Using Transcription and Replay in Analysis of Groupware Applications

Seth Landsman
Department of Computer Science
Brandeis University
415 South Street
Waltham, MA 02454
+1-781-736-2712
seth@cs.brandeis.edu

Richard Alterman
Department of Computer Science
Brandeis University
415 South Street
Waltham, MA 02454
+1-781-736-2703
alterman@cs.brandeis.edu

ABSTRACT
As corporations and organizations become more distributed enterprises, the use of groupware applications as part of day-to-day activities becomes more prevalent. How to build groupware applications so that they work as expected, both as the developer expects and the community of users expect, is a challenge that must be addressed in order for groupware applications to be adapted as part of daily business activities. In practice, assumptions made by the developer may not match the expectations of the users.

A set of techniques are discussed in this paper that enable the ethnographic analysis of an online groupware application already in use. This analysis depends on two specific capabilities, the ability to collect a *transcript* of the application use, and then *replay* the transcript for the developer, allowing its use to be analyzed and understood.

This paper details our approach to collecting online data that can subsequently be replayed as part of ethnographic analysis. We present our groupware application framework that enables the production of complete transcripts of groupware use, for which customizable replay application can be created. An example analysis is presented that shows how the replayable transcripts that are produced can be used as a basis for refinement of a groupware application.

Author Keywords
Groupware, Analysis, Transcription, Replay

ACM Classification Keywords
H.5.3 Information interfaces and presentation (e.g., HCI): Group and Organization Interfaces

INTRODUCTION

Companies and organizations are becoming increasingly distributed enterprises, with offices, employees, clients, and consultants spread throughout the world. People in different locations need to collaborate to achieve company and organizational goals. In-person meetings are desirable and most effective, but the cost of travel can be an unaffordable luxury.

In his book, *Designing the User Interface* [12], Shneiderman says:

> Goal-directed people quickly recognized the benefits of electronic cooperation and potential to live in the immediacy of the networked global village. The distance to colleagues is measured not in miles, but rather in intellectual compatibility and responsiveness; a close friend is someone who responds from 3000 miles away within three minutes at three A.M. with the reference you need to finish a paper.

Corporate employees do not work in an eight-hour day configuration. Players in the global economy exist in all time zones, and they must make business decisions correctly and quickly, no matter the hour. Collaborative applications can allow organization-wide work to continue after traditional working hours.

Beyond the need to be competitive, collaboration also improves overall productivity. An agile corporation succeeds through the intelligent use of its personnel by leveraging each worker's abilities and knowledge. Coordination between employees is necessary in order to share and support the tasks required by the organization. These software applications that mediate the distributed work of a community of users as they work towards a common goal are collectively called *groupware* [3].

No software applications, and especially groupware, are fixed entities. They evolve and change constantly, both during development and after they have been fielded. Software needs to change in order to accommodate different work flows, different groups of users, and changing business environments, all of which invalidates the view of software as fixed, static objects. Figure 1 shows a Congressional General Account-

ing Office study [10], which illustrates that only 2% of acquired software products are deployed in the state in which they are delivered, and that this percentage is not improving, despite sixteen years of progress in the field.

Suchman and Trigg [14] developed *ethnographic analysis* as a method of studying collaborative activity in the workplace. Their approach involved the video taping of several stations where a physical collaborative artifact was modified or otherwise used, and then passed to another station. A similar approach is needed for the study of software systems, especially groupware applications where the activity is mediated by the application. Our understanding of how to build better collaborative tools is improved through understanding how collaboration occurs in the field.

Ethnographic techniques are effective for studying the use of physical, off-line artifacts of collaboration. To extend the applicability of these techniques to on-line collaboration it is necessary to collect reviewable transcripts of users' behavior, as mediated via the software system.

This paper describes the techniques that we have implemented for allowing ethnographic analysis to be realized on an online groupware application and how we integrate the use of these techniques into the lifecycle of building, understanding, and modifying groupware applications. The next section discusses the state-of-the-art in transcription and replay technology. Following that our approach to enabling the collection of reviewable transcripts of use is presented. An example ethnographic analysis clarifies the kinds of analysis that are enabled by this technology. The paper concludes with a discussion of future work.

TRANSCRIPTION AND REPLAY

Transcription of the use of the online groupware application and the replay of this transcript enable ethnographic analysis. Transcription is the mechanism that records the interaction that the user has vis-a-vis the groupware application. As discussed in this section, there are a number of vectors on which the transcription capabilities of the application can be described. How the transcript is collected influences the available replay options, including the fidelity and precision available to the analyst.

Similarly, there are several different properties associated with the replay application. Some capabilities depend on the corresponding properties of the transcript, while others are inherent in the replay application itself. We will discuss these properties and compare requirements for online ethnographic analysis with the properties that have been achieved by other efforts.

Transcription

A transcript is the record of the user's activity, as mediated by the collaborative system. The way a transcript is collected can have direct influence on the quality and quantity of replay techniques available to analyze the system. Our set of identified properties include:

Collection of Online and Off-line Activity Transcripts may encode online behavior, which is the activity that is directly mediated by the software system. They may also encode off-line behavior, which may be part of the collaboration, but not directly mediated by the software. Eye tracking, for example, is usually a collection of off-line behavior, where mouse actions are online behaviors. While off-line behavior can add value to observation, the quality of the online behavior collection is most important for our analysis techniques.

