

Using Transcription and Replay in Analysis of Groupware Applications

Seth Landsman
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
+1-781-271-7686
landsman@mitre.org

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.

This paper details a model for collecting a replayable transcript of online collaboration. This transcript can then be used to study the interaction, through techniques such as ethnographic analysis. We also present software frameworks that implement this model. An example of an analysis is presented, as are two examples of experimental groupware applications that use this model.

Categories and Subject Descriptors

H.5.3 [Information interfaces and presentation (e.g., HCI)]: Group and Organization Interfaces

General Terms

Groupware, Analysis, Transcription, Replay

1. INTRODUCTION

Collaborative processes are not fixed entities. There is a constant and considerable need to study and evaluate how people work together and how group work can be improved to better fit a community of users. A number of techniques and approaches have been explored in this area, from the understanding of interaction [5], distributed cognition [10], and ethnographic study of an artifact's use in group activity [25].

A collaborative process that is mediated by a software application

(i.e., groupware [12]) also needs to be studied and improved. External perspectives of activity, such as video taping a user, become less useful as the interaction may be distributed and requires both context of a user's activity and correlation between users' activities. Ethnographic analysis has been shown to be effective in the study of off-line interaction (i.e., not mediated by software), and can be applied to online behavior given the proper software support.

Ethnography for off-line collaboration generally captures the activity of the users (transcription) and provides a mechanism for further review (replay). Our approach is to provide similar capabilities for the analysis of online collaboration by collecting the activity of the users as they collaborate into a continuous transcript. Review of the transcript is accomplished by a software system that replays the transcript in the same context as the transcript was captured.

We have built several groupware applications to study collaboration and groupware issues. In each case, the application produced complete, replayable transcripts. The replay of these transcripts were used for a variety of tasks: as a basis for redesigning an application, as part of a research study on online collaboration, as data that is used for teaching analysis techniques, and as resource data for term projects. In addition to development of several groupware applications, we have developed a pair of toolkits: THYME [15] and SAGE. THYME is used to generate groupware applications that automatically produce complete, replayable transcripts. SAGE provides the basis for constructing a replay application. Both toolkits have been used to create numerous applications. The THYME toolkit was also used in an undergraduate HCI class.

This paper extracts a general model for building groupware applications that automatically produce replayable transcripts. This model supports the transcription of user activity, both "low-level" activity such as mouse clicks and "task-level" activity such as chat events. Replay of the application occurs through the basis groupware application used to collect the transcript. The model provides methods to enhance the replay of the transcript given the structured, introspectable nature of the transcript, such as searching for types of events in the transcript and annotation of interesting aspects of the transcript. The THYME and SAGE toolkits are an implementation of this model.

The remainder of this paper starts with a simple example of how transcription and replay can be used to support the redesign of a

groupware application. This example shows some of the capabilities of the model of transcription and replay. Following that, our model is detailed and the implementation is discussed. Some examples of other applications that use our analysis techniques are then described. This paper concludes with a discussion of future work.

2. ANALYSIS OF GROUPWARE

This section provides a simple, concrete illustration of the use of transcription and replay.

In the Fall of 2002, a Human-Computer Interaction (HCI) class held at Brandeis University implemented synchronous groupware applications as part of their term projects. These students were given access to the THYME groupware framework and only twenty-eight days of implementation time. The class was divided into fourteen teams of 2 - 3 students per team. Despite the short amount of implementation time, twelve out of the fourteen teams successfully implemented a groupware application that could collect replayable transcripts of use.

One group in this class produced an application called the **Online Research Assistant (ORA)**. This application 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 1. The left side of the screen (see 1) is the web browser with a shared whiteboard in a transparent overlay. The browser is a relaxed WYSIWIS [23] component, in that the web browser context is synchronized between users, but the scrolling of the web page is not. 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 artifacts and button to toggle the whiteboard's display.

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 (i.e., entering a new URL, and going forward and backwards in the history of visited URLs).

