

A Survey of Computer-Aided Techniques for Extracting Usability Information from User Interface Events

David M. Hilbert

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1 714 824 3100
dhilbert@ics.uci.edu

ABSTRACT

This survey examines computer-aided techniques used by HCI practitioners and researchers to extract usability-related information from human-computer interaction events. A framework is presented to help HCI practitioners and researchers categorize, evaluate, and compare the strengths and limitations of approaches that have been, or might fruitfully be, applied to this problem. An agenda for future research in this area is also presented.

Keywords

User interface event monitoring, sequential data analysis, usability evaluation, human-computer interaction

1 INTRODUCTION

User interface events (UI events) are typically generated as natural products of the normal operation of window-based user interface systems such as those provided by Microsoft Windows, the Macintosh Operating System, the X Window System, and the Java Abstract Window Toolkit (AWT). Such events indicate user behavior with respect to the components that make up an application's user interface (e.g. movements of the mouse with respect to application windows, keyboard events with respect to application text fields, mouse selections with respect to application buttons, menus, and lists). Because such events can be automatically captured and because they indicate, albeit at low levels of abstraction, user behavior with respect to an application's user interface, they have long been regarded as a potentially fruitful source of information regarding application usage and usability. However, because user interface events are typically extremely voluminous and rich in detail, automated support is generally required to extract information at a level of abstraction that is useful to an investigator interested in analyzing application usage or evaluating usability.

While there are a number of potentially related techniques that have been applied in other domains to the problem of analyzing sequential data, this paper surveys techniques that have been applied within the domain of HCI. Providing an in-depth treatment of all potentially related techniques would necessarily limit the amount of attention paid to characterizing the approaches that have in fact been brought to bear on the specific problems associated with analyzing UI events. However, this survey will attempt to characterize

UI events and analysis techniques in such a way as to make comparison between techniques used in HCI and those used in other domains straightforward.

1.1 Goals and Research Method

The fundamental goal of this survey is to construct a framework to help HCI practitioners and researchers categorize, evaluate, and compare the strengths and limitations of approaches that have been, or might fruitfully be, applied to this problem. Because exhaustive coverage of all existing and potential approaches is impossible, an attempt has been made to identify key characteristics of existing approaches that divide them into more or less natural categories. This allows classes of systems, not just instances, to be compared. The hope is that illuminating comparison can be conducted at the class level, and that classification of new instances into existing classes will prove to be unproblematic.

The basic strategy has been to search as exhaustively as possible the literature in both academic and professional computing forums (including magazines, conference proceedings, and journals) for papers describing computer-aided techniques for extracting usability-related information from user interface events. An initial set of papers were selected and analyzed to identify key characteristics (including goals, features, and techniques) that distinguish the approaches applied by various investigators.

A two-dimensional matrix was constructed with instances of existing approaches listed along one axis, and characteristics listed along the other. This led to an initial classification of approaches based on visually apparent clusters of related attributes. The comparison attributes and classification scheme were then iteratively refined based on further exploration of the literature. The resulting matrix indicates areas in which further research is necessary and suggests synergistic combination of currently isolated capabilities.

1.2 Comparison Framework

The framework used to classify and compare approaches is presented in more detail in Section 4. This subsection introduces the high level categories that have emerged as a result of this survey.

1.2.1 Techniques for synchronization and searching

These techniques allow user interface events to be synchronized and cross-indexed with other sources of data such as video and/or coded observations. This allows searches in one medium to locate supplementary information in others. In some ways, this is the most simple (i.e. mechanical) technique for exploiting user interface events in usability evaluation. It is, however, quite powerful.

1.2.2 Techniques for transforming event streams

Transformation involves filtering, abstracting, and recoding event streams to facilitate human pattern detection, comparison, and characterization, or to prepare event data as input to automatic techniques for performing these functions. *Filtering* operates essentially by subtracting information from the event stream, allowing events and sequences of interest to emerge from the “noise”. *Abstraction* operates essentially by “synthesizing” new information that might be added to the event stream based on patterns of events, as well as other contextual information. *Recoding* involves generating a new event stream based on filtering and abstraction in order to allow filtered and abstracted events to be subjected to the same types of manual and/or automated analysis techniques normally performed on raw event streams.