Type of Online Information Encoded There are two types of online information that a transcript can encode:

User Interface events (UI) UI events include direct manipulation events in the user interface, such as mouse clicks and key presses.

Task Events Task event information refers to interaction that is mediated by the groupware application. Where user interface events depict the users activities at the level of point-and-click, task events depicts the users activities at the level of plans and communication. The level of event structure is important because it enables the analyst to review and replay the transcript in terms of the semantics of the domain.

Online Completeness A complete transcript contains information sufficient to re-create the state of the application at any given point in time. A transcript can be complete with respect to user interface events or task events. A transcript that is complete with respect to user interface events is not necessarily complete with regards to task events, and visa versa. For example, a transcript can encode the sequence of keys that were tapped, but in itself that information is not sufficient to reliably reconstruct whether the user's task was planning or chatting.

jRapture [13], Playback [9], and others [11] provide complete transcripts of user interface events only. jRapture replaces the underlying Java libraries with ones that transcribe external interactions with the application. The Playback application captures interface events by "intercepting" interaction at the device interface level.

Timewarp [2] and Chimera [6] provide a complete transcript only with respect to an enumerated set of task events. They both collect a history of actions within the application by collecting the interaction with the groupware application into a transcript. Chimera uses the transcript as a basis for end-user programming of macros. Timewarp constructs a history of changes to a data object, e.g., a document. The user can modify a historical data object, and thereby change all of its descendants. Neither Chimera or Timewarp provide replayable transcripts that could be used for ethnographic analysis.

Transitions or States As the transcript is generated during a session of use, it can either record *transitions between states* or *individual states*. The advantage of a transcript that encodes data in terms of state is that it allows the playback tool to directly access any state; the disadvantage is that collecting such a transcript is spatially and computationally expensive. Storing transitions result in a smaller

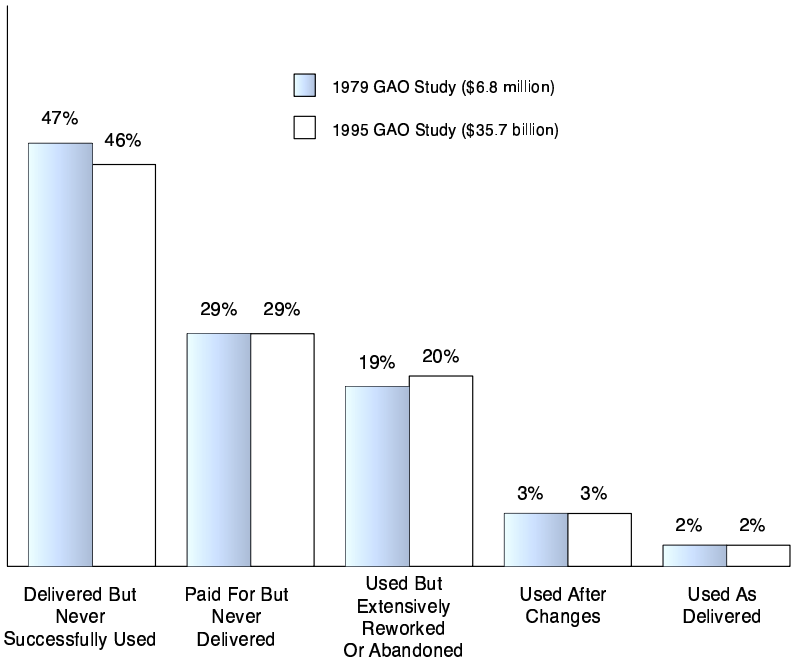

Figure 1. A Congressional GAO Study

transcript and will be computationally cheaper to collect, but at the cost of more expensive post processing and playback of the transcript. Rewind may be costly as it may entail re-running playback from the start of the transcript until it reaches the prior state the analyst chooses to examine.

Table 1 summarizes our discussion of prior efforts at transcription in the terms of the criteria we have developed.

Application	Complete	Info Type	Transitions	Off-line
jRapture	UI	UI	transitions	none
Playback	UI	UI	transitions	none
TimeWarp	task	task	transitions	none
Videotape	none	N/A	states	video

Table 1. Transcription Features

Replay

Ethnographic analysis of online collaboration requires the ability to replay the transcript of system use. Depending on how the transcript is collected, different capabilities become available to the replay system. Which capabilities the replay system implements affects how the analyst can interact and use the transcript, and include:

Search What kinds of events the analyst can use the playback tool to search for depends on what kinds of events are in the transcript. For example, a transcript that only encodes mouse clicks and key presses will not provide the basis for the analyst to use the playback tool to search for chat events among users.

Annotation Annotations give the analyst the ability to annotate, tag, or otherwise mark the transcript as the application session is replayed. In addition to providing information for an analyst to refer to in later analysis sessions, they also provide additional information for the playback tool to search for and notes for other analysts use.

The video tape solution provided by Suchman and Trigg [14] provides the means for annotating a video tape transcript during playback, allowing areas of interest to be clearly marked for future reference.

