SL BAR
Z
699
C3
no. 93-49

Hypertext for Heterogeneous Software Environments[1]

Kenneth M. Anderson[2], Richard N. Taylor[3], and E. James Whitehead, Jr.[4]
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92717-3425
FAX +1.714-856-4056

TR-93-49

September 6, 1993

Abstract

Emerging software development environments are characterized by heterogeneity: they are composed of diverse object stores, user interfaces, and tools. This paper presents an approach for providing hypertext services in this heterogenous setting. Central notions of the approach include the following. Anchors are established with respect to interactive *views* of objects, rather than as attributes of objects. Composable, *n*-ary links can be established between anchors maintained by different viewers for objects stored in distinct object bases. Viewers (and objects) may be implemented in different programming languages, as a client-server architecture can be used to support the approach. Multiple, concurrently active viewers can be utilized. The paper describes the approach and considers how interfaces to these hypertext services can be supported. Experience with a prototype implementation is described.

Keywords: heterogeneous hypertext, separation of concerns, software development environments

1. This material is based upon work sponsored by the Defense Advanced Research Projects Agency under Grant Number MDA972-91-J-1010. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.
2. E-mail: kanderso@ics.uci.edu. Tel. +1.714-725-2704
3. E-mail: taylor@ics.uci.edu. Tel. +1.714-856-6429
4. E-mail: ejw@ics.uci.edu. Tel. +1.714-856-5086

1 Introduction

Software development environments (SDEs) are used to develop and maintain a diverse collection of highly interrelated software objects [Kad92, BGMT88, FNO92, Tho89]. Large software systems may, for example, consist of multiple versions of requirements specifications, designs, prototypes, code, test cases, test results, test drivers, installation scripts, user's manuals, and so on. The connections between these components are many and complex. Establishing and exploring these connections are major tasks of development, program understanding, and maintenance. Providing hypertext capabilities in an SDE should allow an engineer to freely associate objects without regard to the type of those objects or where they are stored. These relationships can then be accessed at a later time through a convenient user interface which allows the engineer to easily navigate them [SZ92]. Yet while some excellent work in this area has taken place [Con87, Nie90, DS86, GS90, Pea89, SS91, SCG89, FHS$^+$92, KL91, CFG91], it is clear that no single system to date has effectively addressed all the technical challenges posed by this task.

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

Heterogeneous object editor & viewer support. SDEs contain a wide variety of tools for developing and manipulating objects. Different kinds of editors are used for different types of objects. SDEs also increasingly include multiple viewers of single objects, where each viewer presents different aspects of the object, perhaps using different depiction styles. Ideally all editors and viewers in an environment should be able to use hypertext services and respond to hypertext events. (From now on we will use the term "viewer" to denote tools capable of visually depicting an object and which may include interactive editing capabilities.)

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

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

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

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

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

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

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

This approach brings along with it some limitations and requirements. In order for our techniques and interfaces to be of value the viewers in the SDE must use the hypertext interfaces. The viewers are also responsible for maintaining (over time) the associations they make between the anchors they announce to the hypertext system and the objects for which they are a viewer. There is nothing to prevent them from doing this, of course, by inserting anchor information into the objects themselves, though that is a style which may well have the undesirable consequence of making that (modified) object usable only by that viewer (i.e. change the object's type such that other tools can no longer operate on it).

The discussion of our approach is in three parts. We first describe the concepts upon which the approach is based. A generic, conceptual architecture is then presented which indicates an effective strategy for implementing the concepts. Third, we describe our particular (subset) implementation and the experiences we have had with it.

Since heterogeneous environments are most often multilingual and distributed, the generic architecture and our implementation is serverized and a multilingual RPC mechanism (Q [MHLO92]) is utilized. Our prototype implementation, Chimera[1], utilizes the Pleiades object management system from the University of Massachusetts [TC93] for persistence of the server's data structures. To illustrate the concepts and the Chimera system, we discuss an application in which graphical views of a flight simulator's instrument panel are hyperlinked to statements in a requirements document maintained by FrameMaker®[2]. The serverized Chiron user interface development and management system from UCI [TJ93, KCTT91] is used as the graphical viewer of the instrument panel.

The remainder of the paper is organized as follows. The next three sections present the basic concepts, the conceptual architecture, and the prototype. Open design issues are discussed in Section 5. A comparison between the approach we offer and other systems can be found in Section 6. Finally, we conclude with Section 7.

2 Hypertext Concepts

This section describes the hypertext concepts we feel are necessary to support hypertext in a heterogeneous SDE. Our concepts include objects, viewers, views, anchors, and links. We provide examples of each concept. We then discuss the semantics that can be associated with anchors and links.

2.1 Basic Concepts

Objects. *Objects* are named, persistent entities whose internal structure and behavior (if any) is unknown and irrelevant to the hypertext system. An object's name does not change during its lifetime even if its contents or state does.

Example objects in a SDE include module interface definitions, source files, data flow diagrams, documents, process models, and shell scripts.

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

To participate in the hypertext system, a viewer must provide features for the creation and display of anchors (actually, anchor-view-events, described below) pertinent to the objects that it is displaying. A viewer is responsible for the specific, and potentially non-uniform, ways in which it supports this[3].

1. According to Merriam-Webster's 9th Collegiate Dictionary, "an individual, organ, or part consisting of tissues of diverse genetic constitution..."
2. FrameMaker is a registered trademark of Frame Technology Corporation.
3. This is potentially troublesome since the user has to remember how this hypertext functionality is invoked for each viewer [FHS$^+$92]. This is a design trade-off involving ease-of-use, open systems, and customized interfaces. We believe requiring a single, standard style to be too restrictive: that would prevent many existing editors from participating in the environment's hypertext system. On the other hand, it is possible to provide a set of capabilities that viewer creators can utilize which simultaneously simplifies the task of writing viewers and promotes uniform authoring, display, and interaction styles.

Views. The term *view* denotes 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.

