings of the Eighth ACM Conference on Hypertext*, pp. 146-156. Southampton, UK. April 6-11, 1997.

28. Grønbæk, K., and Trigg, R. (1994). Design issues for a Dexter-Based Hypermedia System. *Communications of the ACM,* 37(2)**:** 40-49.

29. Haake, A., and Hicks, D. (1996). VerSE: Towards Hypertext Versioning Styles. In *Proceedings of the Seventh ACM Conference on Hypertext*, pp. 224-234. Washington DC, USA. March 16-20, 1996.

30. Halasz, F., and Schwartz, M. (1994). The Dexter Hypertext Reference Model. *Communications of the ACM,* 37(2)**:** 30-39.

31. Halasz, F. G. (1988). Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM,* 31(7)**:** 836-855.

32. Hall, W., Davis, H., and Hutchings, G. (1996). *Rethinking Hypermedia: The Microcosm Approach*. Kluwer Academic Publishers, Norwell, MA, USA.

33. Hicks, D. L., Leggett, J. J., Nürnberg, P. J., and Schnase, J. L. (1998). A Hypermedia Version Control Framework. *ACM Transactions on Information Systems,* 16(2)**:** 127-160.

34. Hoek, A. v. d., Heimbigner, D., and Wolf, A. L. (1996). A Generic, Peer-to-Peer Repository for Distributed Configuration Management. In *Proceedings of the 18th International Conference on Software Engineering*. Berlin, Germany. March 1996.

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

36. Kadia, R. (1992). Issues Encountered in Building a Flexible Software Development Environment: Lessons Learned from the Arcadia Project. In *Proceedings of the ACM SIGSOFT'92: Fifth Symposium on Software Development Environments*, pp. 169-180.

37. Leggett, J., and Schnase, J. (1994). Viewing Dexter with Open Eyes. *Communications of the ACM,* 37(2)**:** 77-86.

38. Malcolm, K. C., Poltrock, S. E., and Schuler, D. (1991). Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise. In *Proceedings of the Third ACM Conference on Hypertext*, pp. 13-24. San Antonio, TX, USA. December 15-18, 1991.

39. Marshall, C. C., Halasz, F. G., Rogers, R. A., and Janssen, W. C., Jr. (1991). Aquanet: A Hypertext Tool to Hold Your Knowledge in Place. In *Proceedings of the Third ACM Conference on Hypertext*, pp. 261-275. San Antonio, Texas, USA. December 15-18, 1991.

40. Marshall, C. C., and Shipman, F. M., III. (1997). Spatial Hypertext and the Practice of Information Triage. In *Proceedings of the Eighth ACM Conference on Hypertext*, pp. 124-133. Southampton, UK. April 6-11, 1997.

41. Marshall, C. C., Shipman, F. M., III, and Coombs, J. H. (1994). VIKI: Spatial Hypertext Supporting Emergent Structure. In *Proceedings of the Sixth ACM Conference on Hypertext*, pp. 13-23. Edinburgh, Scotland. September 18-23, 1994.

42. Maurer, H. (1996). *Hyper-G now Hyperwave: The Next Generation Web Solution*. Addison Wesley Longman. 635 pages.

43. Maybee, M. J., Heimbigner, D. H., and Osterweil, L. J. (1996). Multilanguage Interoperability in Distributed Systems: Experience Report. In *Proceedings of the Eighteenth International Conference on Software Engineering*. Berlin, Germany.

44. Meyrowitz, N. (1989). The Missing Link: Why We're All Doing Hypertext Wrong. Pages 107-114, *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*. MIT Press.

45. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. (1997). As We Should Have Thought. In *Proceedings of the Eighth ACM Conference on Hypertext*, pp. 96-101. Southampton, UK. April 6-11, 1997.

46. Nürnberg, P. J., Leggett, J. J., Schneider, E. R., and Schnase, J. L. (1996). Hypermedia Operating Systems: A New Paradigm for Computing. In *Proceedings of the Seventh ACM Conference on Hypertext*, pp. 194-202. Washington DC, USA. March 16-20, 1996.

47. Oinas-Kukkonen, H. (1997). Towards Greater Flexibility in Software Design Systems through Hypermedia Functionality. *Information and Software Technology,* 39(6)**:** 391-397.

48. Østerbye, K. (1995). Literate Smalltalk Programming Using Hypertext. *IEEE Transactions on Software Engineering,* 21(2)**:** 138-145.

49. Østerbye, K., and Wiil, U. K. (1996). The Flag Taxonomy of Open Hypermedia Systems. In *Proceedings of the Seventh ACM Conference on Hypertext*, pp. 129-139. Washington DC, USA. March 16-20, 1996.

50. Parunak, H. V. D. (1991). Toward Industrial Strength Hypermedia. Pages 381-395 in E. Berk and J. Devlin, Eds., *Hypertext/ Hypermedia Handbook*. McGraw-Hill.

