The Design of an Interactive Online Help Desk in the Alexandria Digital Library

Robert Prince Jianwen Su* Hong Tang Yonggang Zhao
Department of Computer Science
University of California
Santa Barbara, CA 93106
{robertp, su, htang, yzhao}@cs.ucsb.edu

ABSTRACT

In large software systems such as digital libraries, electronic commerce applications, and customer support systems, the user interface and system are often complex and difficult to navigate. It is necessary to provide users with interactive online support to help users learn how to effectively use these applications. Such online help facilities can include providing tutorials and animated demonstrations, synchronized activities between users and system supporting staff for real time instruction and guidance, multimedia communication with support staff such as chat, voice, and shared whiteboards, and tools for quick identification of user problems. In this paper, we investigate how such interactive online help support can be developed and provided in the context of a working system, the Alexandria Digital Library (ADL) for geospatially-referenced data. We developed an online help system, AlexHelp!. AlexHelp! supports collaborative sessions between the user and the librarian (support staff) that include activities such as map browsing and region selection, recorded demonstration sessions for the user, primitive tools for analyzing user sessions, and channels for voice and text based communications. The design of AlexHelp! is based on user activity logs, and the system is a light-weight software component that can be easily integrated into the ADL user interface client. A prototype of AlexHelp! is developed and integrated into the ADL client; both the ADL client and AlexHelp! are written in Java.

Keywords

Online help desk, online support, collaboration, user interface, digital library.

1 INTRODUCTION

Online customer service systems such as "Call Centers" or "Customer Care Centers" have been widely used, e.g., homebanking, telephone registration, online shopping, airline ticket booking, and digital libraries, etc. However, most of those systems lack sufficient capabilities of interactive communication mechanisms necessary for providing online customers with more sophisticated support and help. The emergence of the World Wide Web provides some new options for the user support and help problem because multi-media information such as images, audio/video, and animation can be easily presented in addition to traditional text. On the other hand, because of this and rapid advances in technology, the software systems for existing applications are becoming more and more complicated and at the same time new applications are being quickly developed. Such applications include distance learning, computer-based training, electronic commerce applications, and more. We believe that a good package of online help facilities can not only compensate for deficiencies of user interfaces, but also will make very complex services easier to learn and use. At the core of the help facilities lies the cooperating software systems or modules enabling communication between the user and service (human) agents. The communication may be based on paradigms of specific software systems, in addition to the ASCII and voice channels. For example, in the context of a digital library for georeferenced information both the user and the information specialist at the help desk (at a distant site) should be able to have a view of the *same* interface session so that the user can see what the information specialist does exactly. Similar situations can also occur in sessions for seeking technical support for a software systems, making online reservations and orders, etc. In this paper, we present the design and implementation of an online interactive help system in the context of a digital library for geospatially referenced information.

In particular, we report our experiences in the development and integration of an online help desk system AlexHelp!, for the Alexandria Digital Library (ADL) [FFL+95]. In ADL, the user starts a *session* by initiating an HTTP-based connection to the ADL server. During one session, the ADL client software has a relatively complex user interface that allows users to browse maps, zoom in/out, construct queries (searches), and manipulate the results of queries. The potential users of ADL do not necessarily have much experience in accessing computers, and dealing with spatial information; their knowledge about software systems may be very limited. Our objective is to provide "just in time" online (collaborative) help facilities for the users of the ADL system.

The AlexHelp! system provides the following functionality. The user can request to establish a connection to a help desk. Once the connection is made, both the user and the information specialist will share a common view of the user's ADL session. The information specialist is now able to perform various actions on the ADL client interface, including

*Supported in part by NSF grants IRI-9411330 and IRI-9700370.

To appear in *Proc. ACM International Joint Conference on Work Activities Coordination and Collaboration, WACC '99, Feb. 22–25, San Francisco, CA 1999*

those mentioned above, and all actions happen simultaneously at both sides. In addition to such synchronized session support, AlexHelp! also allows the presentation of prerecorded demonstration sessions to the user, and showing or replaying user sessions to the help desk.

AlexHelp! is designed using user activity logs (i.e., session logs), and is a light-weight software component that can be easily integrated into or layered on top of the ADL user interface client. Our design is different from many other collaborative systems that are developed from scratch with collaborative environment tools. This is because the ADL system is already operational, and using tools like Habanero [CGJ+98] or Sun's JSDT [JSDT] would require rewriting much of the ADL client code and could significantly increase the cost of communication. Moreover, the advantage of our approach is that it saves time by not repeating the work already done.

As pointed out in [BM94], a straightforward approach to constructing collaborative environments that combines different communication mechanisms together will not necessarily result in a good collaborative environment. Bergmann and Mudge found in their experiment that the successful use of their system requires much logistics and support. However, what we find is that for a certain class of collaborative applications, careful design makes it possible to significantly reduce such costs by automating most repetitive tasks.

Most of our work on AlexHelp! is focused on the ADL client. Generally speaking, AlexHelp! is quite different from typical groupware [Gru94b]. The collaborative model that AlexHelp! supports is much simpler, since collaboration only occurs between users of the system and the help desk. In a collaborative session, one participant is designated the *master* and the others are *slaves*. The master controls all the slaves' views of the client interface (essentially). During the session, either the user or the help desk can be the master and the master "token" can be passed among participants. Such transferring must be under the control of the help desk. The collaborative actions supported during the session are not very complex, and the nature of the connection between master and slaves is almost stateless. But clearly such an application in the context of ADL is also very representative for various other online assistance services.

In the CSCW research community, several typologies have been proposed. In terms of the Grudin's typology [Gru94a], CSCW models are categorized in two dimensions, space and time, as follows.

