DATA LOGGING: HIGHER-LEVEL CAPTURING AND MULTI-LEVEL ABSTRACTING OF USER ACTIVITIES

Jim Helms, Dennis C. Neale, Philip L. Isenhour, & John M. Carroll
([jhelms, dneale, isenhour]@vt.edu; carroll@cs.vt.edu)
Center for Human-Computer Interaction, Department of Computer Science
Virginia Tech, 660 McBryde Hall, Blacksburg, VA 24061-0106

ABSTRACT

Data logging has been a standard, but under utilized, software evaluation technique for single-user systems. Large volumes of objective data can be collected automatically and unobtrusively. This data, however, is usually in the form of low-level system events, making it difficult to analyze and interpret meaningfully. In this paper we extend traditional logging approaches to collaborative multi-user (groupware) systems. We also show how data captured at a higher level of abstraction can characterize user-system interaction more meaningfully. Lastly, we show how higher-level data abstracted from logging can be more effectively combined with data from other usability methods.

Cite As:

INTRODUCTION

A range of methods are now commonly used to improve software usability, such as heuristic evaluations and think aloud. Although these methods are useful, instrumenting software with embedded metering can augment these methods by providing continuous, objective, accurate, and unobtrusive data collected from both users’ normal work contexts and in controlled laboratory environments (Nielsen, 1993). Logging also provides quantitative and qualitative data regardless of research setting.

Typically, logged data is used in three ways. First, data logs can be studied to gain an understanding of how the software system is operating (Hilbert and Redmiles, 1998). Technical problems can be identified allowing developers to optimize certain aspects of the software. Second, researchers often examine the logs using pattern analysis (Siochi and Hix, 1991), resulting in a quantitative record of user repetition that can be useful in identifying interface problems. Lastly, data logs can undergo sequential event-based analysis (Sanderson and Fisher, 1994), providing rich contextual descriptions of users' actions.

Third-party monitoring software that runs on top of the application of interest can be used to log system events, or the application code of the software being evaluated can be modified directly (Nielsen, 1993). The data gathered usually consists of large amounts of low-level system events such as keystrokes and mouse movements that are cryptic and difficult to analyze in a meaningful manner. Extracting and filtering important data segments from the large amounts of irrelevant data is crucial to using logged data effectively, but it is difficult (Hilbert and Redmiles, 1998). Once relevant information is identified, it can be even more difficult to interpret the data from a qualitative standpoint because logged data does not convey this information well (Hammondtree, Hendrickson, and Hensley, 1992). Lastly, a significant goal for evaluators, but one that is rarely achieved effectively, involves combining logged data with other measures of usability for data analysis and interpretation.

Logging procedures and related issues typically have been addressed in the context of single-user systems. Large-scale, multi-user systems distributed over the Internet significantly complicate logging. Automated data collection at the client is not practical in most cases because events have to be transmitted via the Internet to a collection site, and this approach consumes considerable network bandwidth. As an alternative, we pursued a higher-level, server-based approach.
In this paper we discuss data logging as part of a comprehensive strategy for evaluating groupware. We describe how events passed between collaborating users served as a rich source of log data. These first-order raw events were filtered and formatted to produce second-order chronologies of user actions. Merging data from other sources with these chronologies produced third-order activities that provided comprehensive descriptions of collaborative sessions. Macro filtering logged data to produce sequential user actions and the integration of the transformed data with other measures of usability to reconstruct activities were central to our approach. We outline three important contributions to current data logging practices: 1) innovative extension of logging to distributed, multi-user applications, 2) logging of complete event traces at a higher level of abstraction, and 3) comprehensive integration of logged data with a multitude of other data sources. In addition, we developed a model to characterize the uses of data logging, illustrating how higher-order abstractions of logged data based on these contributions extends and enriches current data logging approaches.