Example. To illustrate these definitions, consider the specification, development, and testing within an SDE of a flight simulator which has graphical, simulated cockpit displays. The displays are to be driven off of a model of the aircraft's state, where that state is encapsulated in a variety of abstract data types (modules). Among the modules are altitude, pitch, roll, and throttle-setting. Requirements for the system are to be specified in English, and maintained in a FrameMaker document.

A module design editor (MDE) is used in the SDE to identify the modules, specify their interfaces and interconnections, and manipulate them graphically (at software design time)[4]. The design editor maintains a hierarchy of module specifications. At simulator run-time the instrument panel, throttle, joystick, and artificial horizon displays are to be created and managed by multiple Chiron viewers (*artists* in Chiron terminology). For instance, one Chiron viewer will display the altitude of the plane digitally; another will display the same altitude information via an analog gauge. Both viewers will utilize information obtained from the altitude-module. The artificial horizon viewer will present a synthesis of information from both the attitude-module and the roll-module.

Objects in this example include the altitude-module and the module hierarchy. *Viewers* in the example include FrameMaker, the MDE, and the Chiron artists used to drive the flight simulator's display. The MDE and the module hierarchy together define one *view*. Other views include (FrameMaker, requirements document), (digital-altitude-artist, altitude-module), and (artificial-horizon-artist, (attitude-module, roll-module)).

Anchors. *Anchors* are defined and managed by viewers. A viewer can tailor the notion of an anchor to the particular object that it is displaying. A special type of anchor is a *view event*: any effect (such as receipt of an audio signal or visual blinking) which acts like an anchor. When the effect occurs it is as if a traditional anchor had been highlighted. Since conceptually anchors and view events are equivalent, we refer to both with the term *anchor-view-event* or AVE. When an AVE is created, it is always created in the context of a view. Thus for each view defined in the system there is an associated collection of AVEs. Note that the hypertext system knows only of an AVE's existence; it does not know what the AVE actually refers to in the viewer.

Example. To continue with the example, the flight simulator's display artists could allow anchors to be defined on the analog altitude display (e.g. to allow linking to a statement of the requirements on the display). The MDE could allow the user to define an anchor which refers to the entire module hierarchy, or anchors which refer to specific modules within the hierarchy, but perhaps not on module interconnections. The MDE could also define view events for zooming in upon modules in its display, wherein the "anchor" is "highlighted" by the verbal command "pick zoom" and "traversed" by the verbal command "zoom". Traversal to that "anchor" from elsewhere would cause the viewer's display to be zoomed. This action could be linked, e.g., to

4. For an example of such an editor see [WP86].

requirements documents which state the requirements for that particular module.

Links. A link is a set of AVEs. The AVEs in a link can be members of any view.

The hypertext system is in charge of all aspects of link management. This includes creating and deleting links, and controlling all aspects of link semantics (discussed below). Viewers are responsible for informing the hypertext system of events upon AVEs (which may be members of links and which therefore could initiate link semantics). Similarly viewers are responsible for receiving "traversal" events from the hypertext system.

Link Composition. In our approach to hypertext for SDEs, links are first-class objects, just as well as a design diagram or a piece of source code. Consequently we can construct viewers for these objects, viz. link viewers. A link viewer paired with a link (object) forms a view. A link viewer can thus create anchors upon this view, and these anchors can be included in other links. If links are created between the anchors of two link viewers' views, this can be seen as composing the two links. We see this as a technique for the creation of arbitrarily large hyperwebs.

2.2 Semantics

Anchor semantics. Anchor semantics refers to the run-time behavior of an anchor. Viewers could provide a wide range of behavior for their anchors since they are first-class executing programs. Anchor behavior can include what user interactions with the viewer cause an anchor to highlight or initiate link traversal. An anchor could even perform calculations like keeping track of the number of times it has been selected. Anchor semantics could also specify what an anchor can do when it is the destination of a link traversal. When a user creates a link to a particular anchor, a good link creation protocol would display the list of actions the anchor can perform, enabling the user to select the desired behavior of the anchor[5].

Example. The MDE may decide to specify the semantics for an anchor on a module such that a single click causes the anchor to highlight and a double click causes the anchor to send a hypertext event to the hypertext system. When the user reaches a module via link traversal the MDE could highlight it or center it within the display, or both. "Highlight" and "Center" are examples of possible anchor commands defined by the MDE.

Link Semantics. Link semantic specifications provide information about a link and define its run-time behavior. The standard hypertext notion of link traversal is one kind of behavior that may be specified, but it is not required; the choice is fully up to the hypertext author.

Example. One motivation for a flexible way of specifying link behavior comes from consideration of what traversal means for an *n*-ary link. Consider the action that should take place in response to a user clicking on an anchor in the artificial horizon indicator in the flight simulator,

5. This is only possible, however, if the viewer decides to provide this functionality. The hypertext system can not require this behavior as a viewer's anchors are not under its direct control.