		TIME		
		Same	Different but Predictable	Different and Unpredictable
PLACE	Same	Meeting Facilitation	Work Shifts	Team Rooms
	Different but Predictable	Tele-, Video-, Desktop Conferencing	Electronic Mail	Collaborative Writing
	Different but Unpredictable	Interactive Multicast Seminar	Computer Boards	Workflow

The AlexHelp! system supports a combination of these models: in the space dimension, the collaboration can occur at different locations, either predictable or unpredictable; in the time dimension, it can be at the same time or at a different but predictable time.

Our work is also related to synchronized web browsers such as [WR94, FLF94]. However, AlexHelp! is based on and integrated into the ADL client software rather than relying on web browsers; the design issues and implementation techniques are thus quite different.

This paper is organized as follows. Section 2 presents the motivation of AlexHelp!. Section 3 outlines the overall design and system architecture. Section 4 focuses on the session log file, the key component used in AlexHelp!. Section 5 briefly summarizes and discusses some implementation issues. Conclusions and future work are included in Section 6.

2 MOTIVATIONS

In this section, we first give a motivating example in the context of the Alexandria Digital Library system. We then discuss the general concept of online interactive help systems or help wizards and their applications.

The Alexandria Digital Library (ADL) [FFL+95] is a digital library for geospatially referenced data including maps, images, and text. ADL's graphical user interface is a client application implemented in Java; the user runs it locally (on the user's workstation), connecting to the ADL via the Internet. The ADL client includes the following components (see Figure 1).

1. *Map Browser Window* (window 1 in Figure 1).

   The Map Browser window allows users to interactively pan and zoom a two-dimensional map of the globe to locate their area(s) of interest. In addition, the user can select one or more areas on the map that may be used to constrain queries. The map is also used to display the spatial extent of the items retrieved from a query.

2. *Search Window* (window 2 in Figure 1).

   The Search window allows users to set query parameters; parameters include choosing the collection(s) to search, location (i.e., coordinates from the Map Browser), type of information (maps or aerial photographs from the the catalog or hydrologic features from the gazetteer), date range, free text, and other options.

3. *Workspace Window* (window 3 in Figure 1).

   The Workspace window displays and allows manipulation of query results. A query history is also maintained here, allowing the ADL client to be returned to the state of a prior query. A *Scan Display* or metadata browser (bottom part of the workspace window) displays brief object metadata. Brief metadata includes title, format, access information, spatial/temporal references, and a small scale version of the original image if available. From this window, full metadata and access information can be requested.

4. *Help Window* (window 4 in Figure 1).

   The ADL client's Help Window is a typical example of today's software help systems. It shows, depending on what component in the graphical user interface is selected, help topics specific to that component. ADL has a wide range of potential users, many of whom may not have experience interacting with complex software systems. The Help Window is of some assistance. A tutorial and walkthrough are also made available through the ADL homepage[1]. Clearly something more is required, however, since it is still difficult for inexperienced users

[1] `http://www.alexandria.ucsb.edu/adljigi`.

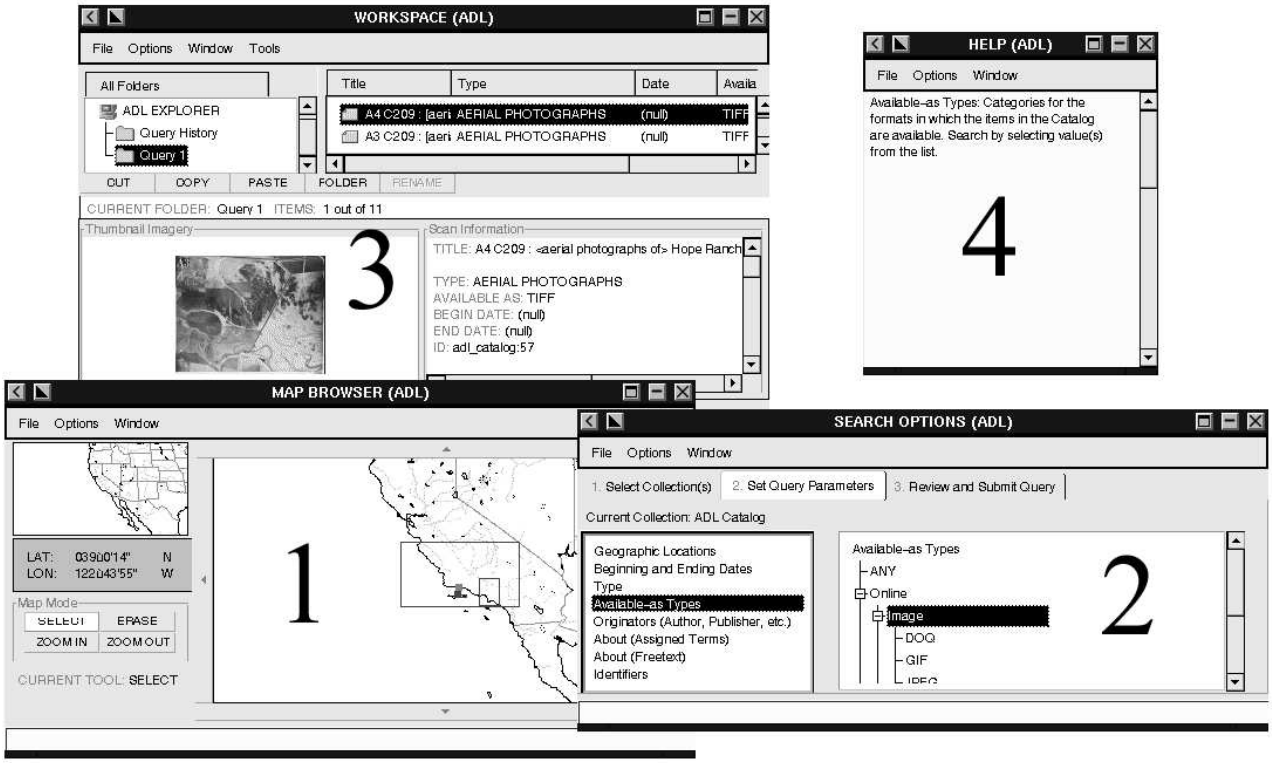

Figure 1: The ADL client/User Interface

to know how to use the system for their particular purposes.