PROJECT OVERVIEW

This study was conducted as part of the Learning in Networked Communities research project at Virginia Tech. User session logs were collected from a distributed Java-based learning environment called the Virtual School that integrates a collaborative science notebook, Web-based discussion forums, e-mail, synchronous chat, video conferencing, and application sharing (Isenhour, et al., 2000a; Koenemann, et al., 1999). Student groups spanning classroom and school boundaries used the Virtual School to conduct distributed science projects. Experts from the community also served as mentors and used the software to collaborate with students remotely.

METHOD

Hilbert and Redmiles (1998) suggest evaluators separate application and instrumentation code to allow both to evolve independently. Network-based groupware applications can support this separation implicitly. The synchronization of multiple collaborating sessions requires that a representation of each user's activities be shared with all collaborators. Representations of user activity, in varying degrees of granularity, are encapsulated in messages that are sent across the network. These can be intercepted for logging by a component that is independent of the application itself. If a collaborative application transmits low-level user interface events to all participants (e.g., as in application-sharing packages like Microsoft's NetMeeting) then logging the messages describing such events produces a huge quantity of data that is difficult to analyze. If, however, the application transmits messages that describe a collaborator's actions at the level of changes to shared data, then the application inherently provides data that is more indicative of human behavior.

The components of the Virtual School use an object replication toolkit called CORK (Content Object Replication Kit) to synchronize data among collaborating sessions (Isenhour, et al., 2000b). Clients contact the server to retrieve copies of objects representing each piece of shared content. When the local copy of one of these objects is modified, the CORK infrastructure detects the change and sends a message to the server. This message is itself a Java object that includes data describing the change and logic for reproducing changes on other replicas of the object. The message is first used to apply the modification to the master copy of the changed object on the server. The message is then propagated to any other active clients who have retrieved a copy of the object. By capturing messages as they pass through the server, we were able to generate logs of all changes to shared data without instrumenting client-side software. The server knows the shared content object to which each change message applies, from which machine the message originated, and which user initiated the change. Where appropriate, snapshots of the modified objects themselves can also be logged at the server, again imposing no overhead on the client. Overhead on the server is also limited since messages need not be logged immediately. In the Virtual School server, messages are placed on a queue after all other processing. When server load is light, messages are pulled off of this queue, logged, and discarded.

Server-side logging of network interactions is a technique that has been widely employed in web servers and similar client-server applications. Collaborative systems differ because they tend to have more complex protocols. Whereas a Web browser communicates with a server using a fixed set of eight basic textual messages (e.g., GET, POST, and PUT), each communication or authoring tool in the Virtual School uses its own arbitrarily complex set of messages, each implemented as a Java object with both data and behavior. Implementing messages as objects helps manage the complexity introduced by the diversity of messages, since the behavior implemented by a message object can include logic that generates additional distinct log information. For example, a message describing an addition to a chat session...
can dump the text of the chat message to the logged form of
the message. This is a significant advantage of an object-
oriented approach to messaging because it allows new types of
messages (e.g., for new tools in the Virtual School) to be
introduced without affecting log generation components.

Logging the changes to shared data captures most of the
user actions of interest. However, actions that involve the
browsing of shared data, where users view data but do not
modify it, produce no messages and therefore no logs. A
record of these actions was also necessary for complete
analysis, so the Virtual School client was modified to produce
browsing messages for the logs.

The format of the Virtual School logs captured all
information about each modification, including a timestamp,
user id, host name, unique identifier of the modified content
object, and any additional descriptive data provided by the
message object. Figure 1 shows the raw form of the log for a
chat entry (event). This raw data is post-processed to produce
human-readable records (action) (see Figure 2). Post-
processing was originally done with a set of Perl scripts. More
recently we have implemented a Java servlet for log
processing that provides better integration with the Virtual
School server and simplifies Web access to log data. Both the
Virtual School software and the needs of the evaluators and
developers evolved over the period of the study, resulting in
several significant revisions to the post-processing tools that
produced more useful statistics and more readable output.