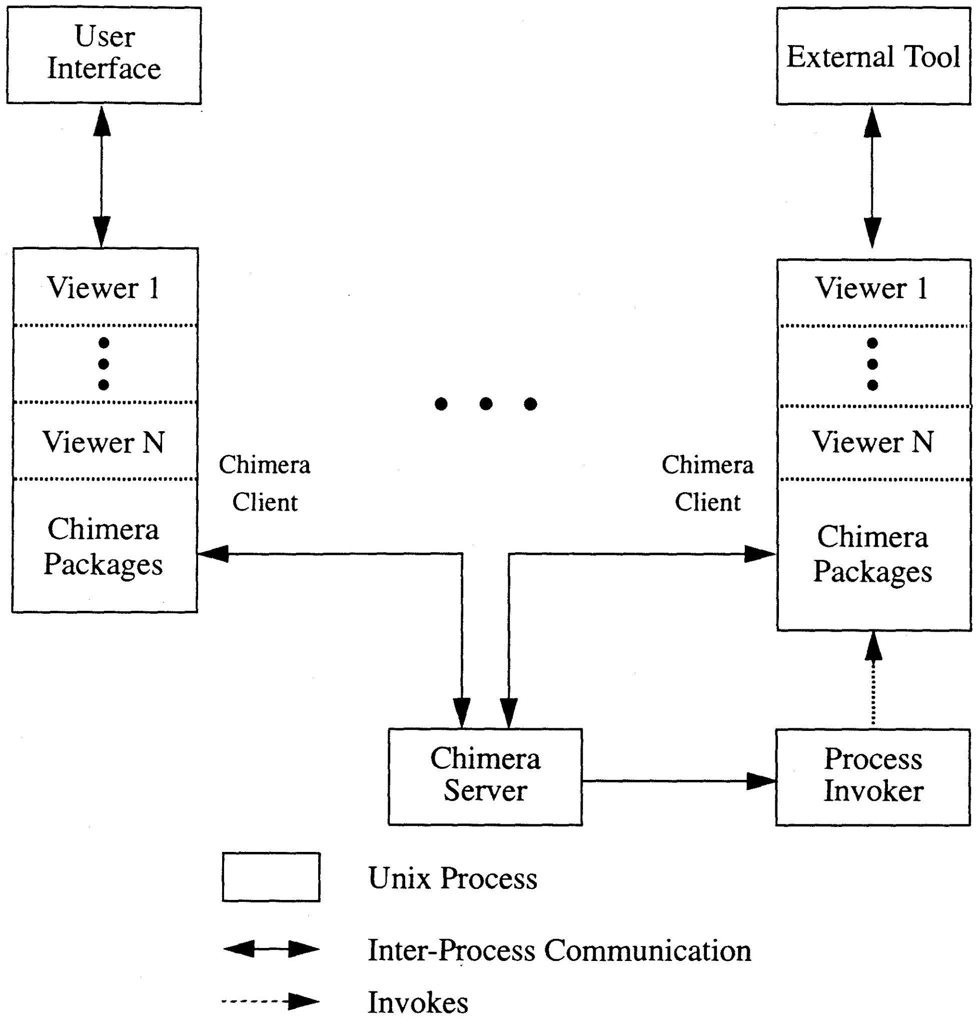

Figure 1: Example Configuration of Chimera Architecture

where that anchor is linked to anchors in three MDE depictions of modules in the module design. Perhaps the MDE should be invoked three times, each window highlighting and centering one of the three modules. Or perhaps the MDE should be invoked only once, with all three modules highlighted within the resulting display. Alternatively a menu could appear, showing the three associated anchors and allowing the user to select the desired module, at which time the MDE is invoked and displays the appropriate module.

3 A Conceptual Architecture

The preceding section outlined the key concepts of a hypertext system for SDEs. This section sketches a conceptual architecture which supports these concepts. This architecture adopts a serverized approach to providing hypertext functionality. We term this the Chimera architecture (Figure 1).

The key component of the architecture is the Chimera server which provides support for managing AVE and link definitions, tracking of active views, event routing, and persistence of hypertext entities. Viewers are clients of the Chimera server.

A serverized approach is adopted to help meet the challenges of a heterogeneous SDE in which there are many users. (This approach is in keeping with the experience of the Arcadia software environment project [Kad92].) With a serverized implementation, multiple users on different machines can access a hyperweb from a dynamically changing set of viewers; hypertext events originate in one process and travel to (potentially many) others. The use of a server supports a multilingual approach where tools written in one language can work with other tools, each of which may be written in different languages. The use of a server also keeps process sizes down, since code to manage hyperwebs is centralized in the server. (The Chimera server could be replicated to meet the needs of distributed environments, but we do not here show the additional components or discuss the additional issues thus raised.)

Chimera Server. The primary responsibilities of the Chimera server are as follows.

- Track all views, active or inactive, in the system. An *active view* is one in which the viewer is currently running and is displaying the specified object on the screen of a user's workstation. *Inactive views* are views that were active sometime in the past but were made inactive by the user quitting the viewer or by system shutdown. Inactive views are important because they may have AVEs defined for them. The server uses the Process Invoker to activate an inactive view.

- Manage the AVEs associated with each view in the system. This includes assigning AVEs a system-wide unique identifier that can be used by both the Chimera server and viewers.

- Manage the creation, manipulation, and traversal of links in the system. This includes executing link semantics in response to hypertext events.

- Receive and route hypertext events. A viewer will send a hypertext event to the Chimera server when a user has performed the actions required by an AVE's semantics to activate it. The Chimera server is responsible for retrieving all links that contain that AVE and executing the semantics associated with those links for that particular AVE.

- Manage the persistence of hyperwebs. Through the use of an object manager, the Chimera server must make persistent the information that it has about the entities that make up a hyperweb.

Process Invoker. The Process Invoker is used by the Chimera Server to invoke clients needed as the result of a link traversal.

Chimera Client. The key component of the client is the *viewers* which were described in the previous section. It should be noted that the concepts and architecture of Chimera place a significant burden upon the viewers. This burden is a result of the issues facing designers of open hypertext systems [Pea89, CFG91].

To summarize, a viewer is responsible for:

- definitions of the concepts "object" and "view". For instance, a word processor (viewer) might decide that each of its documents is a distinct object and a new view occurs when the word processor displays a new document. A gauge of a flight simulator, however, may only have one object, the value that it is displaying, and thus only have one view.
- a definition of the concept "anchor" (AVE). This includes identifying what elements of a view can have anchors created on them, how anchor creation occurs, how the presence of an anchor is indicated, what semantics can be assigned to an anchor, and how a user activates link semantics from an anchor.
- a mapping function from an AVE id (received from the Chimera server at the time the AVE is created) into a specific region or object of its display or a view-event that it can perform. This mapping may be simple and all the information needed could be stored with Chimera, or it may be complex and require (e.g.) the viewer to associate with each object a file that helps reconstruct and maintain the mapping each time the object is displayed.
- communicating with the Chimera server at run-time. This is accomplished by the viewer linking in the Chimera Packages, which hide from the viewers all of the details concerning connecting, sending messages, and receiving messages from the Chimera server.