Consider the following example. Suppose M is a student majoring in political science. M is working on a research paper about the civil population in the Santa Barbara area, and one of her friends suggests that she can get some useful data from ADL. She follows her friend's suggestion and launches an ADL client. Unfortunately, she quickly finds herself frustrated. She tries the help system, but is still confused and unable to learn how to effectively utilize ADL.

She needs a quick answer. If she were in the university library, she could ask one of the librarians to help her find the information she's looking for.

With AlexHelp!'s extension to the ADL help system, she has more choices than blindly searching for answers from the static help system. The following are possible alternatives.

- She could try the online tutorial (Demo Sessions). She could download some pre-recorded sessions and replay them (the Demo Session player is discussed in Section 4). These sessions could demonstrate to her the basic operation of the ADL client.

- If the online tutorial doesn't help, she could try the Help Wizard. She would be guided through a process where she answers questions based on the nature of her problem. The Help Wizard would then direct her to several Demo Sessions that would hopefully help solve her problem.

- If the first two methods don't solve her problem, she could connect to the help desk at ADL for an interactive help session. M and the help desk could communicate through text-based chat, phone, or online audio channels while the information specialist guides M through the process of constructing a query and evaluating the information returned from the query. M would actually see, on her own ADL client, exactly what she would need to do because the help desk controls M's client while they talk.

Clearly, these help desk services can help the student in this example. It is conceivable different kinds of users may prefer some of the service to others. To provide the help desk services in the above example, the following capabilities need to be developed.

- *Collaborative sessions*

  By collaborative session, we mean that the ADL clients on both sides (the user and the help desk) are "connected"; that is, one client is in control of the session and sends its actions to the other client, which mirrors them on its graphical user interface. The clients can switch roles during the session, allowing the user and help desk to participate in a rich exchange of information.

- *Session replay (demo sessions)*

  Support staff are not always immediately available. Instead of forcing users to wait until they can contact an information specialist, AlexHelp! uses the concept of demo sessions. Demo sessions are examples of using the ADL client that may be replayed on the client itself, showing the user successful ways to use the client.

  To support this concept, we developed a Session Player. This gives the user VCR-style control over a session; the

user may pause, rewind, start and stop sessions. Different sessions show the user how to perform tasks of varying complexity, and for those that are more complex, it is very useful to be able to pause a session and replay subtle or difficult portions.

In addition, the Session Player allows users to record their own sessions, which may be sent to support staff for offline analysis or detection of usage patterns.

- *Multimedia communication*

  Interaction between the user and the help desk should not be restricted to manipulating or observing the ADL client. It seems reasonable to assume that a network connection of some type has been established between the help desk's and user's ADL clients; it could be utilized for more than just synchronizing the clients. Collaborative help systems should provide additional methods for communicating; text-based chat, voice, and even video could be used to allow participants to communicate. Shared whiteboards fit the paradigm as well, but in this case the "whiteboard" is the application itself – the reason the participants are connected is to explain the use of the application.

In addition to the above capabilities, it is also very useful in such a context to provide:

- *Call waiting and forwarding*

  In order to serve the users fairly and efficiently, our design calls for a way to queue help requests when there is more than one user that wants to participate in a collaborative session.

  Call waiting means that when the user requests to connect with the help desk for a collaborative session, if the help desk is not immediately available, the user is notified that there will be some period of waiting, and information such as expected wait time and the reasons for the current delay might be made available to the user.

  Another method to support interactive help is call forwarding: if the current information specialist is not proficient in the particular area in which the user needs help, the user's connection may be "task switched" to another information specialist that has the proper expertise.

- *Multicast collaborative sessions*

  With multicast support, more than one slave mode client can be synchronized with a single master mode client — a desirable feature to support scheduled instructional/ demonstration sessions or distance learning.

3 THE AlexHelp! SYSTEM

Based on the analysis of the requirements for online assistance in ADL and the design of the ADL client, we decided that the interactive online help facilities in AlexHelp! should be designed as an independent component that is easily integrated with the ADL client. Even though the AlexHelp! system is relatively simple and does not include everything listed at the end of Section 2, it succeeds as a good example of adding collaborative functions on top of the single-user version of a software package. The feedback from the development and user evaluation groups within the ADL project shows that our system provides a feasible and efficient way to support interactive online help in ADL.

In this section we discuss the main issues in developing the AlexHelp! system, including functionality, design, and architecture.

3.1 Functionality

Our prototype AlexHelp! provides the following facilities:

- *Collaborative session establishment and operation*

  A simple dialog window is used to initiate the connection between two ADL clients. Once the connection is made, AlexHelp! runs in one of two modes:

  – *Slave (receive) mode*

    In this mode, the ADL client uses AlexHelp! to listen for incoming messages from the other client; when a message (a remote event) is received, the remote graphical user interface event is "replayed" or duplicated on the Slave client. The user of the Slave client is unable to change the client's state.

  – *Master (send) mode*

    In this mode, the AlexHelp! system sends local graphical user interface events to the Slave ADL client, where they are replayed. The state of the Master ADL client is thus reflected in the Slave ADL client.

  In particular, the ADL client Map Browser window is able to replay operations such as "Zoom In", "Zoom Out", "Pan", etc.