Precision After the transcript is generated, the analyst can always annotate the transcript noting events of particular interest that can later be returned to for further analysis. Annotation is a time consuming and potentially inaccurate task for the analyst. Ideally, the analyst can replay an unannotated transcript stopping at, for example, each chat event. *Precision* is used to indicate that a transcript is sufficiently encoded with information such that the replay application can accurately differentiate between different features of events.

A playback tool is precise with regards to time if the analyst can replay the transcript (without annotation) to directly display an event that occurred at a specific timestamp. A playback tool is precise with regards to task event if the analyst can replay the transcript to display a task event of a certain type.

The playback provided by jRapture and Playback, for example, allow for precision based on the type of user interface event and the timestamp. However, they do not provide precision based on the task event, since those do not exist within the transcript.

Aggregate information Some playback tools allow the ana-

lyst to summarize and display quantitative data of system use. For example, this data can include a count of window events or a count of collaboration failures.

Applications such as CollabLogger [8] are specifically designed to allow for the collection of aggregate information. These applications collect transcripts and analyze them to gather statistics as to how the application was used, how particular participants performed as a measure of their interaction with the application, and other similar measures.

Table 2 summarizes our discussion of prior efforts at playback in the terms of the criteria we have developed. In particular, ethnographic analysis is best served by search, precision, and annotation capabilities.

ENGINEERING ONLINE ETHNOGRAPHIC ANALYSIS

To engineer the online ethnographic analysis capabilities into an application and its development lifecycle, the appropriate transcription and replay technology must be put into practice. Our implementation provides the feature set necessary to do the level of replay necessary for ethnographic analysis. This section describes how we engineered this technology into our groupware application framework.

Transcription and Replay Requirements for Ethnographic Analysis

Many of the features of a replay tool depend on the quality of the transcript. Analysis is best supported by a complete transcript, where each state of the collaboration can be reconstituted. It is important that this transcript be complete for events that describe interaction with the user interface, such as mouse clicks, and events that describe interaction with the task environment, such as chat utterances, as both types of information can be critical to understanding the collaboration process.

The technology we have developed produces a transcript that is complete and encodes both task and user interface events. The transcript itself is encoded as additive transitions. Our framework is based on a message-passing architecture, and collects information from both user interface and task environment actions by collecting messages as they are generated within the application during run-time.

The replay technology processes the transcript so that the analyst can view the collaborative activity from a perspective similar to that of the users who generated the transcript. Our replay tool allows the analyst to search an unannotated transcript for the next task event of a certain type. The analyst can also step through the transcript one event at a time. Each of these features depend on the completeness and level of information encoded in the transcript.

Because the transcripts encode task events, the transcripts can be used to do a quantitative analysis of, for example, how much chatting the users did. Similar, they can also be used to perform various kinds of quantitative analyses on user interface interaction with the application.

Implementation

Our transcription and replay techniques were realized in two complementary software libraries. The first library, THYME, is a framework for building message-oriented groupware applications. A groupware application constructed using the THYME framework automatically generate transcripts of their use during the application's runtime. These transcript, as described above, are complete, encode task and user interface events, and are transitional. They encode only online user information.

The second framework, SAGE, provides the foundation for assembling replay applications. The THYME application that was used to collect a replayable transcript is the basis for constructing the replay application. The replay application that is assembled provides the necessary "over-the-shoulder" perspective as to what the user was doing when the transcript was collected.

Some implementations of replay, such as jRapture, use, without modification, the original application to do replay, but at a cost. SAGE allows the replay application to better support the analysts work. For example, a SAGE-produced replay application can include useful features like rewind, filtering, and alternative views of shared representations. Thus, while the most basic SAGE application will look identical to the groupware application on which it is based, the underlying capabilities support the replay of the collected transcript.

Transcription

A THYME application is defined by a set of components and the messaging connections between them. Components are grouped into structures called *nodes*, each of which has its own namespace. These components communicate via messages, orchestrated by a per-node object called the *message router*. A component sends an addressed message to the message router. The message router will find the component the message is addressed to and deliver it. More details of how THYME works and the types of groupware applications built using THYME can be found in another work [7].

THYME's message-oriented architecture has several useful consequences for collecting a transcript of use:

1. All communication between components happens through messages, so a THYME application only needs to collect messages in order to generate a complete transcript.
2. Since all messages go through routing components, only the message routers need to be accessed in order to record all the messages.

To collect the transcript for an application's session of use, the message routers collect all messages at their point of origin, defined as the first message router that handles the message. This approach ensures that every message is logged once and only once. As a router collects messages, it sends the message to the *transcript collector* component. This component will store all messages it receives, thereby building the transcript. This component forms a unified interface to the transcription subsystem, abstracting the means

Application	Search	Precision	Annotation	Aggregate Information
jRapture	No	time, ui	none	none
Playback	No	time, ui	none	none
CollabLogger	No	none	none	yes
Videotape	No	time	yes	none