External Tools/UIMSs. Viewers in a Chimera client may directly interface with the user, may require the use of external tools, or may use a user-interface management system (UIMS) to present their interface. Chimera does not dictate how viewers actually communicate with humans.

4 An Implementation Architecture and Demonstration

We have constructed a prototype to help validate the concepts presented in Section 2 and the conceptual architecture presented in Section 3. It has been applied in a demonstration situation based on the running example of Section 2.

4.1 Overview

The architecture of the demonstration is presented in Figure 2. It consists of a Chimera server, a Process Invoker, three Chimera clients, and one external tool, running in their own processes and communicating via RPC mechanisms. The clients are a flight simulator, a link control panel, and FrameMaker. The flight simulator and the link control panel make use of the Chiron UIMS to present their interfaces to the user. FrameMaker provides its own user-interface directly to the

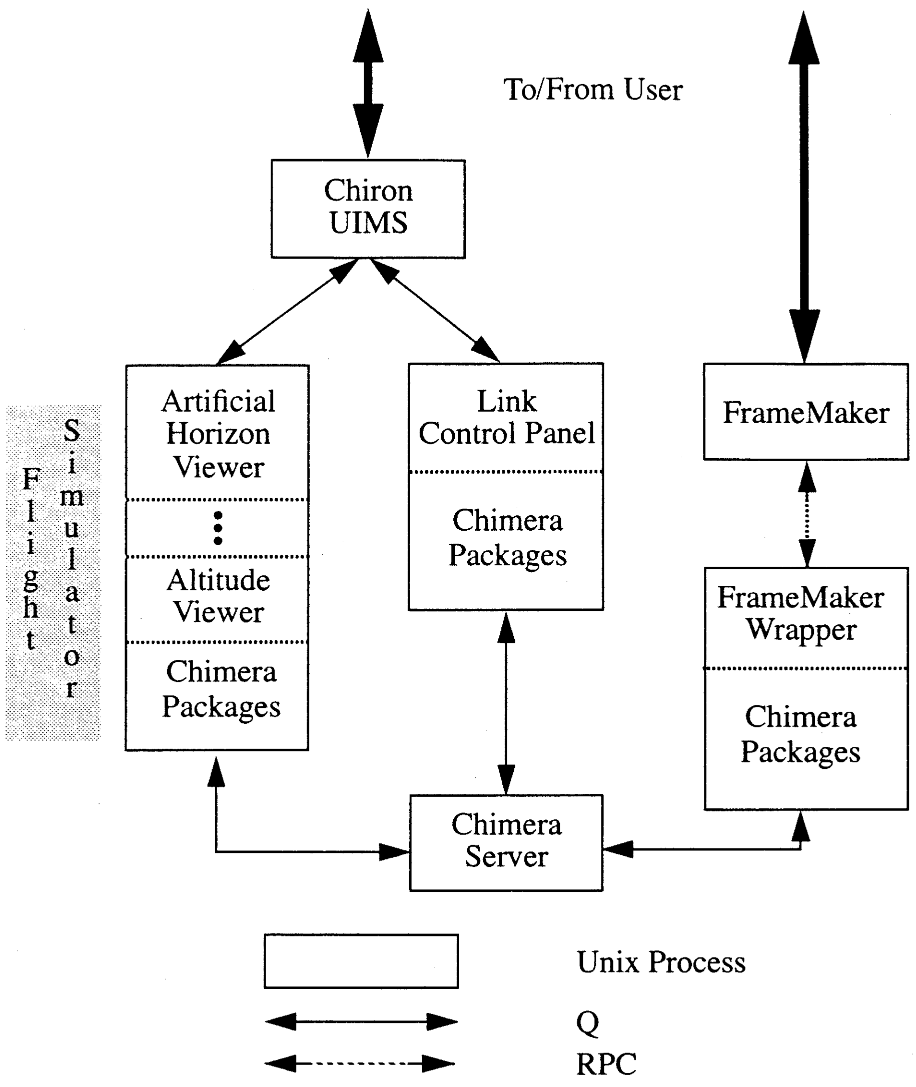

Figure 2: A specific Chimera Architecture

engineer.

4.2 Chimera Server

The Chimera server coordinates access to a set of abstract data types (ADTs) that implement Chimera's primitive hypertext concepts. The ADTs provide creation, maintenance, and query operations on hyperwebs. Thus, viewers which make use of these ADTs via the Chimera server can create a hyperweb, populate it with views and links, and save it for later use. Existing hyperwebs can be opened and modified.

In order to use Chimera, a user must invoke the server and pass it the name of a hyperweb. If the hyperweb does not exist the server creates a new one, assigning it the specified name, otherwise the server calls the ADTs to read in the persistent data for that hyperweb restoring its state from a previous invocation. The server then creates a file containing the name of the machine that it is running on and its process id. At this point the server is ready to accept client connections.

A Chimera client, upon start-up, reads the file that the Chimera server created and uses the information contained within to establish a connection. We use the Q multilingual interprocess communications package to pass messages between the server and its clients. Once a connection has been established clients can query and modify the state of the hyperweb by making calls on the server. The server responds to such calls by decoding the message that it receives, calling the appropriate ADT, and passing any return values back to the client. The server always sends at least one return value back to the client for each call, and this value is an indication of whether the ADT call completed successfully.

In addition to coordinating access to the ADTs, the Chimera server also sends messages to its connected clients. Currently it can send two types of messages. The first is notification of a link traversal. When a link's semantics dictate that traversal to a set of anchors is required, the server finds all links which contain the specified anchors and sends notifications to all clients containing viewers that can display views which contain the destination anchors of the link traversal. If a destination anchor resides in a view requiring an inactive viewer, the server calls the Process Invoker requesting activation of a client containing the required viewer. The second type of message deals with link creation. It is used to coordinate a protocol which allows a user to add multiple AVEs to a new link.