- *Session replay*

  We also developed a mechanism for replaying recorded sessions. As we discuss below and in the next section, every ADL client records its operations in a log file. AlexHelp! utilizes the logs in both Collaborative Sessions and Session Replay.

3.2 Design

Consideration of an appropriate collaborative model for the AlexHelp! system was driven in part by our goal of quickly developing a prototype that could be used to demonstrate the possibilities of extending and developing online help systems. While we had access to the ADL client's code base, we also had the constraint that we could not change the basic functionality of the client; in other words, the client had to perform in all other respects exactly as it had before. Thus we were faced with adding multi-user ability to an existing single-user application that was not originally designed with multi-user access in mind, a task Grudin recommends as a reasonable way to approach building groupware [Gru94b].

Not being able to fundamentally alter the client prevented us from using frameworks or toolkits for building collaborative environments like NCSA's Habanero [CGJ+98] or Sun's JSDT (Java Shared Data Toolkit) [JSDT]. Using a framework such as JSDT would entail in essence rebuilding the application. On the other hand, including the ADL client in an application that comprises a collaborative environment like Habanero would require the user to run the ADL client as a "component application" from within the new environment, making the ability to participate in a collaborative help session dependent on this new environment. This was deemed unacceptable for two reasons. First, the ADL client is meant to be a stand-alone application; it is the library user's interface to the ADL. Second, the ADL client has a fairly small footprint in terms of memory and network resources; including it in a larger framework would impose additional resource requirements.

Our specification calls for a model of collaboration best described as turn taking or token passing: semantically, only one of the user or the help desk should have control of both

clients at a given time. During the period of collaboration, one of the clients is in the "slave" or receive mode, and the other is in the "master" or send mode. Considering this simple model and our desired features within the context of a collaborative help system for ADL, there is no need to use complex concurrency control or shared object models [GM94, MR91]. The user interface of the client, say A, that is currently in receive mode is simply "locked;" it may receive and interpret events or messages from the other client, say B (i.e., the remote client), but the user is unable to change client A's state. When the clients trade roles, client A takes charge and the events generated at A are sent and duplicated at client B.

We realized early on that the ADL client log facilities already in place were rich enough to duplicate the actions of one client at another client separated by space and/or time. This not only simplified design of the client-to-client communication, but also makes it easy to generalize the client-to-client model into a one-to-many "multicast" model. Rather than capturing and packaging system or user interface events for transmission, which is a potentially complex endeavor even in a syntactically sweet language like Java [JAVA], the log entries can simply be sent to the remote client as they are generated. The use of session logs in AlexHelp! is discussed in detail in Section 4.

3.3 Architecture

Initially, it was intended that the help desk and the ADL user would run the same program (on their respective workstations). It was thought that the symmetry of both participants using the same application (i.e., one set of code) was good design. As the project progressed, it was decided that the help desk should be responsible for controlling a help session, and the necessary features for control should be built into the help desk's client. During a help session, the information specialist may turn control over to the remote user (i.e., the help desk's client becomes the slave); the information specialist should have the ability to recover control of the session if desired. However, because AlexHelp! will be rewritten along with the next version of the ADL client, development of a single application continued. The help desk's session management features were not implemented in our current prototype.

AlexHelp! consists of three layers: the ADL Client layer, the Event Handler layer, and the Communication layer (Figure 2). These three layers are described below.

3.3.1 ADL Client

The ADL client is the primary interface to the ADL for users of the library. In its first incarnation, the ADL client was a Java applet suitable for running within a Java-capable browser such as Netscape's Navigator. The current version is a stand-alone application, built entirely in Java. Subsequent development is intended to produce both stand-alone and applet versions.

As mentioned in Section 3.2, we were not at liberty to fundamentally change the ADL client's design or functionality. We chose therefore to layer the additional functionality we required on top of the ADL client (see Figure 2). Note that in our prototype, the ADL client communicates directly with both the Event Handler layer and the Communication Layer; Figure 2 reflects our design rather than our current implementation. In an iterative software development model, we would choose in our next iteration (post prototype) to restrict the ADL client to interacting with a single layer, the Event Handler layer, in order to keep the module dependencies clean.

3.3.2 AlexHelp! Event Handler Layer

The Event Handler is layered directly on top of the ADL client. Its primary function is to receive incoming remote events from the Communication Layer, and replay or reproduce them on the local ADL client.

We noted in Section 3.2 that the existing logging mechanism in the ADL client is rich enough in content to allow us to duplicate or replay remote events on a local ADL client. This allowed us a very simple design for the Event Handler layer: when a remote event is received in the form of a log entry (a string of ASCII characters), it is parsed to determine what event should be triggered locally in the ADL client. The Event Handler then directs the local ADL client to perform the event. For example, if the remote event is a pan (horizontal or vertical movement) in the remote ADL client's Map Browser window, the Event Handler notifies the local ADL client, providing the direction to pan (north, south, east, west), and also the distance.

This design for handling remote events is easily generalized for a distance learning or "multicasting" scenario: a single information specialist or instructor runs an ADL client in master mode, and the client's events are broadcast to a group of ADL client users, all of whose clients are operating in the slave mode. Each ADL client in slave mode receives and handles remote events in the manner described above.

3.3.3 AlexHelp! Communication Layer

In the discussion of the collaborative model chosen for the AlexHelp! system we mentioned JSDT, an example of a framework for building collaborative applications that provides abstractions for typical components of such systems such as "channel," "client," and "server." Had we the opportunity of building AlexHelp! (and also the ADL client) from scratch, it is likely that we'd have chosen such a framework to ease the typically troublesome task of properly designing and building a system in which distributed communication is central to operation. However, it was decided that given the time and limited flexibility in terms of altering the ADL client, it would be easier and faster to use simple TCP/IP (sockets) to implement client-to-client communication. The choice was made easy because of Java's inclusion of networking abstractions as part of its core libraries [JAVA]. The `java.net.*` library provides objects that encapsulate network connections that are either connection-oriented or connectionless, and also provides a model for sending the same message to multiple recipients (useful for our distance-learning "multicast" model). The current implementation of the AlexHelp! system provides client-to-client connections using connection-oriented Java sockets and a turn-taking slave/master model; the next version of AlexHelp! will include the distance learning mode as well.