1.2.3 Techniques for performing counts and summary statistics

Once user interface events have been captured, there are a number of counts and summary statistics that might be performed to summarize user behavior, e.g., feature use counts, error frequencies, use of the help system, and so forth. Although most investigators rely on general-purpose spreadsheets and statistical packages to provide such functionality, some investigators have proposed specific “built-in” functions for calculating and reporting this sort of summary information.

1.2.4 Techniques for detecting sequences

These techniques allow investigators to identify occurrences of concrete or abstractly defined “target” sequences within “source” sequences of events that may indicate potential usability issues. In some cases, target sequences may be quite abstractly defined and are supplied by the developers of the technique. In other cases, target sequences may be more specific to particular applications and are supplied by the users of the technique. Sometimes the purpose is to generate a list of matched source subsequences for further perusal by the investigator. Other times the purpose is to automatically recognize particular sequences that violate expectations about proper user interface usage. Finally, in some cases, the purpose may be to perform transformation of the source sequence by abstracting and recoding instances of the target sequence into “abstract” events.

1.2.5 Techniques for comparing sequences

These techniques help investigators compare a “source” sequence against concrete or abstractly defined “target” sequences indicating the extent to which the sequences

match, or *partially* match, one another. Some techniques attempt to detect divergence between an abstract model representing the target sequence and the source sequence. Others attempt to detect divergence between a concrete target sequence produced, e.g., by an expert user, and a source sequence produced by some other user. Some produce diagnostic measures of distance to characterize the correspondence between target and source sequences. Others attempt to perform the best possible alignment of events in the target and source sequences and present the results to investigators in visual form. Still others use points of deviation between the target and input sequences to automatically indicate potential usability issues. In all cases, the purpose is to compare actual sequences of events against some model or trace of “ideal” sequences to identify potential usability issues.

1.2.6 Techniques for characterizing sequences

These techniques take “source” sequences as input and attempt to construct an abstract model to summarize, or characterize, interesting sequential features of those sequences. Some techniques compute probability matrices allowing process models with probabilities associated with transitions to be produced. Others construct grammatical models or finite state machines (FSM’s) to characterize the grammatical structure of events in the source sequences.

1.2.7 Visualization techniques

These techniques present the results of transformations and analyses in forms allowing humans to exploit their innate visual analysis capabilities to interpret results. These techniques can be particularly useful in linking results of analysis back to features of the interface.

1.2.8 Integrated evaluation support

Integrated support is provided by evaluation environments that facilitate flexible composition of various transformation, analysis, and visualization capabilities. Some environments also provide built-in support for managing domain-specific artifacts such as data regarding evaluations, subjects, tasks, and results of analysis.

1.3 Road Map

This subsection provides an overview of the sections to come. Section 2 provides important background and context and presents working definitions of such key terms as “usability”, “usability evaluation”, and “usability data”. Section 3 discusses the nature and characteristics of UI events and provides examples to illustrate some of the difficulties involved in extracting usability-related information from such events (herein lie some of the key insights that have arisen as a result of having performed the survey). Section 4 presents a comparison of approaches based on the framework outlined above. Section 5 discusses and summarizes the most important points of the survey and outlines implications on future research. Section 6 presents related work and Section 7 presents conclusions.

2 BACKGROUND

This section serves three purposes. First, it establishes working definitions of key terms such as “usability”, “usability evaluation”, and “usability data”. Second, it

situates observational usability evaluation within the broader context of HCI evaluation approaches, indicating some of the relative strengths and weaknesses of each. Finally, it isolates user interface events as one of the many types of data commonly collected in observational usability evaluation, indicating some of its strengths and weaknesses relative to other types. The definitions and frameworks presented here are not new and can be found in standard HCI texts (e.g. Preece et al. 1994 and Nielsen, 1993). Those well acquainted with usability, usability evaluation, and user interface event data, may wish to skip directly to Section 3 where the specific nature of user interface events and the reasons analysis is complicated are presented.

2.1 Usability, Usability Evaluation, and Usability Data

“*Usability*” is often thought of as referring to a single attribute of a system or device. However, it can more accurately be characterized as referring to a large number of related attributes. Nielsen provides the following definition (Nielsen, 1993):

Usability has multiple components and is traditionally associated these five usability attributes:

Learnability: The system should be easy to learn so that the user can rapidly start getting some work done with the system.