51. Pearl, A. (1989). Sun's Link Service: A Protocol for Open Linking. In *Proceedings of the Second ACM Conference on Hypertext*, pp. 137-146. Pittsburgh, PA, USA. November 5-8, 1989.

52. Schnase, J. L., Leggett, J. J., and Hicks, D. L. (1991). HB1: Initial Design and Implementation of a Hyperbase Management System. Technical Report TAMU-HRL 91-003. Texas A&M University.

53. Schnase, J. L., Leggett, J. J., Hicks, D. L., Nürnberg, P. J., and Sánchez, J. A. (1994). Open Architectures for Integrated, Hypermedia-Based Information Systems. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, pp. 386-395. Weilea, HI, USA. January, 1994.

54. Smith, J. B., and Smith, F. D. (1991). ABC: A Hypermedia System for Artifact-Based Collaboration. In *Proceedings of the Third ACM Conference on Hypertext*, pp. 179-192. San Antonio, Texas, USA. December 15-18, 1991.

55. Streitz, N., Haake, J., Hannemann, J., Lemke, A., Schuler, W., Schütt, H., and Thüring, M. (1992). SEPIA: A Cooperative Hypermedia Authoring Environment. In *Proceedings of the Fourth ACM Conference on Hypertext*, pp. 11-22. Milano, Italy. November 30 - December 4, 1992.

56. Tarr, P. L., and Clarke, L. A. (1993). PLEIADES: An Object Management System for Software Engineering Environments. In *Proceedings of the 1993 ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp. 56-70. Los Angeles, CA, USA. December 7-10, 1993.

57. Taylor, R. N., Nies, K. A., Bolcer, G. A., MacFarlane, C. A., Anderson, K. M., and Johnson, G. F. (1995). Chiron-1: A Software Architecture for User Interface Development, Maintenance, and Run-Time Support. *ACM Transactions on Computer-Human Interaction,* 2(2)**:** 105-144.

58. Thomas, I. (1989). Tool Integration in the Pact Environment. In *Proceedings of the Eleventh International Conference on Software Engineering*. Pittsburgh, PA, USA.

59. Tichy, W. F. (1982). Design, Implementation, and Evaluation of a Revision Control System. In *Proceedings of the Sixth International Conference on Software Engineering*, pp. 58-67. Tokyo, Japan.

60. Trigg, R. H. (1983). A Network-Based Approach to Text Handling for the Online Scientific Community. Ph.D. Thesis. Department of Computer Science. University of Maryland.

61. Whitehead, E. J., Jr. (1997). An Architectural Model for Application Integration in Open Hypermedia Environments. In *Proceedings of the Eighth ACM Conference on Hypertext*, pp. 1-12. Southampton, UK. April 6-11, 1997.

62. Whitehead, E. J., Jr. (1999). Control Choices and Network Effects in Hypertext Systems. In *Proceedings of the Tenth ACM Conference on Hypertext*, pp. 75-82. Darmstadt, Germany. February 21-25, 1999.

63. Whitehead, E. J., Jr. (1999). Goals for a Configuration Management Network Protocol. In *Proceedings of the Ninth International Symposium on Systems Configuration Management*. Toulouse, France. September, 1999.

64. Whitehead, E. J., Jr., Anderson, K. M., and Taylor, R. N. (1994). A Proposal for Versioning Support for the Chimera System. In *Proceedings of the Workshop on Versioning in Hypertext Systems*, pp. 45-54. Edinburgh, Scotland. September 18-23, 1994.

65. Wiil, U. K., and Leggett, J. J. (1992). Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems. In *Proceedings of the Fourth ACM Conference on Hypertext*, pp. 251-261. Milano, Italy. November 30 - December 4, 1992.

66. Wiil, U. K., and Leggett, J. J. (1993). Concurrency Control in Collaborative Hypertext Systems. In *Proceedings of the Fifth ACM Conference on Hypertext*, pp. 14-24. Seattle, Washington, USA. November 14-18, 1993.

67. Wiil, U. K., and Leggett, J. J. (1996). The HyperDisco Approach to Open Hypermedia Systems. In *Proceedings of the Seventh ACM Conference on Hypertext*, pp. 140-148. Washington DC, USA. March 16-20, 1996.

68. Wiil, U. K., and Leggett, J. J. (1997). HyperDisco: Collaborative Authoring and Internet Distribution. In *Proceedings of the Eighth ACM Conference on Hypertext*, pp. 13-23. Southampton, UK. April 6-11, 1997.

69. Wiil, U. K., and Leggett, J. J. (1997). Hyperform: A Hypermedia System Development Environment. *ACM Transactions on Information Systems,* 15(1)**:** 1-31.

70. Yang, J. J., and Kaiser, G. E. (1998). JPernLite: An Extensible Transaction Server for the World Wide Web. In *Proceedings of the Ninth ACM Conference on Hypertext*, pp. 256-266. Pittsburgh, PA, USA. June 20-24, 1998.