From a comparison of the ORA replay application (Figure 2) and the basis ORA application (Figure 1), it is clear that they share a common lineage. As shown in Figure 2, the center replay window is a direct analogue of the ORA client screen, and contains individual components that are leveraged from the basis application.

The top left object in Figure 2 is the replay controller. It allows basic playback of a transcript through a VCR-like metaphor. The top buttons are recognizable as varying speeds of replay, both forward and backward, as well as a button to stop the replay. Also of note is the *Event Type* field, which allows the analyst to move forward or backward in the transcript to the next instance of a specific type of event. More detail on how this controller operates is discussed later in this paper.

2.1 Example Analysis

We performed data collection and analysis 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 *Bob*, are using the baseline version of ORA. Bob 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 replay application by *italicizing* his actions. Figure 2 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 completion of the task

The first part of the session starts with the interaction between Tom and Bob (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.

Bob 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 Bob 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 Bob 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. Bob 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.

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

Tom where are you? nothing is happening

Bob 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 Bob's actions. He continues to *step forward*, observing that Bob 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 Bob 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?

Bob 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 subject matter 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?

Bob I do not think citeseer has that capability.

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

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

3. MODEL OF 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 through 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.

3.1 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 that information is not sufficient to reliably reconstruct whether the user's task was planning or chatting without further context.

jRapture [24], Playback [20], and others [21] provide complete transcripts of user interface events only. jRapture replaces an application's underlying standard 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 [7] and Chimera [14] provide a complete transcript only with respect to an enumerated set of task events. They 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 then 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 replay 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 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 if the transitions are not reversible. Commonly, the application’s state would need to be reset and the playback re-run from the start of the transcript until it reaches the state the analyst chose to examine.

In addition to the two extremes of storing transitions or states, a hybrid approach of storing *checkpoints* is also seen in some transcription implementations. Checkpointing is the storing of occasional states of the application’s execution as well as transitions between those states. The resulting transcript allows for faster movement between positions in the transcript and definitive points of coordination between different parts of a distributed application. Examples of this technique are found in the BugNet application [13], as well as Chandy and Lamport’s work [4], and in others [22] [26].

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

3.2 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 replay 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 replay 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 replay tool to search for and notes for other analysts use.

The video tape solution provided by Suchman and Trigg [25] 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 replay 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 replay 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 replay tools allow the analyst 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 [19] 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.

4. ENGINEERING TRANSCRIPTION AND REPLAY

To engineer online ethnographic analysis capabilities into an application and its development lifecycle, the appropriate transcription and replay technologies 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.

4.1 Transcription and Replay Requirements

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

| 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

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.

4.2 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 run-time. These transcript, as described above, are complete, encode task and user interface events, and are transitional. They contain only online user interaction.

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.

4.3 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 [15].

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 by which the transcript is stored and reconstructed and allowing different transcription formats and strategies to be used without changing the application. Figure 3 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 replay application.

Within the THYME application, transcripts can be accessed during the run-time of the application through a component called the transcript emitter. This component 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

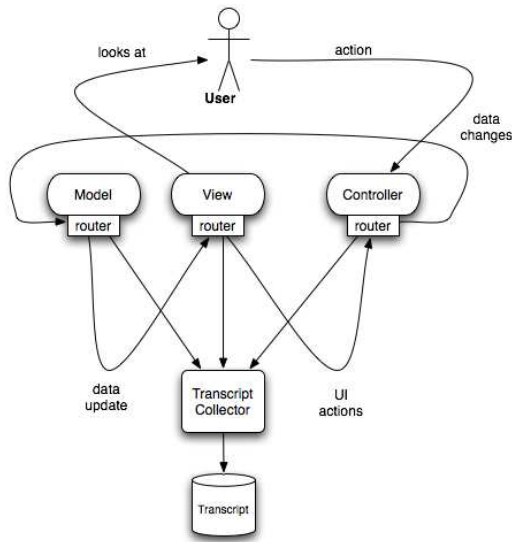


Figure 3: Transcripting messages in a THYME application

makes use of the transcript emitter to playback the messages contained in the archived transcript, in the order those messages were generated by the participants in the archived session.