Table 2. Playback Features

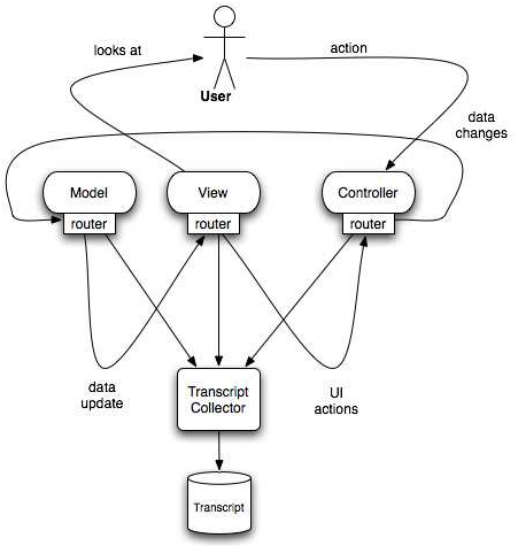

Figure 2. Transcripting messages in a THYME application

by which the transcript is stored and reconstructed and allowing different transcription formats and strategies to be used without changing the application. Figure 2 illustrates the interaction of a THYME application with the transcription subsystem.

Each transcribed event in the THYME framework is stored with two timestamps, when it is first handled by a router, and again when it is actually transcribed. The two timestamps gives sufficient data for clock skew correction to be performed, if necessary. The timestamp is the discrete points in the timeline of the session, and is used to explicitly order messages as they get injected into the playback application.

Within the THYME application, transcripts can be accessed during the run-time of the application through a service component called the transcript emitter. This service provides access to previous transcribed messages in the current session of use. Transcript collection is ended when the session of use is over, and archived so that it includes session summary information (called *meta* information) in addition to the transcript of events. Archived sessions of use can be loaded into the transcript emitter explicitly. A SAGE application makes use of the transcript emitter to playback the messages contained in the archived transcript, in the order those messages were archived.

In a THYME application, all changes to the application state occur through the reception and processing of messages. A *complete* transcript is, therefore, the collection of all messages that are sent between components. A transcript is represented as $TRANSCRIPT_{T=\{X,Y\}}$ and represents the transcript for all messages between M_X and M_Y inclusive, where M_I is the $I-th$ message sent in that session of use. $TRANSCRIPT(X)$ is used for referring to the message M_X that is stored in the transcript.

The transcript collects messages that are a consequence both interface events and task actions. Interface events can be encapsulated as messages to an *interface controller* and thereby are recorded into the transcript. The task events are recorded through the collection of messages that are the consequence of task environment activity.

The collection of task activity provides the ability to work with the replay application in a way that encodes the meaningful activity in the application. For example, a message passed from one client's chat component to another client's chat component contains information that the message is a chat message. Thus, during replay, the analyst can run the replay until it comes to a *chat message*. Alternately, if users are collaboratively constructing a plan in a shared window, messages between client planning components will contain information that enable the precise replay of planning actions.

Replay

Replay of a transcript is enabled by the SAGE framework. The framework provides access to the collected transcript and enables a replay application to rebuild the state of the basis application at a precision level of individual messages. Additionally, these replay applications can be built relatively cheaply by leveraging the basis application, yet still enjoy many of the expected advantages of a customized replay application, such as reviewing past states and customized views as warranted.

The state of the application is built from the application of messages. During a THYME application's run-time, a message, M, is applied to a component C, resulting in a component C'. Succinctly, this process is represented as $C(M) = C'$. An application consists of a set of connected components, $\{C_0, C_1, \ldots, C_n\}$. Therefore, applying a message M to an application A is represented $A(M) = A'$, which is also written as $A(M) = \{C_0(M), C_1(M), \ldots, C_n(M)\}$.

Given a basis application A and a collected transcript T of size S replaying T on A is done by applying the elements of

the transcript on each component in the application. This is done as follows:

$REPLAY(A, T, 0, S) =$
$(\forall_i : 0 \leq i \leq S; A_{t=i} = A_{t=i-1} + T(i))$

Where the statement $A_{t=i} = A_{t=i-1} + T(i)$ is expanded to

$A_{t=i-1} + T(i) = (\forall_C : C \in A_{t=i-1}; C = C(T(i)))$

Where $C_{t=x}$ is the component C after $T(x)$ has been applied. Replay to a specific message X is done by applying all positionally previous events from the transcript, up to and including M_X, to the application.

Technically, SAGE only provides event-level *precision*. Time-level precision within a collected transcript is limited to the instants of time that are collected in the transcript. While each message has an associated timestamp that refers to when it was collected, the granularity is limited to the points in time where the messages were actually collected. For example, if the time between a message M_{i-1} and M_i is 10 minutes, there is no way to display any activity between those two events. However, if the transcript is complete, it is possible to go to a specific point in time by progressing to the last message that occurred before the requested timestamp. If, in the example given, the transcript is complete, it can be deduced that no system activity occurred in the intervening 10 minutes, so there is no real precision lost. From these conditions, it can be shown that SAGE emulates time-level precision.

Supporting user interface or task-level precision occurs through being able to search for specific types of events. Given a transcript, the set of possible event types is represented by the set $EVENT\text{-}TYPES(T)$, which are mined from the transcript T. The available set of $EVENT\text{-}TYPES$ is related to the level of task information in the transcript. If the transcript only contains UI events, then that level of granularity is available to the analyst through the replay tool. However, if the transcript contains events that illustrate interaction with the task environment, such as chat messages, then the replay tool can use those events to show event boundaries. This information would allow the analyst to say "skip to the next chat utterance", for example.