AlexHelp!'s architecture is structured in layers so that it would be relatively easy to replace one of the layers with an alternative implementation — hopefully, it would be easy to the point that replacing one layer would require no modifications to the adjacent layers. The Communication Layer is meant to be no exception. In addition, we foresee utilizing the Communication Layer to apply additional communication mechanisms: text-based chat, voice, video, etc.

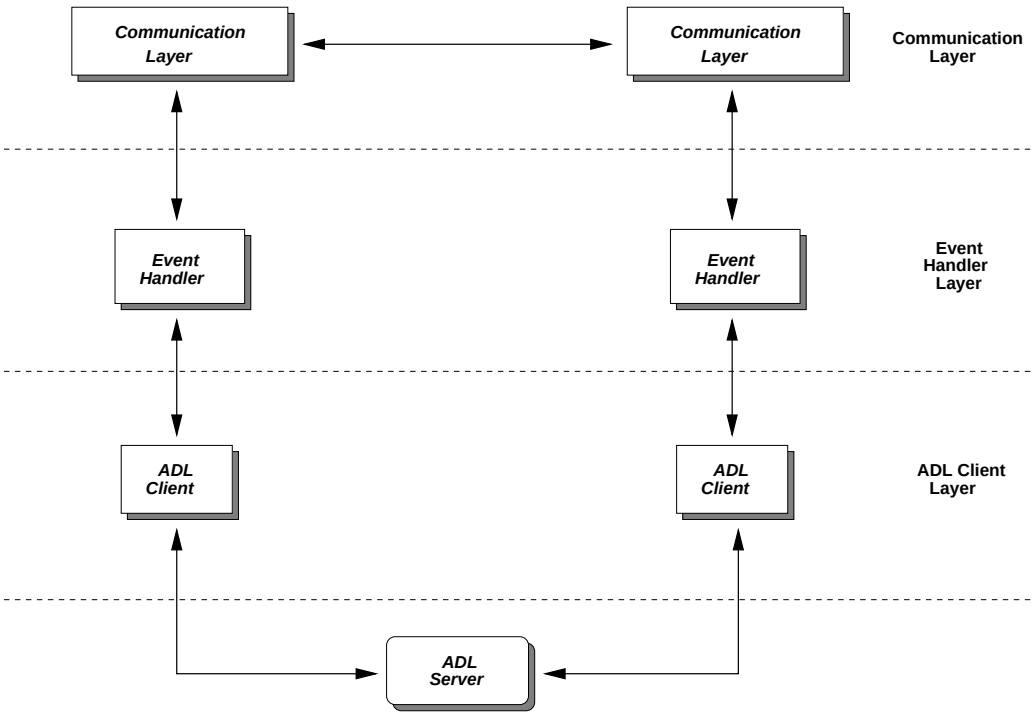

Figure 2: The Architecture of AlexHelp!

4 SESSION LOGS

In this section we discuss the central technique of using session logs in developing synchronized sessions in AlexHelp!. We also illustrate that session logs facilitate the design of session replay and give a general discussion on applying this technique in other contexts.

Several of our early design meetings included personnel from the ADL development team. Among the many invaluable things we learned from them was the fact that they built a simple user activity log system into the client. As part of the event handling in the ADL client, certain user- and system-generated events are logged into an ASCII text file, which may then be analyzed after the fact.

The original intent of the ADL development team was to use the activity logs for analysis of user activity and usage pattern discovery. We saw that if the logs were augmented to include more information about each event, we could then use the log entries to implement coordination of two physically separate ADL clients. Further, the logs could be viewed as a persistence mechanism, giving us a method for "session replay". We felt that this (persistence of an entire user's session) would be crucial to integrating AlexHelp! into the ADL's existing help system.

Our experience developing AlexHelp! using session logs as both a way to forward local events to a remote client and as a persistence mechanism showed us that for a certain class of applications, log files can be instrumental in rapidly and simply augmenting existing help systems (or building new ones), and also adding multi-user capability to single-user applications.

In Section 4.1, we discuss the format of the ADL client's log files, and explain some example log entries. In Section 4.2, we discuss the use of logs in controlling collaborative sessions in AlexHelp!. Section 4.3 examines how log files are used to implement VCR-style control of Demo Sessions (log replay). Section 4.4 discusses general application of of the log-based technique.

4.1 The ADL Client Log File

The ADL client log consists of event summaries, each of which makes up a single log entry. The order in which entries appear in the log maps loosely to the order in which they occur; there may be variation between two logs recording identical sessions due to timing differences that can be tracked to the effects of multithreading; i.e., a system event may occur before a user-generated event, but the user event is logged first due to scheduling. We did not experience unexpected behavior related to such scheduling discrepancies.

In addition to information recording an event (see examples below), a log entry also includes information identifying the particular session and client that generated the event. In a log entry, the information of interest (information that allows an event to be reconstructed from the log entry) actually makes up very little of the entry. In many cases, less than about one third of a log entry is of interest. For brevity, the other information (such as session or client identification) has been removed from the log entries used in the discussion below.

The following are examples of the ADL client's log entries, and comments related to using the logs with AlexHelp!.

- `client action | Map Mode:  ZOOM IN`

  This log entry is among the simplest types. It is generated when the user clicks with the mouse on the button labeled "ZOOM IN" on the Map Browser window. Similar log entries are made when the user clicks on

the buttons labeled "ZOOM OUT," "SELECT," and "ERASE."