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 $TRANSCRIPT_{T \in [X, Y]}$ and represents the transcript for all messages between M_X and M_Y inclusive. $TRANSCRIPT(X)$ is used for referring to the message M_X that is stored in the transcript.

Given that a transcript is a collection of messages that is sent throughout the application, a transcript can be shown to be complete if it captures all interaction throughout the application. The set of messages captured in a transcript during the runtime of an application a is represented as $TRANSCRIPT_a$. If each instant of change that occurred during the runtime of a is contained in $TRANSCRIPT_a$, then $TRANSCRIPT_a$ is a complete transcript of the runtime of a .

The transcript collects into a transcript containing both interface events and domain actions. Interface events can be encapsulated as messages to an *interface controller* and thereby are collectable in the transcript. Because messages between components are encoded in terms of the representation system of the application, the information contained in the message includes domain action information. For example, a message passed from one client’s chat room component to another client’s chat room 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.

4.4 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, $\{C_0, C_1, \dots, C_n\}$. Therefore, applying a message M to an application A is represented $A(M) = A'$, which is also written as $A(M) = \{C_0(M), C_1(M), \dots, C_n(M)\}$.

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 $EVENT-TYPES(T)$, which are mined from the transcript T . The available set of $EVENT-TYPES$ 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.

4.4.1 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 as 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* [9]. 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.

The milestone process is related to *checkpointing* [18] a database to ensure consistency of the database state. Milestones may, unlike checkpoints, change in number and temporal location during execution. Global system snapshots [4] [26], a technique used in distributed debugging, is also similar, in that it looks to collect a

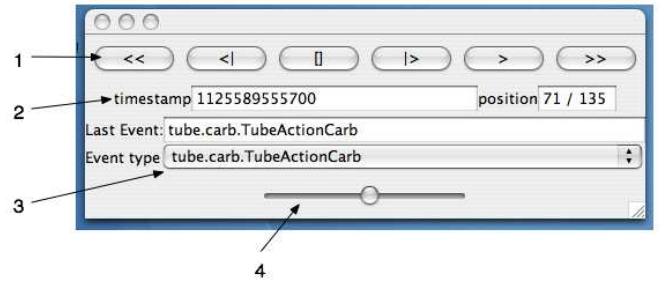


Figure 4: SAGE Playback Controller

consistent state across multiple systems. The milestones described here are not used to create a unified system state, although they do that as a consequence because of the simple nature of the SAGE application. Instead, milestones are designed to provide an accessor for a set of temporal positions within a single model.

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.

4.4.2 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 4) 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-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 replay 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

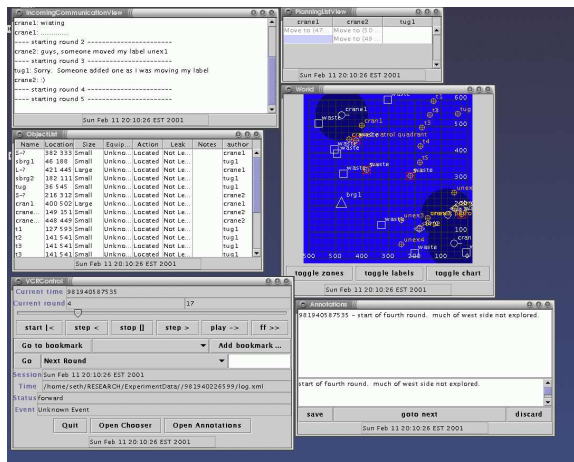


Figure 5: The VW-SAGE system

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

5. APPLICATIONS

The transcription and replay model has been successfully used in a number of projects. This section shows two example projects that have successfully used this model to analyze their use.