The post-processing tools discard a variety of uninteresting
entries, including automatically generated changes to shared
data and entries representing activity by developers and users
peripheral to the study. To enhance readability, user profile
data was matched to find real names, project group names, and
schools based on user identifiers in the raw logs. This
information was then added to the logs. Figure 2 shows the
translated form of a chat message. For entries representing
changes to notebook content, the unique content object
identifier stored in the raw log entry is used to retrieve a
snapshot of the content for inclusion in the finished log. The
final product consisted of a chronology of the session with
statistics summarizing the frequency of each type of user
action.

Figure 2: Example of formatted log action.

We collected data from roughly twenty-five sessions (ten
project groups consisting of two machines distributed across
locations, plus five mentors) for twenty-two weeks. Each
session consisted of a few hours of machine use a week. The
result was 9.7 Mb of raw data describing 41,489 events. After
post-processing we were left with 3.6 Mb of transcripts, which
contained translated forms of 14,496 actions and snapshots of
modified notebook content.

Quantitative data was easily extracted from the logs
because of the uniformity with which the data was captured
and transformed. Qualitative analysis of logged data has
traditionally proven more difficult (Hammond, Hendrickson,
and Hensley, 1992; Castillo, Hartson, and Hix, 1997), and when one collects data at the level of keystrokes
and mouse movements, it becomes nearly impossible. The
difficulty lies in relating low-level information to high-level
human behaviors (Yoder, McCracken, and Akscyn, 1985).
Our higher-level logging facilitates quantitative and
qualitative analysis by recording events at a more meaningful
level and then transforming them into easily interpreted
actions. Thus, less has to be inferred about a user's intention
from a recorded action.

DATA ANALYSIS

Data logging provided a wealth of rich information, but
logging requires complementary methods to guide in the
interpretation process crucial to evaluation (Wright and Monk
1989). In addition to logs, we collected real-time contextual
interviews, videotapes of user sessions, field notes, student
artifacts, and critical incidents. Our distributed system forced
us to substantially modify existing single-user methods and
create new procedures for mechanically and analytically
bridging methods and data. Integral to these procedures were
the integrated activity scripts, and coding methods used to
integrate the evaluation processes.

Integrated activity scripts documented episodic
chronologies constructed from multiple data sources. Field
observations were video recorded with time stamps that were
synchronized with the server clock. Field observations,
contextual interview, supplementary field notes, and comments were transcribed for each group interaction, and time stamps were assigned during transcription to each passage of dialog and each relevant occurrence. This allowed field observations to be collated intricately and chronologically with server logs. This combined record resulted in a comprehensive transcript of the distributed user groups’ activity including verbalizations, observed behaviors, dialog, and computer server events described above. Thus, the various sources of relevant data for each interaction were organized and consolidated into a single document that provided investigators with a comprehensive script of the distributed interaction (Neale and Carroll 1999).

Once a textual record was produced for each interaction, the document was imported into qualitative analysis software for coding analysis. Coding is the process of systematically organizing the data, reducing it into manageable units, synthesizing the units, detecting patterns, and presenting the information for interpretation. The completeness of the information and identifiers contained in the scripts allowed easy manual and automated coding of passages according to type of data, source, and chronology. Coding software allowed investigators to generate a variety of quantitative measures of use, but it also provided a means of organizing the content of the scripts and relating occurrences in and among scripts for qualitative analysis.

Logged data was central to the process of combining and coding many types of data. In addition to providing an initial structure for the chronology of the distributed interactions, logged data filled holes left by field observation techniques. It allowed evaluators access to computer interactions without confusing and irrelevant stores of data. In some cases the logs were the only source of data on a group (i.e. unobserved groups); here the logs provided, at the least, a chronological record of user interaction. This lone interaction could then be coded and combined with like data categories observed in other groups. The flexibility and completeness of our multi-faceted evaluation process ensured that interactions were properly situated and documented so that analysis could be comprehensive and relevant.