Milestoning

The approach of encoding transitions between states is done for two major reasons. Encoding the entire state at each change will take up a large amount of disk and processing resources and will require a level of introspection and access to all aspects of the application that may not be easily available. Instead, encoding transitions is accomplished through the use of the existing message passing infrastructure, without needing information about the components, their state, and how their state can be captured.

In encoding transitions, the ideal situation would have each transition reversible. That is, $C_{T=i} = C_{T=i-1} + M_i$ and $C_{T=i-1} = C_{T=i} - M_i$. However, messages in our framework, as is true in the majority of message-passing frameworks, are not designed to be reversible, as doing so puts a large burden on the developer to track state and limit actions on the application data. Without reversible messages, actions such rewinding an application's state is difficult. A brute force example of rewinding state could consist of selecting the desired point in the transcript, resetting the application and applying all messages up to the newly desired timestamp. Early versions of SAGE provided such a mechanism, which was quickly deemed unsatisfactory.

Since the data that makes up a THYME application is stored in components, to implement a proper rewinding mechanism for a transcript requires each component to be capable of rewinding its state. To accomplish rewind across all types of components, SAGE provides a set of component wrappers, based on a design pattern called *mementos* [5]. A wrapper's internal state is actually a collection of milestones, which are indexed instances of the component it is wrapping. Each index refers to a specific message number in the transcript. Milestones are laid out so that to retrieve an instance that corresponds to a timestamp previous to the current one, it is only necessary to find the component that is closest to, but previous to, the desired timestamp. That component is then copied and any messages that exist between the desired timestamp and the current component's timestamp are retrieved and applied. This process is shown in Definition 1.

DEFINITION 1. *Given a component wrapper CW that wraps a component type C, upon receiving message M, which is position I in transcript T, the following takes place:*

1. *if there is a milestone MI in CW that corresponds to $C_{T=I}$, activate that milestone and exit*
2. *if there is no such milestone, find the milestone MI_J that has the closest index less than I*
3. *copy MI_J to a new component $C_{T=I}$*
4. *apply every message M_X where $X > J \geq I$ to $C_{T=I}$*
5. *activate $C_{T=I}$*
6. *if a new milestone is desired at I, store $C_{T=}I$ into a milestone MI_I*

The decision to store milestones depends on the application and storage needs. Generally, milestones exist increasing intervals.

A component can implement its own state mechanisms so that it can provide its own reverse functionality. The wrapper approach is a general solution if the component does not have this capability already.

Analysis Interface

Using the replay application, an analyst can perform precise analysis of the usage of the basis application. The SAGE Playback Controller (shown in Figure 3) gives the analyst control over the flow of the playback of the transcript. The playback controller has standard VCR-like controls: play,

rewind, fast forward, and stop (see 1 in the figure). The controller adds two other standard movement controls: step forward and step back, which move one event forwards and backwards, respectively. The controller also allows movement through the transcript by searching for types of messages that are in the transcript's set of $EVENT\text{-}TYPES$ (see 3). The list of types is populated from the transcript at the run-time of the replay application. (Note that the message displayed in this example is the Shared Browser message, more meaningful event names will be available in a future version of the replay application.)

The controller also provides the analyst feedback as to where he is in the session. The information underneath the VCR controls (see 2) shows the current timestamp of the session, based on the timestamp of the last event replayed. This number may be the standard Unix milliseconds-since-epoch, or more a traditional format, showing time and date. Next to the time display is the current message and the total number of events in the session. Movement within the session can also be controlled via a slider (see 4), at the bottom of the window. The slider provides feedback as to where in the session the current timestamp is, with the far left of the slider being the beginning of the session and the far right being the end. The analyst can manipulate the slider, causing the playback tool to go to the event closest to the timestamp selected.

The controller exposes six types of playback actions. They are:

Step Forward Move precisely one event forward in the transcript

Step Backwards Move precisely one event backwards in the transcript

Play Step forward in the transcript until the end of the transcript or the analyst stops the playback

Rewind Step backwards in the transcript until the end of the transcript or the analyst stops the playback

Play Until Plays the transcript until a condition is met, such as an event type or timestamp being reached

Rewind Until Rewinds the transcript until a condition is met

ANALYSIS OF GROUPWARE

In the Fall of 2002, a Human-Computer Interaction (HCI) class held at Brandeis University used the THYME framework to implement their term projects, which called for them to implement synchronous groupware applications. One such group in this class produced an application called the **O**nline **R**esearch **A**ssistant. This application is an application that allows a more experienced researcher (such as a librarian) to help another researcher locate information on the World Wide Web. This section details an example analysis performed on a transcript collected using this application.

The ORA application contains several common groupware components, including a shared whiteboard, a chat room, and a shared web browser; see Figure 4. The left side of the screen (see 1) is the web browser, which has the shared whiteboard is a Java glasspane on top of it. The browser is a relaxed WYSIWIS component, in that the web browser context is synchronized, but not the scrolling of the web page. The right side of the screen (see 2) is the chat room. The bottom of the screen (see 3) contains the tools used to manipulate the shared whiteboard, including the palette of tools and button to toggle the whiteboard's display. ORA was implemented using the THYME framework and heavily leverages existing components and capabilities. All the groupware components feed UI and task events into the transcript of use.