5.1 VesselWorld

The first example of this model of transcript and replay was implemented in the VesselWorld group problem solving system [2] [16]. The problem domain encoded in the VesselWorld application has three participants engage in a computer-mediated problem solving session. To complete a set of tasks in this simulated environment, the participants must communicate and jointly problem-solve. The only avenue of communication is via the application client. Access to the environment, and objects in the environment, is also mediated through representations provided by the software application. The problem solving sessions require cooperation, coordination and collaboration.

There have been multiple experiments run with the VesselWorld application [1] [8] [11], all of which have leveraged the analysis capabilities of the application. The replay application associated with VesselWorld can be seen in Figure 5. The replay of transcripts was used for both redesign tasks and analysis of experimental data. Over the course of these experiments VesselWorld has been modified several times to support different types of collaborative interactions and different types of analysis. Over two hundred hours of VesselWorld data has been collected across these versions of the VesselWorld application.

5.2 CEDAR

A second application, CEDAR [17], was developed using THYME as a collaborative WikiWikiWeb editor [6]. It enables multiple

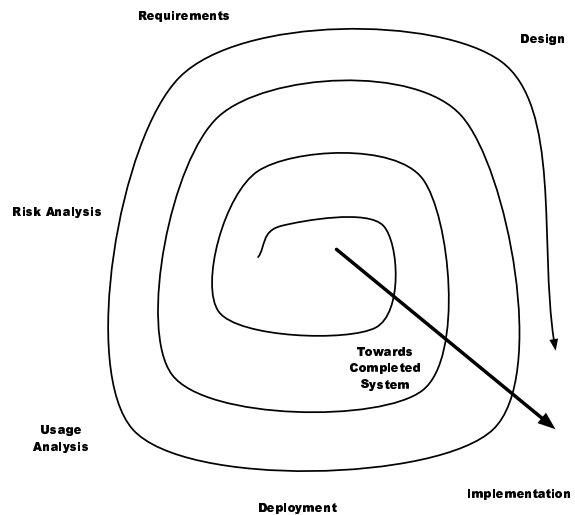


Figure 6: Spiral Lifecycle with Ethnographic Analysis

users to research information on the web and edit Wiki pages while in communication via both shared web browsers and a chat channel. Additionally, the web site structure is visible to all the users, showing them the links between pages and groupings in the web site.

CEDAR was used in a class at Brandeis University to study collaboration in a cooperative task environment. The class used CEDAR to collect transcripts of online collaboration; in all, approximately fifty hours of data were collected. The replay of these transcripts were used to teach analysis methods like conversation analysis, support discussion on theoretical topics like awareness, and as a basis for student term projects.

6. CONCLUSIONS

This paper discussed our approach to 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 [8].

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 [3], 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 [15] 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.

7. ACKNOWLEDGEMENTS

This work was supported under ONR grants N00014-00-1-8965 and N00014-96-1-0440, NSF grant EIA-0082393, and MITRE Innovation Grant 03MSR309-A6.