Efficiency: The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.

Memorability: The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.

Errors: The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.

Satisfaction: The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.

“*Usability evaluation*”, can be defined as the act of measuring (or identifying potential issues affecting) usability attributes of a system or device with respect to particular users, performing particular tasks, in particular contexts. The reason that users, tasks, and contexts are mentioned explicitly is that the values of usability attributes will vary depending on the background knowledge and experience of users, the tasks for which the system is used, and the context in which it is used.

“*Usability data*” is any information that is useful in measuring (or identifying potential issues affecting) the usability attributes of a system under evaluation.

It should be noted that the definition of usability cited above makes no mention of the particular purposes for which the system is designed or used. Thus, a system may be perfectly usable and yet not serve the purposes for which it was designed. Furthermore, a system may not serve any useful purpose at all (save for providing some form of subjective satisfaction) and still be regarded as perfectly

usable. Herein lies the distinction between usability and utility.

Usability and utility are regarded as subcategories of the more general term “usefulness” (Grudin, 1992). Utility is the question of whether the functionality of a system can in principle support the needs of users, while usability is the question of how satisfactorily users can make use of that functionality. Thus, system usefulness depends on both usability and utility.

While this distinction is theoretically clear, usability evaluations often identify both usability and utility issues, thus more properly addressing usefulness. However, to avoid introducing awkward terms such as “usefulness evaluation” or “usefulness data”. This survey will simply assume that usability evaluations and usability data can address questions of utility as well as questions of usability.¹

2.2 Types of Evaluation

This section contrasts the different types of approaches that have been brought to bear in evaluating usability in HCI.

First, a distinction is commonly drawn between formative and summative evaluation. *Formative* evaluation primarily seeks to provide feedback to designers to inform and evaluate design decisions. *Summative* evaluation primarily involves making judgements about “completed” products, to measure improvement over previous releases or to compare competing products. The techniques discussed in this survey can be applied in both sorts of cases.

Another important issue is the more specific motivation for evaluating. There are a number of practical motivations for evaluating. For instance, one may wish to gain insight into the behavior of a system and its users in actual usage situations in order to improve usability (formative) and to validate that usability has been improved (summative). One may also wish to gain further insight into users’ needs, desires, thought processes, and experiences (also formative and summative). One may wish to compare design alternatives, for example, to determine the most efficient interface layout or the best design representation for some set of domain concepts (formative). One may wish to compute usability metrics so that usability goals can be specified quantitatively and progress measured, or so that competing products can be compared (summative). Finally, one may wish to check for conformance to interface style guidelines and/or standards (summative). There are also academic motivations, such as the desire to discover features of human cognition that affect user performance and comprehension with regard to human-computer interfaces (formative).

There are a number of HCI evaluation approaches for achieving these goals which fall into three basic categories: predictive, observational, and participative.

1. These informal definitions provide a basis for later discussion. Alternative definitions that make finer or coarser distinctions might be substituted without significantly impacting what is to follow.

Reasons for evaluating	Predictive evaluation	Observational evaluation	Participative evaluation
Understanding user behavior & performance		X	x
Understanding user needs & experiences		x	X
Computing usability metrics	x	X	X
Comparing design alternatives	x	X	X
Certifying conformance w/ standards	X		

Table 1: Reasons for evaluating and types of evaluation.

Predictive evaluation usually involves making predictions about usability attributes based on psychological modeling techniques (e.g. the GOMS model or the Cognitive Walkthrough), or based on design reviews performed by experts equipped with general knowledge of HCI principles and past experience in design and evaluation. A key strength of predictive approaches is their ability to produce results based on non-functioning design artifacts without requiring the involvement of actual users.

Observational evaluation involves measuring usability attributes based on observations of users actually interacting with prototypes or fully functioning systems. Observational approaches can range from formal laboratory experiments to more less formal field studies. A key strength of observational techniques is that they tend to uncover aspects of actual user behavior and performance that are difficult to capture using other techniques.

Finally, *participative evaluation* involves collecting information regarding usability attributes directly from users based on their subjective reports. Methods for collecting such data range from questionnaires and interviews to more ethnographically inspired approaches involving joint observer/participant interpretation of behavior in context. A key benefit of participative techniques is their ability to capture aspects of users' needs, desires, thought processes, and experiences that are difficult to obtain otherwise.