- `client action | New Extent:`
  `-205.3 -40.56 -25.29 49.44`

  This log entry occurs when the user clicks the mouse in the map on the Map Browser window when the selected mode is either ZOOM IN or ZOOM OUT. "New Extent" identifies this as a map resizing event. The parameters correspond to the lower left and upper right corners of a rectangular geographic region which defines the new extent of the map in the Map Browser window. The first two numbers are the latitude and longitude of the lower left corner; the second two are the latitude and longitude of the upper right corner.

- `client action | Query Region(s) Modified:`
  `-127.64 43.33 -113.19 31.20`

  This log entry occurs when the selected mode is SELECT and the user clicks and drags the mouse in the map on the Map Browser window. This action creates a selection box, which defines a rectangular geographic region, as in the previous log entry. The parameters are again the latitudes and longitudes of the lower left and upper right corners of the geospatial region, which may be used to constrain queries. Selection boxes may overlap each other.

- `client action | Query Region(s) Modified:`
  `-124.74 41.78 -123.02 38.64`
  `-125.08 38.68 -120.09 36.27`
  `-122.59 36.96 -120.35 34.98`
  `-121.30 35.41 -115.02 33.95`
  `-120.40 34.51 -115.58 32.10`

  In this log entry, a selection box is again drawn; in this case, however, several selection boxes already existed. Log entries for modifications to Query Regions (selection boxes), like the log entry above, simply enumerate all of the selection boxes, whether the change was adding or removing a selection box. The log entry above shows the five currently existing selection boxes.

- `client action | Map: Button Pressed: west`

  This log entry shows a "pan" event (horizontally or vertically repositioning of the map in the Map Browser window).

4.2 Logs in Collaborative/Synchronized Sessions

The ADL client's log facility was central to our design and implementation of synchronized help sessions. Our specification calls for the ability to capture, distribute, and reproduce graphical user interface events. Capturing such events as they occur can be troublesome at best, and at worst impossible. Although Java aids the programmer in this type of endeavor (see Section 3.2), it was clear to us that in light of the fact that the session log mechanism was already in place, utilizing it would permit implementation to proceed much faster than dealing with (intercepting) events at the graphical user interface level.

Augmenting the ADL client in order to send log entries to a remote client was a simple matter of hooking into the module responsible for writing log entries to a file. Acting on the log entries that have been received from a remote client entails using the ADL client's public application programming interface (API) to set its internal state, which triggers any corresponding changes in the graphical user interface.

As an example, consider again two ADL clients, A and B, connected in a collaborative session. Client A is being operated by the Alexandria help desk, and B is being operated by a user of the library. A is currently in the master (send) mode; B is in the slave (receive) mode. When the information specialist draws a selection box on client A's Map Browser, the log module in client A appends to the log file an entry corresponding to drawing the selection box. Since A is in the master mode, the log entry is also handed up to AlexHelp!'s Event Handler Layer. There, the log entry is forwarded to AlexHelp!'s Communication Layer, where it is sent over its network connection to client B. At client B, the log entry is received at the Communication Layer, which immediately hands it down to the Event Handler Layer. The Event Handler parses the log entry to determine if it is of interest; in this case, it finds that the log entry consists of parameters making up a list of selection boxes. The Event Handler uses ADL client B's public API to set its collection of selection boxes; client B then redraws its display in the Map Browser window to reflect the change.

When events are received at an ADL client in the slave (receive) mode, the graphical user interface gives the user some kind of visual cues to alert him or her that something has changed: when the state of the Selection Box/Map Zoom mode buttons are changed, for example, the button that is now selected flashes red and white several times.

Given a priori knowledge of the possible forms log entries can take, the task of parsing a log entry and deciding what to do in terms of reproducing the event is rather simple. Consequently, the Event Handler Layer, possibly the most complex module in AlexHelp!, is essentially a parser for a rather restricted language (see example log entries in Section 4.1).

4.3 VCR-Style Control of Log Replay (Demo Sessions)

As in AlexHelp!'s collaborative sessions, the ADL client's session logs were important in the design and development of AlexHelp!'s Demo Sessions.

Giving a user the ability to play and replay a series of graphical user interface events as if the user were operating a VCR is a powerful learning tool that goes beyond help systems that are text-based or rely on short animations. Using AlexHelp!'s model for Demo Sessions in conjunction with detailed explanations (easily supplied along with the log files as text or audio files), help systems can be developed that show usage of complex software systems in a way that users would otherwise only experience by using the system, perhaps in a trial-and-error fashion.

Demo Sessions in AlexHelp! use log files in the same way that client B in Section 4.2 uses incoming log entries. An ADL client playing a Demo Session uses AlexHelp!'s Event Handler Layer to read a log file, parsing each log entry and carrying out the event described in the entry. AlexHelp! uses an additional window in the graphical user interface to give the user control over replay. The user may set the rate of playback, start and pause playback, single-step the session (i.e., play a single event and pause), rewind a single event, or rewind to the start of the session.

4.4 The Use and Utility of Log Files

We envision (for ADL as well as other complex software systems) help systems that utilize log files in concert with a "Help Wizard"; the Help Wizard would present the user with a series of questions that help to focus the user towards one or several log files that show examples of sessions that

(hopefully) closely resemble what the user will need to do to achieve his or her goals with the application.

This is in stark contrast to many help systems in current applications, which seem to consist mainly of duplicating the functional descriptions of each user interface component in the help files. Help systems built with session logs can be viewed as an extension of support staff; even if the support staff and the end users operate in different time zones, making synchronized activities difficult to schedule, custom session logs can be created and forwarded to the user in lieu of an online meeting. For example, one of our recommendations for application of AlexHelp! in ADL is to maintain a page on the Alexandria World Wide Web site dedicated to listing demo sessions, from which users may download demo sessions showing queries similar to those the users hope to carry out and thus learn how to use the ADL interface from the example sessions.