From the ORA transcript, there are three types of task events: the Chat Event, Shared Whiteboard Events (i.e., drawing a new artifact on the glass pane or manipulating an existing one), and Shared Web Browser Events (including entering a new URL, and going forward and backwards in the history of visited URLs).

From a comparison of the ORA replay application (Figure 5) and the basis ORA application (Figure 4), it is clear that they share a common lineage. As shown in Figure 5, the center replay window is a direct analogue of the ORA client screen, and contains individual components that are leveraged from the basis application. These components share the user interface directly from the basis application and use the component wrappers, as discussed in Definition 1, to enable the replay of their state.

Example Analysis

We performed data collection and analyses on usage of the initial design of the ORA application. The analysis is used to direct the further development of the application, allowing conclusions to be drawn about how the application is to be used by the end-user. Based on these conclusions, the development directions can be determined with a large degree of precision and accuracy, responding to the elicited and observed needs of the community.

In the example analysis shown, two users with the pseudonyms *Tom* and *Jerry*, are using the baseline version of ORA. Jerry is an ORA developer and researcher who is assisting Tom in finding papers relevant to a specific topic.

Throughout this discussion, we will emphasize how the analyst manipulated the playback application by *italicizing* his actions. The actions available are described above. Figure 5 shows a segment from this analysis session.

This analysis has three major stages:

1. Initial exploration of a university library, that fails because of limitations in the web browser associated with the ORA application.

2. Transition to another citation database (the *Citeseer* database) and resynchronization of the user's common ground

3. Successful competition of the task

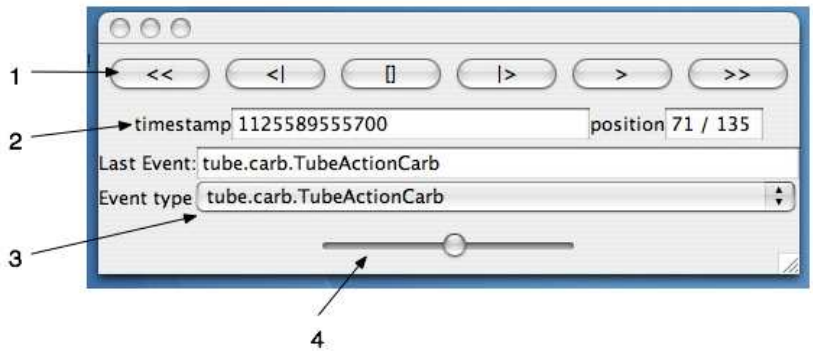

Figure 3. SAGE Playback Controller

The first part of the session starts with the interaction between Tom and Jerry (taken verbatim from the transcript):

Tom Hi. Can you help me to find articles or books on mutual belief? I am particularly interested in representation and mutual belief. But first the general concept of mutual belief.

Jerry sure. let's start in the library.

Tom wait what did you just do??

To obtain this dialogue, the analyst starts *playing* the transcript. By observing the replay, the analyst can see that this exchange corresponds with Jerry going to the library website. The page loading gives no feedback to the user who did not initiate going to the new website. This confusion as to what Jerry is doing results in a question from Tom, which requires a repair.

The analyst continues to observe by *playing until the next chat event*. It becomes clear that the use of the library does not yield any results. Jerry moves on to the CiteSeer website. Again, because of the lack of feedback to Tom as to what the web browser is doing, the following interchange occurs:

Tom are you there? I can't see what you are doing.

Jerry we will search through citeseer for the articles, since the library doesn't have anything good.

Tom where are you? nothing is happening

Jerry we're searching through citeseer for articles that contain the words mutual and belief

Once Tom's second set of questions have been asked, the analyst performs a *rewind until the previous web browser event*. Now the state of the replay is at the web browser event prior to Tom's question and the analyst can ascertain why Tom would not see any feedback from Jerry's actions. He continues to *step forward*, observing that Jerry attempted to use the library website to search for books, but the web browser component failed to interact with the custom JavaScript on this website. He then *plays until the next web browser event* after Tom's last utterance, and can see that Jerry switched to the CiteSeer website.

Through the session, a number of desired features are specifically identified. For example:

Tom can I look at the pdf?

Tom any version of the paper?

Jerry unfortunately, there is no PDF viewer built into this application. We can add it in a few spirals.

In viewing the activity in the replay tool, a workaround is identified by Tom. By continuing to *play until web browser events*, it is observed that Tom discovers that CiteSeer can display images of papers, which worked sufficiently for this task.

Another request was made, to allow searching for specific features and subject matters of papers, which was clearly not within the scope of this tool, and identified as such during the conversation.

Tom is there a way we can filter for philosophy and not ai papers?

Jerry I do not think citeseer has that capability.

Jerry I don't think any paper search engine has that capability, currently.

Jerry how can you tell that a paper is philosophy, and not AI?