8. REFERENCES

- [1] Richard Alterman, Alexander Feinman, Seth Landsman, and Joshua Introne. Coordination of talk: Coordination of action. Technical Report TR-02-217, Brandeis University, 2001.
- [2] Richard Alterman, Seth Landsman, Alexander Feinman, and Joshua Introne. Groupware for planning. Technical Report Technical Report CS-98-200, Computer Science Department, Brandeis University, 1998.
- [3] Barry Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, 1988.
- [4] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63 – 67, February 1985.
- [5] Herbert H. Clark. *Using Language*. Cambridge University Press, 1996.
- [6] Ward Cunningham. WikiWiki. <http://c2.com/cgi/wiki?WikiWikiWeb>.
- [7] 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.
- [8] Alexander Feinman and Richard Alterman. Discourse analysis techniques for modeling group interaction. In *Ninth International Conference on User Modeling*, 2003.
- [9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.
- [10] Edwin Hutchins. *Cognition in the Wild*. MIT Press, 1996.
- [11] Joshua Introne and Richard Alterman. Leveraging collaborative effort to infer intent. *Ninth International Conference on User Modeling*, 2003.
- [12] Robert Johansen. *GroupWare: Computer Support for Business Teams*. The Free Press, New York, NY, USA, 1988.
- [13] Stuart H. Jones, Robert H. Barkan, and Larry D. Wittie. Bugnet: A real time distributed programming environments. In *SRDS*, pages 56–65, 1987.
- [14] 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.
- [15] Seth Landsman. *Building Groupware on THYME*. PhD thesis, Brandeis University, 2005.
- [16] Seth Landsman, Richard Alterman, Alexander Feinman, and Joshua Introne. Vesselworld and ADAPTIVE. Technical Report TR-01-213, Dept of Computer Science, Brandeis University, 2001. Presented as a demonstration at *Computer Support Cooperative Work 2000*.
- [17] Johann Ari Larusson and Richard Alterman. Integrating collaborative technology into the interdisciplinary classroom. In Preperation.
- [18] Raymond A. Lorie. Physical integrity in a large segmented database. *ACM Trans. Database Syst.*, 2(1):91–104, 1977.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] M. Satyanarayanan, D. Steere, M. Kudo, and H. Mashburn. Transparent logging as a technique for debugging complex distributed systems. In *Proceedings of the 5th workshop on ACM SIGOPS European workshop*, 1992.
- [23] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. Wysiwis revisited: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems*, 5(2):147 – 167, 1987.
- [24] 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.
- [25] 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.
- [26] Zhonghua Yang and T. A. Marsland. Global snapshots for distributed debugging. In *Fourth International Conference on Computing and Information*, pages 436 – 440, 1992.