Users can, in turn, record their own session log for support staff analysis; what better summary of a problem using an application than a complete reproduction of the user's activity?

5 IMPLEMENTATION OF A RAPID PROTOTYPE

In this section we briefly discuss issues related to the implementation of our current prototype.

5.1 Module Architecture

The relationship of the AlexHelp! modules with the original ADL system is shown in Figure 3. The implementation details of the extension modules will be discussed in the following sections.

Of interest in Figure 3 are the various working modes of the extended ADL client:

1. *Stand-alone Mode*

   This is the same as the sole working mode supported by the original ADL system. Native user interface (UI) events generated by the local windowing environment are sent to the User Event Handler and then get recorded locally by the event logging module.

2. *Master Mode*

   When the client is in the master mode, the native UI events are still processed by the User Event Handler. However, now they are also sent to the remote client via the network interface.

3. *Slave Mode*

   When the client is in slave mode, the user interface should not respond to local UI events but remote events instead. Remote events go through the Event Parser and then the UI highlighting module, which presents visual cues to the user that remote events are occurring.

4. *VCR Mode*

   When replaying session logs (retrieved from the network or locally), the Event Parser passes each event to the UI highlighting module, which passes the events to the User Event Handler after performing the necessary highlighting.

5.2 Network Interface

The network sublayer is encapsulated in an object which is responsible for maintaining the TCP link and some state information related to the connection. The reason for choosing TCP (as opposed to UDP) is that a reliable link that guarantees delivery of synchronization messages is desired. A reliable link greatly simplifies the mode switching protocol. In the current prototype, we have not added the mode switching control to the help desk's ADL client (since we decided to keep only one code base). When implemented, the information specialist will control mode switches. The reliable TCP connection allows us to use a simple handshaking protocol to ensure state consistency.

Event Parser

The kernel of the session player is an Event Parser which breaks down session logs into a sequence of event records and retrieves the type and parameter of each event. There are a finite number of event types. As shown in Section 4.1, for each type, the format of the events is a simple regular language. This makes the manual coding of the Event Parser that we decided to use much simpler than using a lexical analyzer generator such as `lex`.

5.3 Session Player

Our multithreaded Session Player performs well; however, careful attention must be paid to the "atomicity" of replaying remote events. Aborting an event in the middle will result in an inconsistent state. All events we are interested in (i.e., those that we capture and relay to the ADL client that's in the slave Mode) finish in a finite and brief amount of time, so we simply use a protocol that ensures the completion of the current event.

5.4 User Interface (Receiver Event Notification)

Not all user interface events are obvious if not generated by the user. We found that in replaying events received from the remote Master client, major changes were obvious (such things as zooming in or out in the Map Browser window), but minor changes resulting from selecting a button were easily missed by the user. Ideally, every user interface event could be duplicated, down to tracking the movement of the mouse. This way it is easier to follow subtle changes in the user interface state. It's possible to simulate user interface events at this level of granularity, but it requires a formidable amount of user interface resources. In addition, the time required for implementing such a scheme was not available to us in developing a prototype. We instead added a fairly simple layer between the User Event Handler and the Event Parser that, depending on the user interface component in question, gives the user obvious visual clues. For example, when changing mode from Select to Erase in the Map Browser window, the Select button changes its highlighting to the default color for non-selected components, and the Erase button flashes red and white several times, finally settling on the default color for selected components. This light-weight and modular approach makes it easy to replace or extend the event notification system.

5.5 Synchronized Web Browser

As mentioned in Section 2, the current ADL client also uses a World Wide Web browser to display images (the results of