Based on the observations of use, pointing to the web page and aspects thereof was done frequently, but the drawing tools, which existed to aid in the referencing of objects in the web page were not used. In fact, only 8 of 135 events were shared whiteboard events in this session, as reported by the replay application. Because of how the drawing tools were implemented, in that they required changing the application's mode of use from browsing to drawing, their use may have been too cumbersome. It may also be that in this collaboration, which had only two users and relatively manageable web pages, the pointing capabilities were not

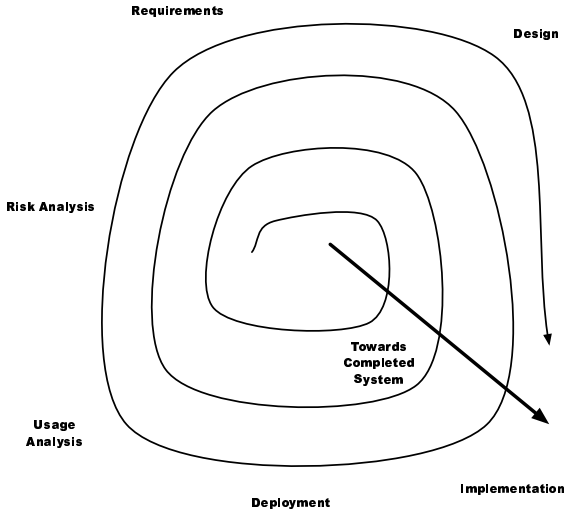

Figure 6. Spiral Lifecycle with Ethnographic Analysis

needed. As the application is used further and more data is collected, the drawing mechanism will see more use and may need to be refined.

CONCLUSIONS

This paper discussed our approach online ethnographic analysis of groupware applications. In the analysis of groupware applications, an "over-the-shoulder" perspective of the user's activity is given to an analyst, allowing him to draw conclusions by observing how the application was used. Contrasted to quantitative methods, this perspective allows insights to be drawn that may not be otherwise available, especially in how collaborative breakdowns occur. When paired with quantitative analysis of the transcript the after-execution analysis process can be greatly enhanced [4].

With the availability of complete, replayable transcripts, ethnographic analysis can be brought to bear throughout the software lifecycle of design, development, and deployment. A modified version of the *spiral* software lifecycle [1], as shown in Figure 6, includes both ethnographic and quantitative stage of analysis that feed into the risk analysis and requirements stages.

If the development of the replay applications can be reduced to be a small part of the total development costs of the application itself then analysis can be realized as part of the groupware software lifecycle. In another work [7] we discuss some techniques in place to allow SAGE to generate the majority of the replay application, based on a well-instrumented THYME application. In order for these techniques to be truly incorporated as part of groupware development, these application generation techniques need to be expanded and studied.

ACKNOWLEDGEMENTS

This work was supported under ONR grant N000-14-02-1-0131.

REFERENCES

1. Barry Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, 1988.
2. W. Keith Edwards and Elizabeth D. Mynatt. Timewarp: techniques for autonomous collaboration. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 218–225. ACM Press, 1997.
3. Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.
4. Alexander Feinman and Richard Alterman. Discourse analysis techniques for modeling group interaction. In *Ninth Internation Conference on User Modeling*, 2003.
5. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.
6. David Kurlander and Steven Feiner. A history-based macro by example system. In *Proceedings of the 5th annual ACM symposium on User interface software and technology*, pages 99–106. ACM Press, 1992.
7. Seth Landsman. *Building Groupware on THYME*. PhD thesis, Brandeis University, 2005.
8. Emile Morse and Michelle Potts Steves. Collablogger: A tool for visualizing groups at work. In *Proceedings of the IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 104–109, 2000.
9. Alan S. Neal and Roger M. Simons. Playback: A method for evaluating the usability of software and its documentation. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 78–82, 1983.
10. U. S. Government General Accounting Office. Stronger management practices are needed to improve DOD's software-intensive weapon acquisitions.
11. Michiel Ronsse, Koen De Bosschere, Mark Christiaens, Jacques Chassin de Kergommeaux, and Dieter Kranzlmüller. Record/replay for nondeterministic program executions. *Communications of the ACM*, 46(9):62–67, 2003.
12. Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley, 1998.
13. John Steven, Pravir Chandra, Bob Fleck, and Andy Podgurski. jRapture: A capture/replay tool for observation-based testing. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 158–167. ACM Press, 2000.
14. L. Suchman and R. Trigg. Understanding practice: Video as a medium for reflection and design. In J. Greenbaum and M. Kyng, editors, *Design at Work: Cooperative Design of Computer Systems*, pages 65–89. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1991.