**DISCUSSION**

Logging has been used successfully in a number of application areas, including development of raw text editors (Good, 1985) and creation of interactive task support systems (Nielminen, Kasvi, Pulkis, and Vartiainen, 1995). Although these studies involved single-user systems, they nevertheless faced large amounts of low-level data. Using this approach for multi-user applications is problematic because these applications are often sensitive to network load. Transmitting logged data in addition to system events can considerably degrade performance. With client-server systems, data can be logged at the server, eliminating the need to burden the network and instrument client software.

Over the course of our study we developed a three-tiered model to characterize the process and use of data logging. This model can be useful for understanding how logged data must be captured, transformed, and fully utilized. Our model consists of three processes that iteratively raise the data to higher levels of abstraction, providing more meaningful information at each stage. First-order raw events, or capture-level processes, must capture user behavior from system-level events, not all of which are of interest to the usability engineer. Second-order transformed actions refine the raw data by filtering and formatting the logged data for human readability or statistical analysis. This step includes removing irrelevant events, reformatting, and adding information that allows data to be used in both quantitative and qualitative analysis.

Third-order, or activity-level transformations, combine transformed data produced by second order processes with other usability data to create a more meaningful transcript of user sessions. Examples of other usability data include video recordings of group sessions, contextual inquiry, student and mentor interviews, screen capture, survey data, and think aloud. The integration provided by third order processes gives the researcher a combined observer and system recorded view of the session.

In multi-user systems, event-level behavior captured at the server, transformed through filtering into user actions, and combined with other data to characterize user activities, can be categorized into user moves, artifacts generated, and computer-mediated human-human communication. Moves consist of opening software tools, browsing user generated content, and initiating conversations. Artifacts are work products produced by users, such as drawings or collaboratively authored text. Communications consist of chat messages, e-mail, annotations, and video conferencing. Capturing events at the system and transforming them into actions allows a more complete understanding of human performance. We were able to build a meaningful chronicle of system events and user actions, and we were able to collect human dialogue surrounding system use as well. Using these categories we can compare what the user did (moves), what
they generated as content (artifacts), and what their dialog was like when engaged in these activities (communications).

The logged data was useful in filling in gaps in our observer field notes and in clarifying scenes viewed on videotapes. For example, during the course of the study many of the student groups began to rely on chat sessions as their main method of communication. This made any type of qualitative analysis difficult since the videotapes did not contain the students’ conversations. The logs gave us the entire discourse, allowing us to easily document user communication that was not available on video.

In another case, we recorded on videotape a middle school student complaining that her high school counterparts would not answer her questions. A field evaluator noted this as a critical incident. The observer at the high school recalled the students there had responded, but that the middle school student had failed to reply to them. The logs showed the high school students had indeed responded to the middle school student’s question, but their response had not come until just after the middle school student had logged out of the system. The timing was so close that the evaluators could not resolve the issue by viewing the videotape, but immediately detected what had happen by going back to the logs. This information will be used to re-design our communication components to prevent trailing off messages from becoming lost when users leave the system.

CONCLUSIONS

The distributed nature of multi-user applications presents challenges for usability evaluation, but opportunities exist for efficiently gathering large quantities of information by intercepting messages exchanged across collaborator’s clients. The raw event data gleaned from messages can be processed to improve readability and narrow the set of actions evaluators are really interested in analyzing. Combining this output with observer notes, video transcripts, and another other available data about a collaborative interaction produces a more complete reconstruction of activities.

The advantages of creating a complete chronology at the action level have become apparent in our situation and hold potential for extended work in this area. In the case of our system we were able to extract significant amounts of both quantitative and qualitative logged data related to remotely situated collaborative users. This data, combined with the other evaluation techniques we incorporated (as in the integrated activity scripts and coding,) have opened a doorway into a vastly untapped pool of information.

ACKNOWLEDGEMENTS

We are grateful for contributions to this work from Dan Dunlap and for grant support from the Hitachi Foundation, the National Science Foundation, and the Office of Navel Research.

REFERENCES