4.3 Chimera Client

A Chimera client consists of four major components: one or more viewers, a hypertext interface package, a callback package, and a client configuration and registration package. Viewers were discussed in detail in the previous section. We now present details about the remaining components.

Hypertext Interface Package. The *hypertext interface package* provides an interface which allows viewers to register themselves and their objects, views, and anchors with the server. There are also routines which allow viewers to open and close hyperwebs, query for information about a hyperweb, begin and end link creation, modify a hyperweb, and instigate link traversals. Viewers can access these routines in response to user actions or the viewer may be a tool which is automatically creating views, anchors, and links on objects within its domain in fulfillment of some artifact construction task. The interface package accepts calls from viewers and then packs the information passed in as parameters into a message which is shipped via Q to the server. It then waits for a return message from the server, unpacks the status of the ADT call invoked and if success is indicated passes any return values back to the viewer. If the status indicates failure, the interface package raises an exception which the viewer can handle in a variety of ways.

Callback Package. The *callback package* is used to notify viewers of messages sent from the Chimera server. The callback package accepts messages from the server and forwards them to the appropriate viewer instance in the client. This package also handles the details of invoking new viewer instances within the client as needed by link traversal events.

Client Configuration and Registration Package. The *client configuration and registration package* allows clients to connect and disconnect from the server, request the server to shutdown, and track viewer instances in a client.

4.4 FrameMaker

Since a major goal of Chimera is to support hypertext in a heterogeneous environment, it is important to demonstrate that Chimera can support third-party software. We chose to integrate the word processor FrameMaker[6] [Fra90, Fra91] into Chimera due to its built-in hypertext support and RPC interface. FrameMaker is also, at present, one of the most frequently integrated tools in a software environment; integrating Chimera with it therefore constitutes a minimal test of integration with non "Chimera-friendly" applications.

Within the Chimera architecture, FrameMaker is a viewer of document objects. As such, FrameMaker itself is only one half of a Chimera client, the other half being the three Chimera packages, contained within the FrameMaker Wrapper (described below). The combination of FrameMaker and the FrameMaker Wrapper is a complete Chimera client.

Since FrameMaker implements hypertext unusually as compared to the typical system described in [Con87], we give an overview of FrameMaker hypertext. Creating hypertext anchors in FrameMaker is accomplished by inserting a marker into the document. Each marker is a tuple containing its location, marker type, and a text field; with these elements, markers can be used for index entries, glossary entries, and general comments. When a marker is used as a hypertext anchor, the type is set to 'hypertext', and the text field is filled with FrameMaker hypertext commands. Some of FrameMaker's hypertext commands allow for the creation of an alert box, for jumping to another anchor, for creating a popup menu, or for sending a message out of FrameMaker via RPC. FrameMaker's hypertext commands are only executed when a user traverses a FrameMaker link. The advantage of using markers for hypertext is that hypertext features are included within FrameMaker's existing architecture. The disadvantage is that anchors and FrameMaker links are not separated. For example, anchors cannot be given a label except by using the 'newlink' command (a misnomer), and FrameMaker links cannot be traversed except after placing the document in 'read-only' mode (through the intuitive key sequence 'ctrl-r F l k').

Despite its drawbacks, FrameMaker does have a vital component necessary for integration with Chimera, the 'message' command. When a 'message' command is executed, it sends an RPC message to the RPC server specified within the command.

While the 'message' command is sufficient for links that begin in a FrameMaker document, it is not sufficient for links that terminate there. In this case, the page of the FrameMaker document containing the link terminus must be displayed. This can only be achieved by issuing

6. We integrated Chimera with FrameMaker version 3.1X.

the FM_RPC_GOTO_LINK command via RPC from the FrameMaker Wrapper. Unfortunately, FM_RPC_GOTO_LINK cannot jump to anchors with the 'message' command, as they are unlabeled according to FrameMaker. Since the only way to label an anchor is via the 'newlink' command, and since an anchor cannot have a label and a 'message' command simultaneously, a 'newlink' labeled anchor must be created at the same location as the marker containing the 'message' command.

Given these mechanisms, integrating FrameMaker with Chimera required two separate programming steps. The first step was the development of a FrameMaker macro which creates an anchor within a FrameMaker document such that it contains a unique Chimera anchor identifier. This macro provides the viewer-required functionality of anchor creation. The second step was the creation of a FrameMaker Wrapper which translates between Chimera and FrameMaker hypertext idioms. These steps are described in detail below.

4.4.1 FrameMaker Anchor/Link Creation

There are three requirements on the creation of a Chimera hypertext anchor within FrameMaker:

- A marker with a command of 'message' must be created. This requires importing a unique Chimera anchor identifier into FrameMaker, a non-trivial task.

- At the same text location, another marker with a command of 'newlink' must be created. This requires the same Chimera anchor identifier.

- The creation of these markers must occur quickly, in a way transparent to a FrameMaker user.

The 'message' marker is needed to inform the FrameMaker Wrapper when an anchor has been selected. We use a 'message' command of the following format:

```
message FM_Wrapper [chimera_anchor_id]
```

When the anchor is selected from within FrameMaker, the FrameMaker Wrapper (known as the RPC server 'FM_Wrapper') receives the file name of the FrameMaker document, and the [chimera_anchor_id]. This is then passed along to the Chimera server.

The 'newlink' marker is needed to label the anchor. We use a 'newlink' command of the following format:

```
newlink [chimera_anchor_id]
```

When a Chimera link is traversed which contains this anchor in its set of AVEs, the FrameMaker Wrapper is informed of the traversal. The FrameMaker Wrapper then sends an

FM_GO_TO_LINK command to FrameMaker causing the page containing the anchor to be displayed.

To make the links transparently, two FrameMaker macros were employed. The first macro contacts the FrameMaker Wrapper which contacts Chimera for the unique anchor identifier. Using this identifier, the FrameMaker Wrapper generates a second macro into a file. The first macro finishes by reading in the second macro and then executing it. This second macro then creates the two hypertext markers containing the 'message' and 'newlink' commands.