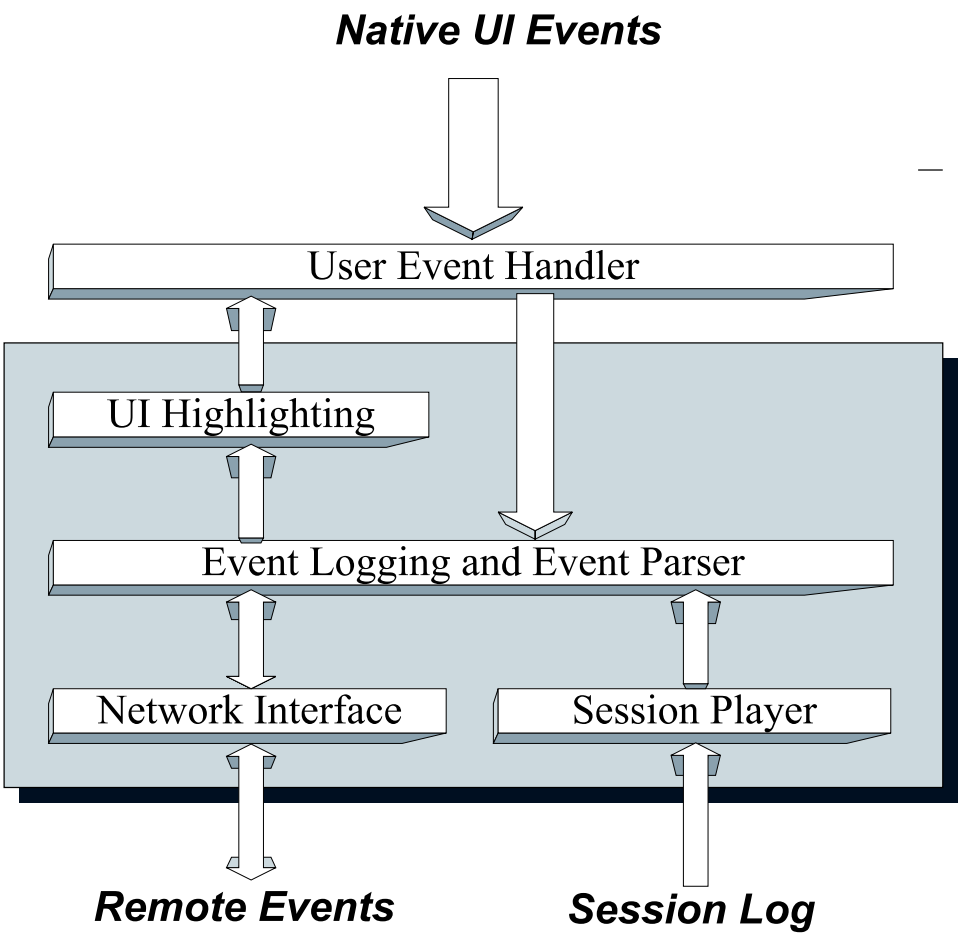

Figure 3: The relationship between the extension modules and the original system

queries). In a separate project, we implemented some primitive synchronization between two World Wide Web browsers (Netscape) using JavaScript and Java; the functionality is similar to but much simpler than [WR94, FLF94].

6 CONCLUSIONS AND FUTURE WORK

Our experience in the design and development of AlexHelp! shows that adding collaborative functionality to an existing system as an independent module can be a viable and fast approach.

Our approach can be summarized in the following steps:

1. Examine the existing system and based on the analysis, carefully design the extension system. The design work includes but is not limited to the extension architecture and function specification.
2. Identify one or several likely hook points between the original system and the extension. In our case, one of the hook points was the session logging; we augmented the logging facilities to enable distribution of user interface events.
3. Perform detailed design work, including communication interface, state transition protocols and user events reproduction, etc.
4. Implement and link the new module to the existing system.

As mentioned previously, what we have done is a throwaway prototype; some promising features have been left out due to time constraints, which include:

- *Help Wizard*

  The design of a good help wizard requires much analysis of the problems that users may encounter. It might be interesting to add some logic to the help wizard that allowed it to learn from users' requests.

- *Call Forwarding*

  Call forwarding is not a trivial feature. Since TCP/IP sockets don't allow the migration of connection information from system to system, transparent call forwarding (meaning the user should not be aware that the connection has been broken and re-established) involves another layer upon the TCP/IP protocol.

- *Wait Time Estimation*

  An accurate estimation of the expected waiting time is necessary for a practical system. However, finding a suitable model for such estimation could itself be an interesting research area. One such model would involve using priority-based scheduling for the incoming help call

requests. Many factors must be considered when determining priority: user class ("premium user", "common user", etc.) accumulated wait time, the difficulty or urgency of the problem (if reasonable measures can be established), etc. Scheduling in this fashion is similar to process scheduling in an operating system.

- *Multicast Support*

  Multicast support will become more important as the user group grows larger. The incorporation of multicast capability involves modification of the collaboration model and mode control protocol.

- *Client Side Action Analysis*

  Users' session logs reflect the actions they've performed. This data is valuable for the analysis of how the digital library is being used. Data mining may be added to ease the task of analyzing such a potentially large volume of data.

- *Security*

  Currently ADL is only accessed by a small testing community; however, security issues will become important when the service is open to public.

- *Multimedia Support*

  Multimedia support is not difficult to plug in as several separate channels within the Communication Layer. However, since users can access the library via different links, ranging from high speed network connections to low speed modem connections, users should be able to control the bandwidth by switching on only the multimedia features their connection can accommodate.

ACKNOWLEDGMENTS

The authors thank Vinod Anupam for his stimulating discussions which lead to the idea of online help studied in this paper; Linda Hill, Mary Larsgaard, and the ADL implementation team, in particular, Nathan Freitas and Kevin Lovette, for their comments and help in the specification and implementation of AlexHelp!; Linda Hill and Mary-Anna Rae for their comments on an earlier version of this paper.

REFERENCES

[BM94] N. W. Bergmann and J. C. Mudge. Automated assistance for the telemeeting lifecycle. In *Proc. ACM Conference on Computer Supported Cooperative Work*, 1994.

[CGJ+98] A. Chabert, E. Grossman, L. Jackson, S. Pietrowicz, and C. Seguin. Java object-sharing in Habanero. *Communications of the ACM*, 41(6):69–76, June 1998.

[FFL+95] C. Fischer, J. Frew, M. Larsgaard, T.R. Smith, and Q.Zheng. Alexandria digital library: Rapid prototype and metadata schema. In *Proc. Int. Conf. on the Advances in Digital Libraries*, 1995.

[FLF94] T. J. Frivold, R. E. Lang, and M. W. Fong. Extending www for synchronous collaboration. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994. (http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/frivold/frivold.html).

[GM94] S. Greenberg and D. Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proc. ACM Conference on Computer Supported Cooperative Work*, pages 207–218, 1994.

[Gru94a] J. Grudin. Computer-supported cooperative work: Its history and participation. *IEEE Computer*, 27(5): 19–26, 1994.

[Gru94b] J. Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):92–105, 1994.

[JAVA] *JDK 1.1.6 Documentation*, 1997. (http://www.javasoft.com/products/jdk/1.1/docs/index.html).

[JSDT] *Java Shared Data Toolkit*, 1998. (http://developer.javasoft.com/developer/earlyAccess/jsdt/index.html).

[MR91] J. A. Mariani and T. Rodden. The impact of CSCW on database technology. In *Proc. ACM Conference on Computer Supported Cooperative Work*, 1991.

[WR94] T. K. Woo and M. J. Rees. A synchronous collaboration tool for World-Wide Web. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994. (http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/rees/SynColTol.html).