In practice, actual evaluations often combine techniques from multiple approaches. However, the methods for posing questions and for collecting, analyzing, and interpreting data vary from one category to the next. Table 1 provides a high level summary of the relationship between types of evaluation and typical reasons for evaluating. An upper-case 'X' indicates a strong relationship. A lower-case 'x' indicates a weaker relationship. An empty cell indicates little or no relationship.

2.3 User Interface Events

Now that observational usability evaluation has been situated within the broader context of predictive, observational, and participative approaches, user interface events can be isolated as just one of many possible sources of observational data.

Sweeny and colleagues (Sweeny et al., 1993) identify a number of indicators that might be used to measure (or indicate potential issues affecting) usability attributes:

- *On-line behavior/performance*: e.g., task times, percentage of tasks completed, error rates, duration and frequency of on-line help usage, range of functions used.
- *Off-line behavior (non-verbal)*: e.g., eye movements, facial gestures, duration and frequency of off-line documentation usage, off-line problem solving activity.
- *Cognition/understanding*: e.g., verbal protocols, answers to comprehension questions, sorting task scores.
- *Attitude/opinion*: e.g., post-hoc comments, questionnaire and interview comments and ratings.
- *Stress/anxiety*: e.g., galvanic skin response (GSR), heart rate (ECG), event-related brain potentials (ERPs), electroencephalograms (EEG), ratings of anxiety.

Table 2 summarizes the relationship between these indicators and various techniques for collecting observational data. This table is by no means comprehensive, and is used only to indicate the rather specialized nature of user interface event data in observational evaluation. UI events provide excellent data for quantitatively characterizing on-line behavior, however, the usefulness of UI events in providing data regarding the remaining indicators has not been demonstrated. Some investigators, however, have used UI events to infer features

Usability Indicators	Video/Audio Recording	UI Event Recording	Post-hoc Comments	User Interview	Survey/ Questionnaire/ Test scores	Psychophysical recording
On-line behavior/performance	X	X				
Off-line behavior (non-verbal)	X					
Cognition/understanding	X		X	X	X	
Attitude/opinion			X	X	X	
Stress						X

Table 2: Usability indicators and techniques for collecting data.