Since there is no standard method for transferring control to an outside program during macro execution, the first macro employed FrameMaker's output filter feature. When the first macro needed to transfer control to an outside program, it instead wrote the current file to a temporary file with a special extension. The output filter file (a shell script) picks up this extension and executes another program instead of an output filter. This new program contacts the FrameMaker Wrapper via RPC which in response contacts Chimera for the unique anchor identifier, and then writes the second stage macro to a file. While it sounds like anchor creation using these macros is slow, in reality creating anchors takes approximately 5 seconds. This is much faster than a person could accomplish when creating the link manually (10-15 seconds), the standard alternative in FrameMaker.

This description is not meant to be an example of the right way to program; the authors are fully aware it is a hack. Soberingly, this appears to be the state-of-the-practice in integration with many commercial tools. Excruciatingly missing are a command to send an arbitrary message outside of FrameMaker from within a macro, and a command to start a separate process from within a macro. These two features would have dramatically eased the difficulty of integrating Chimera with FrameMaker.

4.4.2 FrameMaker Wrapper

There are three main services that must be provided between FrameMaker and Chimera, these being:

- RPC messages originating in FrameMaker must be translated and forwarded to the Chimera server.
- Q messages originating in the Chimera server must be translated into appropriate FrameMaker actions.
- When requested, a FrameMaker macro must be created using a unique anchor identifier requested from the Chimera server. This macro (described above) creates the 'message' and 'newlink' markers.

These services are provided by the FrameMaker Wrapper program.

At start-up, the FrameMaker Wrapper establishes a communications link to FrameMaker using RPC, and a communications link to Chimera using Q. After start-up, the FrameMaker Wrapper waits for messages from either FrameMaker or Chimera. When a FrameMaker message is received (it will always be a FM_RPC_MESSAGE), indicating the user wants to traverse a Chimera link associated with the selected anchor, the FrameMaker Wrapper receives the unique anchor identifier labeling the anchor within Chimera. The FrameMaker Wrapper forwards the anchor identifier along to the Chimera server.

When events are received from the Chimera server, they are translated into appropriate FrameMaker commands. For the link traversal event, the FrameMaker Wrapper receives the anchor identifier for the destination anchor. The FrameMaker wrapper contacts the Chimera server for the view associated with this anchor. With this view information, the FrameMaker Wrapper next requests the object name associated with the view. This is the FrameMaker document file name. The FrameMaker Wrapper now sends FrameMaker the FM_RPC_GO_TO_LINK command with the anchor identifier and document file name. For the link creation event, the FrameMaker Wrapper places all open FrameMaker documents into read-only mode so their anchors may be selected. When link creation is finished, the documents are returned to their original editable or read-only state.

Despite the difficulties encountered, Chimera was able to integrate FrameMaker within its architecture. It thus provides one important data point in demonstrating the ability of the Chimera architecture to support hypertext within a heterogeneous environment.

4.5 Undemonstrated Features

There are two straightforward aspects of the hypertext concepts which are not demonstrated in the current system: links between hyperwebs and link viewers.

Link Viewers. Links are first class objects in the conceptual architecture. As such a viewer could be built to display them, and thus produce a view upon which anchors can be created. These anchors could then be linked together forming a hierarchical structure of links. We see this as an important technique in facilitating the construction of large hyperwebs. Such viewers can also function as browsers so often employed as navigational aids in other hypertext systems [CFG91, SS91, DS86]. We have not yet built such viewers due to resource constraints.

Links between hyperwebs. Our demonstration system does not allow links between hyperwebs because our server can only open one hyperweb at a time (due to a restriction in its underlying object management system). We see such links as a useful extension since we could then investigate topics such as integrating hyperwebs built by different individuals, reusing partial hyperwebs, and distributed hyperweb implementation architectures.

5 Open Design Choices

5.1 Anchor and Link semantics

Currently our demonstration system has no notion of anchor or link semantics beyond selection and traversal. We defined anchor and link semantics to be declarative and procedural information that is associated with an anchor or link. The declarative information provides information about the link or anchor (e.g. keywords to facilitate browsing and navigation) while the procedural information defines the link or anchor's run-time behavior. The majority of hypertext systems allow only declarative information about links to be defined. They provide this functionality using attribute-value pairs. [DS86, FHS^+92] These systems usually define a set of standard attribute-value pairs that the system uses for its own internal purposes, but in addition allow the user to define additional pairs that can be added to the standard set or used by external tools. Procedural information is usually not stored[7]. If a system stores procedural information it is usually via an attribute-value pair which holds the name of a shell script to execute upon link traversal.

Anchor semantics are usually not implemented. For most systems, this is an artifact of their internal hypertext model, that is, anchors are viewed as attributes of a link rather than a first class object with a wide range of attributes and behaviors[8].

We have not included semantics into our prototype as yet because we are still investigating techniques to best implement/represent them. Attribute-value pairs have been successful in other systems but we question whether they provide a wide enough range of behaviors to support a heterogeneous SDE. Hyperweb uses an interpreted scripting language to provide powerful and flexible link semantics, but we question whether a full blown language is necessary or most effective for web authors. PROXHY provides yet another alternative by placing this information into object-oriented classes but this raises issues of how much information should be placed in the class as opposed to the application using that class.

5.2 Process Invoker Improvements

Amy Pearl does an excellent job of presenting issues facing designers of open hypertext systems [Pea89]. She mentions the policy decisions that must be made when invoking new viewers at run-time. Assuming a networked computing environment, upon what machine should new processes be invoked? Do you always invoke a viewer when it is needed or do you notify the user that the requested link traversal will be expensive in terms of time and memory and query whether to proceed? Currently our Process Invoker invokes a new process on the machine that it is running on and it does so without notifying/querying the user. We plan to augment it to be more intelligent about where it runs a new process. For instance, we could let the user specify a machine by an environment variable or we could scan for a machine with a low load average. We also plan to implement a query/notification mechanism to bring the user into the loop. Here is a potential use

7. Hyperweb (see below) is an exception with its interpreted scripting language.
8. PROXHY (see below) with its anchor class is an important exception.

for link semantics, when a link is created the user could specify how to handle traversals of this kind.