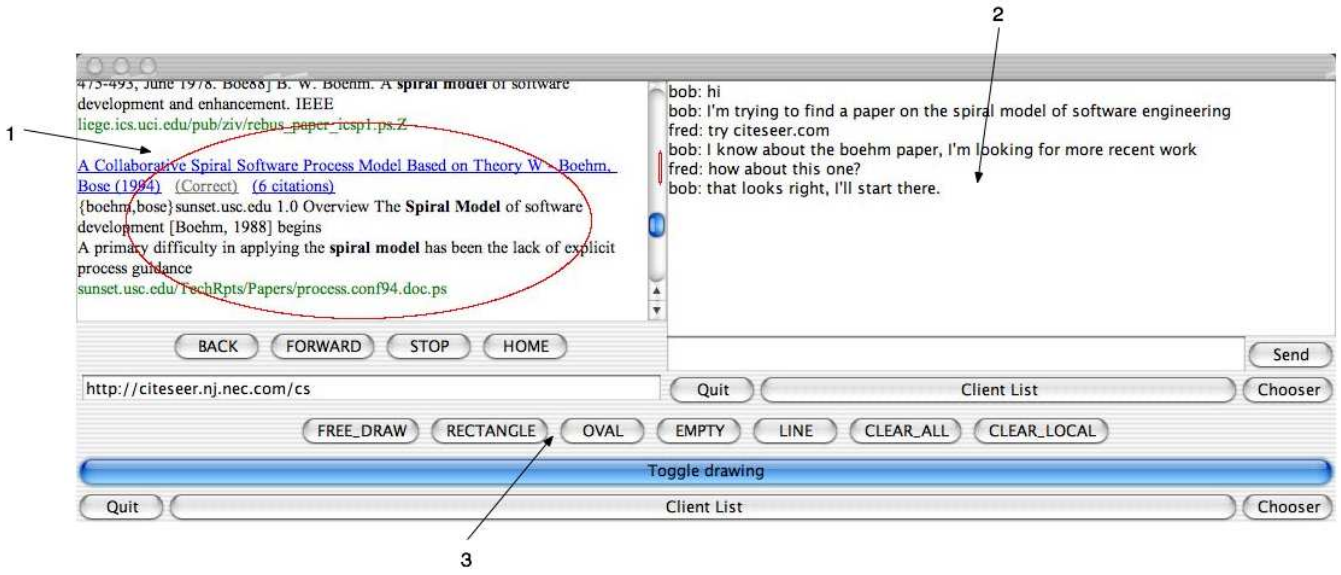

Figure 4. Screenshot of the Online Research Assistant

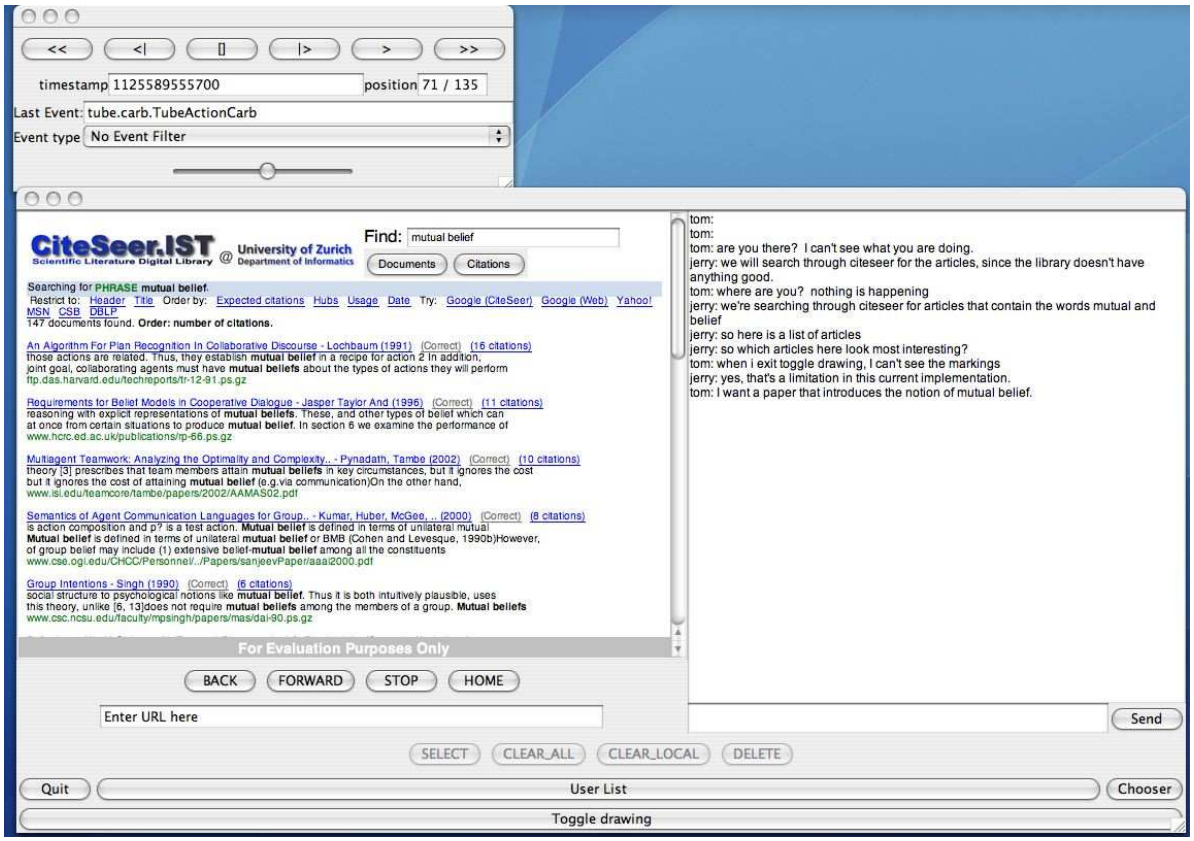

Figure 5. An Analysis Session for the Online Research Application