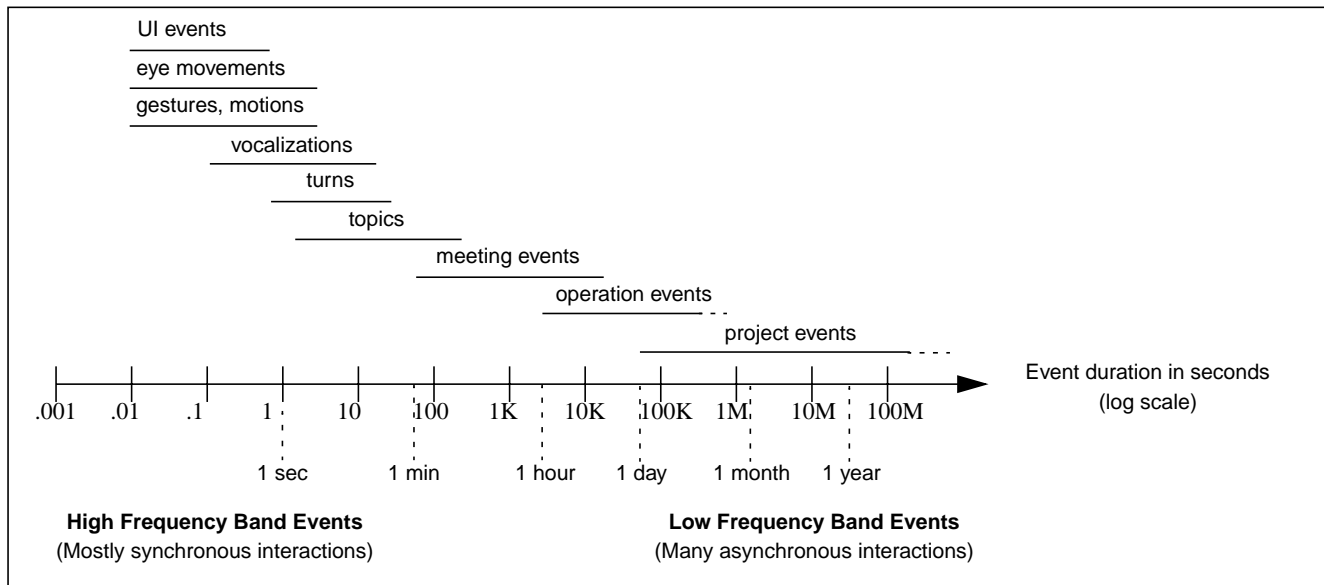


Figure 1. A spectrum of HCI events. Adapted from (Sanderson & Fisher, 1994).

of user knowledge and understanding (e.g. Kay & Thomas, 1995, Guzdial et al., 1993).

3 THE NATURE OF UI EVENTS

This section discusses the nature and characteristics of HCI events in general, and UI events specifically. The grammatical nature of UI events is discussed including some implications on analysis. The importance of contextual information in interpreting the significance of events is also discussed. Finally, a compositional model of UI events is presented to illustrate how these issues manifest themselves in UI event analysis. This model is used to ground later discussion and to highlight some of the strengths and weaknesses of surveyed approaches.

3.1 Spectrum of HCI Events

Before discussing the specific nature of UI events, this section introduces the broader spectrum of events that are of interest to researchers in HCI. Figure 1, adapted from (Sanderson & Fisher, 1994), indicates the durations of different types of events that are significant in research and design in HCI.

The horizontal axis is a log scale indicating event durations in seconds. It ranges from durations of less than one second to durations of years. The durations of UI events fall in the range of 10 milliseconds to approximately one second. The range of possible durations for each “type” of event is between one and two orders of magnitude, and the ranges of different types of events overlap one another.

If we assume that events occur serially, then the possible frequencies of events are constrained by the duration of those events. So, by analogy with the continuous domain (e.g. analog signals), each event type will have a characteristic frequency band associated with it (Sanderson & Fisher, 1994). Event types of shorter duration, e.g. UI events, can exhibit much higher frequencies when in sequence, and thus might be referred to as high-frequency

band event types. Likewise, event types of longer duration, such as project events, exhibit much lower frequencies when in sequence and thus might be referred to as low-frequency band event types. Evaluations that attempt to address the details of interface design have tended to focus on high-frequency band event types, whereas research on computer supported cooperative work (CSCW) has tended to focus on mid- to low-frequency band event types (Sanderson & Fisher, 1994).

Some important properties of HCI events that emerge from this characterization include the following:

1. *Synchronous vs. Asynchronous Events*: Sequences composed of high-frequency event types typically occur synchronously. For example, sequences of UI events, gestures, or conversational turns can usually be captured synchronously using a single recording. However, sequences composed of lower frequency event types, such as meeting or project events, may occur asynchronously, aided, for example, by electronic mail, collaborative applications, memos, and letters. This has important implications on the methods used to sample, capture, and analyze data, particularly at lower frequency bands (Sanderson & Fisher, 1994).
2. *Composition of Events*: Events within a given frequency band are often composed of events from higher frequency bands. These same events typically combine to form events at lower frequency bands. Sanderson and Fisher offer this example: a conversational turn is typically composed of vocalizations, gestures, and eye movements, and a sequence of conversational turns may combine to form a topic under discussion within a meeting (Sanderson & Fisher, 1994). See also (Hilbert et al., 1996) for a UI event-specific example.
3. *Inferences Across Frequency Band Boundaries*: Low frequency band events do not directly reveal their composition from higher frequency events. As a result, recording only low frequency events will typically

result in information loss. Likewise, high frequency events do not, in themselves, reveal how they combine to form events at lower frequency bands. As a result, either low frequency band events must be recorded in conjunction with high frequency band events, or there must be some external model (e.g. a grammar) to describe how high frequency events combine to form lower frequency events.