6 Related Work

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

6.1 Neptune

Neptune [DS86] is an early, layered hypertext system developed at Tektronix Laboratories, built to support engineering information systems. The bottom layer is a transaction-based server called the Hypertext Abstract Machine (HAM) which provides persistent storage for binary objects. Above HAM are the application layers; these consist of programs that manipulate and interpret the hypertext data stored in the HAM. Binary links, created by application layers, are attached as offsets into the binary data. HAM supports the versioning of both nodes and links.The user interface is on top; it is used to browse hypertext data or control programs in the application layers. While the differences between Neptune and Chimera are many and obvious, Neptune is notable for being one of the earliest systems which attempted to bring the power of hypertext to the problems of engineering development environments.

6.2 VNS

The Virtual Notebook System (VNS) was built at the Baylor College of Medicine to support collaborative biomedical research via distributed hypertext services in a heterogeneous computing environment [SCG89]. The VNS is realized by a set of work group servers (WGSs) distributed throughout a network. Each WGS contains a Sybase relational database which is used to store text, graphics, and link information. Users typically store all their data with the WGS on their local machine but can also access information stored on a WGS on another machine. The VNS Gatekeeper is used to integrate external tools with VNS, whereby information from these tools is copied and stored in a WGS. One interesting aspect of VNS is that while users may share data, they do not share links. Thus two users can see the same page but view different links. Link information for each user is stored separately from the data that makes up a page. After a page is constructed dynamically, a user's link information for that page is retrieved and displayed.

VNS and Chimera share the same goal, providing hypertext services to a heterogeneous environment, but they differ in approach. Despite the distribution of the hyperweb over the various work group servers, VNS, in contrast to Chimera, still requires that all information be stored in a Sybase database under its control. Integration in VNS is concerned with providing the ability to copy information out of an external tool and into its database. At that point, the external tool is taken out of the loop; VNS handles the display of the data from then on. Integration in Chimera is concerned with getting a viewer to communicate with the Chimera server. Chimera makes no attempt to display a viewer's objects.

6.3 Sun's Link Service

Sun's Link Service was a commercial product which defined a protocol for an extensible and loosely coupled hypertext system [Pea89]. An application integrates with the Link Service (LS) by loading in a link library which implements the protocol. This library allows communication with the LS control process, which facilitates communication between applications which are link-aware. Applications provide call-back procedures to the LS so that they can receive link-related messages. Links are binary and are stored in a database managed by the LS control process.

Chimera and the Link Service are similar in that integration is achieved by having external tools use a library package to communicate with a hypertext server. They are also similar in that links essentially relate unique identifiers that applications use to map to application specific objects. Like the Link Service, Chimera stores its link information in a separate database which is under its control. Finally Chimera and LS are alike in that they do not attempt to manage the display of objects in the hyperweb, leaving that function to the applications which own them.

Chimera and the Link Service differ in several aspects. The Link Service asks for an identifier that refers directly to an application object. The identifiers which Chimera relates refer to anchors created on a view of an object, not the object itself. This allows a Chimera viewer to store anchor information separately from the object (or objects) to which it refers. Links are hidden in Link Service; that is, an application cannot retrieve links and manipulate them. This is not the case with Chimera, where links can be retrieved by an appropriate viewer and displayed in a variety of ways. This allows anchors to be created on those views and for those links to then be further linked or composed. No such functionality is present in LS. Lastly, Chimera's links are *n*-ary.

6.4 DIF

DIF [GS90] is a hypertext system developed at USC for managing software lifecycle documents. DIF supports the creation of binary hypertext links on textually-stored objects. The objects are stored as Unix files under DIF-managed RCS control; links are stored as relations in the Ingres relational database system. All objects are fully managed by DIF; in this sense it is a "classical" hypertext system which employs its own database and requires all objects participating in the hyperweb to use this one database. Similarly DIF controls the user interface, wherein links can be created and traversed. Unix shell scripts can be associated with links to invoke actions associated with traversing a particular link. DIF lets users associate keywords with nodes and links to facilitate browsing of the documents. It allows external tools to be integrated into the system, but the tool appears in a window that DIF controls.

6.5 PROXHY

PROXHY [KL91], which stands for PRocess-oriented, Object-based, eXtensible, HYpertext system, was developed at Texas A&M University. It defines a four-segment architecture: hypertext, communications protocol, application, and back-end. The back-end, used to store objects and hypertext information, can be implemented by an object server, database, or file-system. The application segment consists of one or more programs which may be running on different machines. An application is defined as one or more classes; each class sends and receives messages using the communications protocol. Each class can be implemented as a single process, allowing an application to execute on several different machines concurrently. The communications protocol implements message passing via interprocess communication services. The hypertext segment consists of anchor and link classes (each of which may be implemented as separate processes). PROXHY defines a simple hypertext model: anchors connect application objects to links. Links connect two or more anchors together.

PROXHY compares with Chimera in that links are *n*-ary, data can be stored in heterogeneous databases if the back-end is appropriately implemented, and external tools can be integrated into the system if they can be integrated via the communications protocol. One key difference between Chimera and PROXHY is that Chimera associates anchors with views, while PROXHY associates anchors with objects. This enables Chimera, when combined with a viewer mechanism such as Chiron, to provide greater flexibility in displaying an anchor, supporting the notion of several viewers (concurrently) providing different views of the same object, where the anchors and their presentation are view-specific (and this all separated by Chiron from any application code). This is similar, though, to PROXHY's notion of a context. In PROXHY, depending on the context, different sets of anchors and links will be made available to an application displaying the object. Another key difference, of probably greater importance, is that it does not appear that in PROXY links can be considered "ordinary" objects such that anchors can be defined upon them and thereby participate in hierarchical webs. Thus Chimera appears to have a scalability advantage.