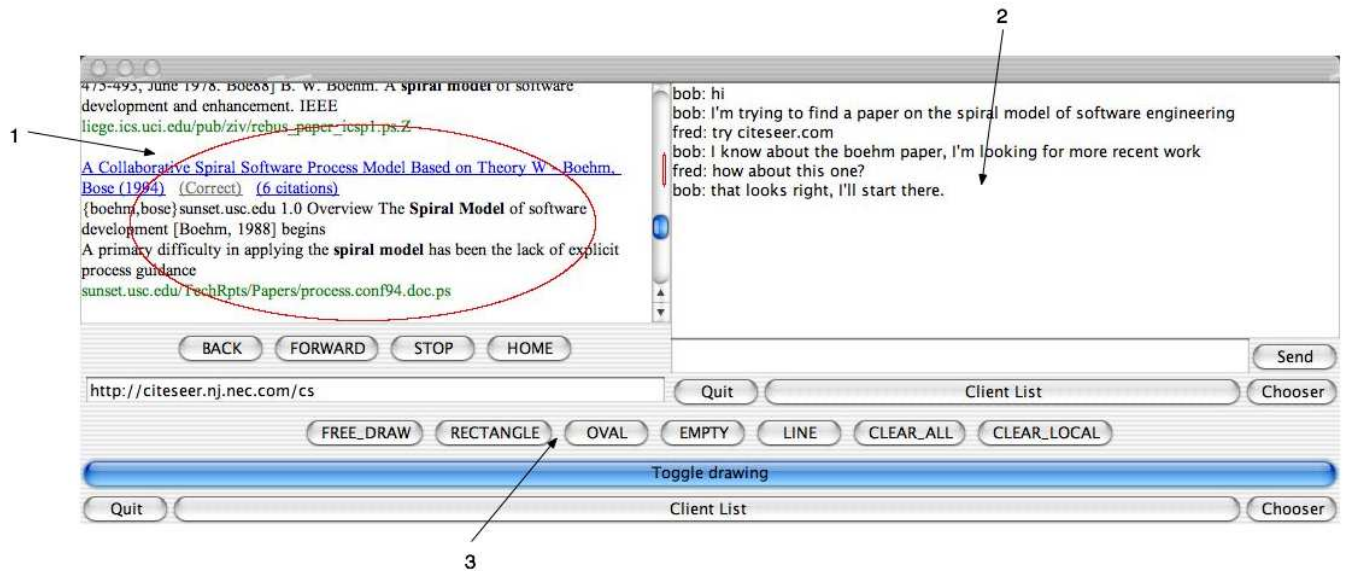


Figure 1: Screenshot of the Online Research Assistant

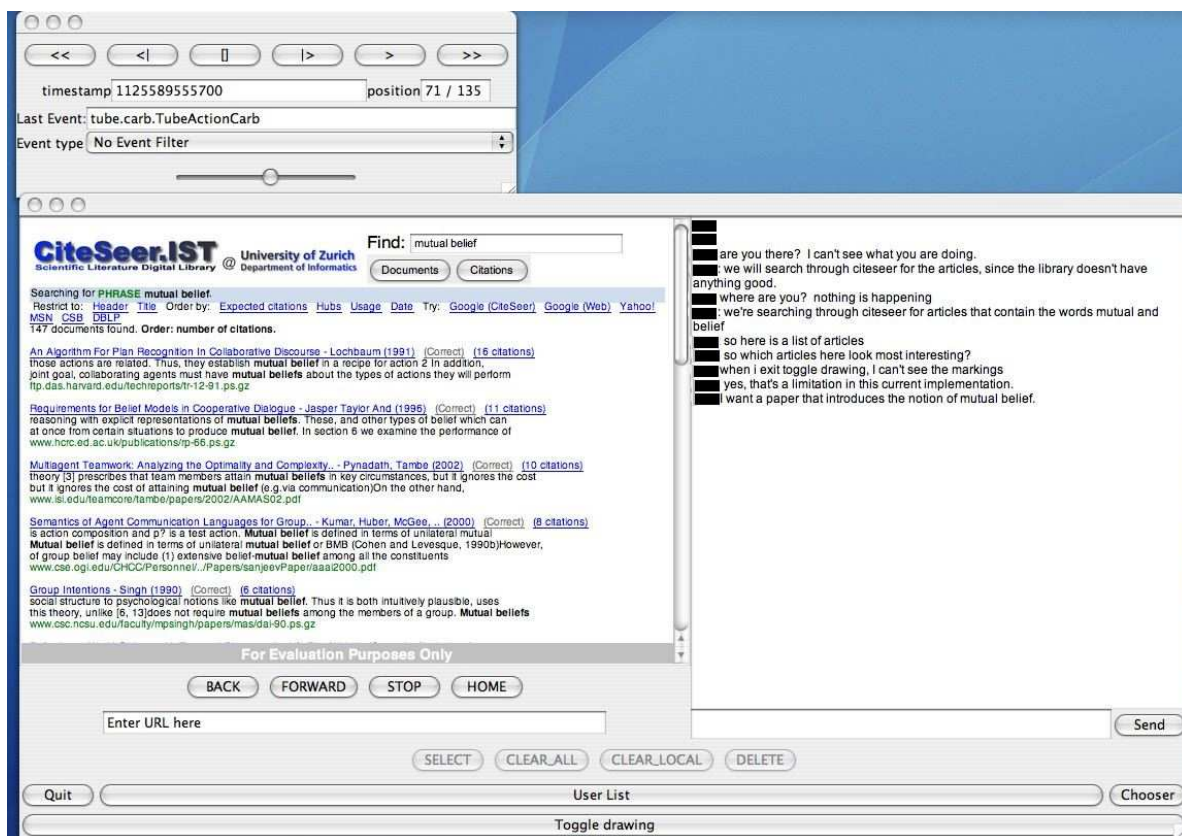


Figure 2: An Analysis Session for the Online Research Application