3.2 Grammatical Issues in Analysis

Grammars have been used in numerous disciplines to characterize the structure of data occurring in sequences. The main feature of grammars that make them useful in this context is their ability to define equivalence classes of patterns in terms of rewrite rules. For example, the following grammar (expressed as a set of rewrite rules) may be used to capture the ways in which a user can trigger a print job in a given application:

```
Print ->
  PrintToolBarButtonPressed OR
  (PrintDialogActivated THEN
  PrintDialogOkayed)

PrintDialogActivated ->
  PrintMenuItemSelected OR
  PrintAcceleratorKeyEntered
```

Rule 1 simply states that the user can trigger a print job by either pressing the print toolbar button (which triggers the job immediately) or by activating the print dialog and then pressing okay. Rule 2 specifies that the print dialog may be activated by either selecting the print menu item in the “File” menu, or by entering an accelerator key (e.g. “Ctrl-P”).

Let us assume that the lexical elements used to construct sentences in this language are:

- A: indicating “print toolbar button pressed”
- B: indicating “print menu item selected”
- C: indicating “print accelerator key entered”
- D: indicating “print dialog okayed”

Then the following “sentences” constructed from these elements are all semantically equivalent:

```
AAAA
CDAAA
ABDBDA
BDCDACD
CDBDCDBD
```

Each of the above sentences indicates a series of four print job activations. All occurrences of ‘A’ indicate an immediate print job activation while all occurrences of ‘BD’ or ‘CD’ indicate a print job activated by using the print dialog and then okaying it.

Notice that each of these sequences contains a different number of lexical elements. Some of them have absolutely no lexical elements in common (e.g. AAAA and CDBDCDBD). The lexical elements occupying the first and last positions differ from one sequence to the next. In short, there are a number of very salient differences between these sequences

at the lexical level. Techniques for automatically detecting sequences, comparing sequences, and characterizing sequences will likely have a lot to say about these sequences. However, unless the technique is grammar based, it will most likely not recognize the fact that the sequences are semantically equivalent, which they happen to be in this case.

3.3 Issues of Context in Interpretation

Another set of problems arises in attempting to interpret the significance of a UI event based only on the information carried by the event itself. To illustrate the problem more generally, consider the analogous problem of interpreting the significance of utterances in transcripts of natural language conversation. Important contextual cues are often spread across multiple utterances or may be missing from the transcript altogether.

Let us assume we have a transcript of a conversation that took place between persons A and B at a car show. The task is to identify A’s favorite cars based on utterances in the transcript.

Example 1: “The Lamborghini is one of my favorite cars”.

In this case, everything we need to know in order to determine one of A’s favorite cars is contained in a single utterance.

Example 2: “The Lamborghini”.

In this case we need access to prior context. ‘A’ is most likely responding to a question posed by ‘B’. Information carried in the question is critical in interpreting the response. For example, the question might have been: “Which car does your grandmother usually drive?” or “Which car here is your least favorite?”.

Example 3: “That is one of my favorite cars”.

In this case, we need the ability to de-reference an indexical. The information carried by the indexical “that” may not be available in any of the utterances in the transcript, but was clearly available to the interlocutors at the time of the utterance. Such contextual information was “there for the asking”, so to speak, and might have been noted had the transcriber chosen to do so at the time of the utterance.

Example 4: “That is another one.”

In this case we would need access to both prior context *and* the ability to de-reference an indexical.

The following examples illustrate analogous situations in the interpretation of UI events:

Example 1’: `PrintToolBarButtonPressed`

This event carries with it enough information to indicate the action the user has taken.

Example 2’: `CancelButtonPressed`

This event does not on its own indicate what was canceled. As in Example 2 above, this event indicates a response to

some prior event, for example, a prior `PrintMenuItemSelected` event.

Example 3': `ErrorDialogActivated`

The information needed to interpret the significance of this event *may* be available in prior events, but a more direct way to interpret its significance would be to query the dialog for its error message. This is similar to de-referencing an indexical, if we think of the error dialog as figuratively “pointing at” an error message that does not actually appear in the event stream.

Example 4': `ErrorDialogActivated`

Assuming the error message is “Illegal Command”, then the information needed to interpret the significance of this event is not only found by de-referencing the indexical (the error message “pointed at” by the dialog) but must be supplemented by information available in prior events. It may also be desirable to query contextual information stored in user interface components to determine the combination of parameters (specified in a dialog, for example) that led to this particular error.

The basic insight here is that sometimes an utterance — or a UI event — may not carry enough information on its own to allow its significance to be properly interpreted. Sometimes critical contextual information is available elsewhere in the transcript, and sometimes that information is not available in the transcript, but *was* available, “for the asking”, at the time of the utterance, or event, but not afterwards. These issues figure prominently in the problem of extracting meaningful information from UI events.