6.6 Hyperweb

HyperWeb [FHS+92] was developed at Vista Technologies Inc. and is now known as PCTE Workbench. It is a framework that accesses a PCTE-compliant database of software objects at the back end, and coordinates a set of tools at the front end. It is built around an interpretive scripting language (an object-oriented Lisp dialect), which implements hypermedia and is open and customizable. Handlers can be associated with links and nodes which provide a way of specifying anchor and link semantics. HyperWeb's links are binary and are stored invasively as attributes of objects in the PCTE database, where all information must be stored. While the differences with Chimera are clear, the interesting aspect of HyperWeb is that the relationship mechanism of an SDE's object management system is used as the vehicle for supporting hyperlinks. This raises the key question of what the relationship is, or should be, between an SDE object manager's relations and the links supported by an SDE's hypertext system. We briefly discuss this issue in the next section.

7 Summary and Future Research

The essence of the approach we have presented is to allow viewers to determine the definition of anchors and to put responsibility for the management of links on a hypertext server. Allowing viewers to define anchors permits a variety of types of anchors to be defined, and they may be implemented in non-invasive ways. The API (application program interface) of the server can be accessed by multilingual RPC mechanisms. We believe that the set of concepts and architecture described go a long way towards satisfying the hypertext needs of open, heterogeneous software development environments. Our focus to date has been on achieving a feasibility demonstration and proof-of-concept.

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

8 References

[BGMT88] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas. An overview of PCTE and PCTE+. In *Proceedings of ACM SIGSOFT'88: Third Symposium on Software Development Environments*, pages 248–257, Boston, November 1988. Appeared as SIGPLAN Notices 24/(2) and Software Engineering Notes 13/(5).

[CFG91] Michael L. Creech, Dennis F. Freeze, and Martin L. Gris. Using Hypertext in Selecting Reusable Software Components. In *Proceedings of Hypertext'91*, San Antonio, Texas, December 1991.

[Con87] Jeff Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987.

[DS86] N. Delisle and M. Schwartz. Neptune: A hypertext system for CAD applications. In *Proceedings of the ACM SIGMOD'86*, pages 132–142, Washington, DC, May 1986.

[FHS+92] James C. Ferrans, David W. Hurst, Michael A. Sennett, Burton M. Covnot, Wenguang Ji, Peter

Kajka, and Wei Ouyang. HyperWeb: A Framework for Hypermedia-Based Environments. In *Proceedings of ACM SIGSOFT'92: Fifth Symposium on Software Development Environments*, Washington D.C., December 1992.

[FNO92] Christer Fernström, Kjell-Haakan Närfelt, and Lennart Ohlsson. Software factory principles, architecture, and experiments. *IEEE Software*, 9(2):36–44, March 1992.

[Fra90] Frame Technology Corporation, San Jose, California. *FrameMaker Reference*, September 1990.

[Fra91] Frame Technology Corporation, San Jose, California. *Integrating Applications With FrameMaker*, June 1991.

[GS90] Pankaj K. Garg and Walt Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, 7(3):90–98, May 1990.

[Kad92] R. Kadia. Issues encountered in building a flexible software development environment: Lessons learned from the Arcadia project. In *Proceedings of ACM SIGSOFT'92: Fifth Symposium on Software Development Environments*, Tyson's Corner, Virginia, December 1992.

[KCTT91] Rudolf K. Keller, Mary Cameron, Richard N. Taylor, and Dennis B. Troup. User interface development and software environments: The Chiron-1 system. In *Proceedings of the Thirteenth International Conference on Software Engineering*, pages 208–218, Austin, TX, May 1991.

[KL91] Charles J. Kacmar and John J. Leggett. PROXHY: A Process-Oriented Extensible Hypertext Architecture. *ACM Transactions on Information Systems*, 9(4):399–419, October 1991.

[MHLO92] Mark J. Maybee, Dennis H. Heimbinger, David L. Levine, and Leon J. Osterweil. Q: A multi-lingual interprocess communications system for software environment implementation. Submitted for publication, 1992.

[Nie90] Jakob Nielsen. *Hypertext and Hypermedia*. Academic Press, Inc., San Diego, California, 1990.

[Pea89] Amy Pearl. Sun's Link Service: A Protocol for Open Linking. In *Proceedings of Hypertext'89*, Pittsburgh, Pennsylvania, November 1989.

[SCG89] Frank M. Shipman, III, R. Jesse Chaney, and G. Anthony Gorry. Distributed Hypertext for Collaborative Research: The Virtual Notebook System. In *Proceedings of Hypertext'89*, Pittsburgh, Pennsylvania, November 1989.

[SS91] John B. Smith and F. Donelson Smith. ABC: A Hypermedia System for Artifact-Based Collaboration. In *Proceedings of Hypertext'91*, San Antonio, Texas, December 1991.

[SZ92] Dani Steinberg and Hadar Ziv. Software Visualization and Yosemite National Park. In *Proceedings of the Twenty-Fifth Annual Hawaii International Conference on System Sciences*, January 1992.

[TC93] Peri Tarr and Lori A. Clarke. Pleiades: An Object Management System for Software Engineering Environments. In *ACM SIGSOFT'93: Proceedings of the Symposium on the Foundations of Software Engineering*, Los Angeles, California, December 1993. To appear.

[Tho89] Ian Thomas. Tool Integration in the Pact Environment. In *Proceedings of the Eleventh International Conference on Software Engineering*, Pittsburgh, PA, May 1989.

[TJ93] Richard N. Taylor and Gregory F. Johnson. Separations of concerns in the Chiron-1 user interface development and management system. In *Proceedings of the Conference on Human Factors in Computing Systems*, Amsterdam, April 1993. Association for Computing Machinery.

[WP86] Anthony I. Wasserman and Peter A. Pritcher. A graphical, extensible integrated environment for software development. In *Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 131–142, December 1986. Appeared as SIGPLAN Notices/ 22(1), January 1987.