3.4 The Composition of UI Events

Figure 2 illustrates a compositional model of UI events. At the lowest level are physical events, for example, fingers depressing keys or a hand moving a pointing device such as a mouse. Input device events, such as key and mouse interrupts, are generated by hardware in response to physical events. UI events associate input device events with windows and other interface objects on the screen. Events at this level include button presses, list and menu selections, focus events in input fields, and window movements and resizing.

Abstract interaction events are not directly generated by the window system, but may be computed based on UI events and other contextual information such as UI state. Abstract interaction events are indicated by recurring, idiomatic patterns of UI events. For example, an abstract interaction event might be associated with the act of providing a value to an application by manipulating user interface components. In the case of an input field, this would mean that the field had been edited, was no longer being edited, and now contains data. The patterns of UI events that indicate an abstract interaction event such as “`ValueProvided`” will differ from one type of interface component to another, and from one application to another, but will remain fairly stable within a given application.

Domain/task-related and goal/problem-related events are at the highest levels. Unlike other levels, these events indicate

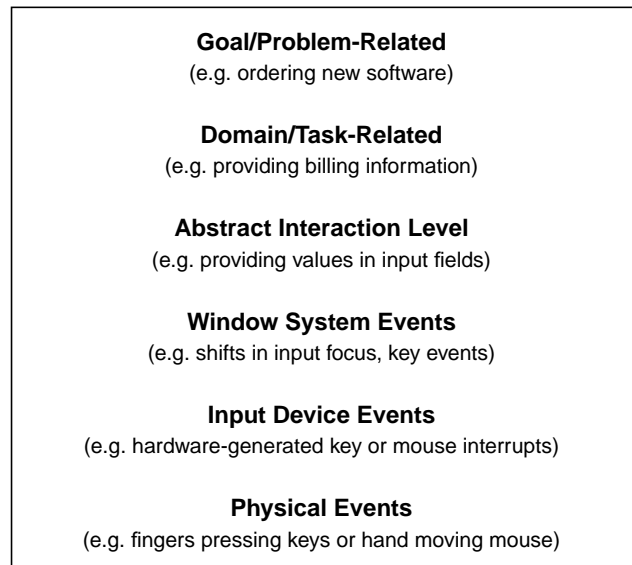


Figure 2. The composition of UI events.

progress in the user’s tasks and goals. Inferring these events based on UI events can be straightforward when the user interface provides explicit support for structuring tasks or indicating goals. For example, Wizards in Microsoft Word™ lead users through a sequence of steps in a predefined task. The user’s progress can be recognized in terms of UI events such as button presses on the “Next” button. In other cases, task and goal related events may be more difficult to infer. For example, the goal of placing an order includes the task of providing customer address information. The task-related event “`AddressProvided`” may then be recognized in terms of “`ValueProvided`” abstract interaction events occurring within each of the fields in the contact information section of the form. Finally, in some cases, it may be impossible to infer events at these higher levels based only on UI events.

4 COMPARISON OF APPROACHES

This section introduces the approaches that have been applied to the problem of extracting usability-related information from UI events. The matrix presented in Table 3 summarizes the features of each approach and classifies approaches into categories. The following sections discuss the features that distinguish each class of approaches and provide examples of some of the approaches in each class. Related work is discussed where appropriate. Strengths and limitations of each class of approaches are also discussed.

4.1 Synch and Search

4.1.1 Purpose

User interface events provide detailed information regarding user behavior that can be easily captured, searched, counted, and analyzed using automated tools. However, higher level events of interest can be difficult to infer from user interface events, and sometimes critical contextual information is simply missing from the event stream, making proper interpretation challenging at best.

Table 3: A classification of computer-aided techniques for extracting usability-related information from user interface events

Techniques (sorted by class)

	Tool/Technique	Reference	Synch/Search	Event Capture	Use of Context	Filter/Recode	Abstract/Recode	Counts & Stats	Sequence Detect	Sequence Compare	Sequence Character	Visualize	Data Manage	UI Platform
Synch & Search	Playback	(Neal & Simmons, 1983)	Obs.	X				X						?
	Microsoft Lab	(Hoiem & Sullivan, 1994)	Obs.+Vid.	X		X	?							MS Windows
	SunSoft Lab	(Weiler, 1993)	Obs.+Vid.	X		X	?							X Windows
	Apple Lab	(Weiler, 1993)	Obs.+Vid.	X		X	?							MacOS
	I-Observe	(Badre et al., 1995)	Vid.	X					Model			X		X Windows
Transform	Incident Monitoring	(Chen, 1990)		X	UI+App	X								X Windows
	Remote "CIs"	(Hartson et al., 1996)		X	UI+User	User	User							?
	CHIME	(Badre & Santos, 1991)		X	UI	Model	Model							X Windows
	EDEM	(Hilbert & Redmiles, 1997)		X	UI+User	Model+User	Model+User		Model			X		Java AWT
Counts & Stats	UIMS	(Buxton et al., 1983)		X	UI+App			X				X		UIMS
	MIKE	(Olsen & Halversen, 1988)		X	UI+App			X						UIMS
	KRI/AG	(Lowgren & Nordqvist, 1992)		X	UI+App			X						UIMS
	Long-Term Monitoring	(Kay & Thomas, 1995)		API	App			Programs				Plotting		API
	AUS	(Chang & Dillon, 1997)		X	UI			X				X		MS Windows
	Aqueduct Profiler	(Aqueduct Software, 1998)		API	App			Database				Plotting	X	API
Sequence Detect	Fisher's Cycles	(Fisher, 1988)							X					
	LSA	(Sackett, 1978)							X					
	MRP	(Siochi & Hix, 1991)							X					
	TOP/G	(Hoppe, 1988)		Simulated	UI				Model					Simulation
	Expectation Agents	(Girgensohn et al., 1994)		X	UI+User	User	User		Model					OS/2
	EDEM	(Hilbert & Redmiles, 1997)		X	UI+User	Model+User	Model+User		Model					Java AWT
	TSL	(Rosenblum, 1991)		X	App				Model					N/A
	Amadeus	(Selby et al., 1991)		X	?				Model					N/A
	YEAST	(Krishnamurthy & Rosenblum, 1995)		X	?		Model		Model					N/A
	EBBA	(Bates, 1995)		X	?		Model		Model					N/A
GEM	(Mansouri-Samani & Sloman, 1997)		X	App	Model	Model		Model					N/A	
Sequence Compare	ADAM	(Finlay & Harrison, 1990)								Seq+Seq				
	UsAGE	(Ueling & Wolf, 1995)		?	?					Seq+Seq		X		X Windows
	EMA	(Balbo, 1996)		API	App					Seq+Model				API
	Process Validation	(Cook & Wolf, 1997)								Seq+Model				N/A
Sequence Character	Markov-based	(Guzdial, 1993)					Model				Manual			
	Grammer-based	(Olson et al., 1994)					Model				Manual			N/A
	Process Discovery	(Cook & Wolf, 1995)									Auto			N/A
Integrated Support	Hawk	(Guzdial, 1993)				AWK	AWK	AWK	AWK				X	
	DRUM	(Macleod & Rengger, 1993)	Obs.+Vid.	X				X					X	MacOS
	MacSHAPA	(Sanderson et al., 1994)	Obs.+Vid.			Database	Database	X	X	Seq+Seq	?	X	X	

Column	Key to Column Values
Synch/Search	(Obs) = events synchronized w/ coded observations; (Vid) = events synchronized w/ video.
Event Capture	(X) = events captured automatically; (API) = application must report events; (Simulated) = events input to a command line simulation.
Use of Context	(UI) = contextual info from the UI is used; (App) = contextual info from the application is used; (User) = the user provides contextual info.
Filter/Recode - Sequence Detect	(X) = built-in filtering/abstraction/counts & stats/sequence detection; (User) = user filters/abstracts events; (Model) = abstract model used to filter, abstract, or detect sequences; (AWK) = AWK programming language used; (Database) = database query & manipulation language used.
Sequence Compare	(Seq+Seq) = two sequences compared; (Seq+Model) = sequence compared against abstract model representing legal/expected sequences.
Sequence Character	(Manual) = statistical/grammatical techniques used to construct abstract model manually; (Auto) = abstract model generated automatically.
Visualize	(X) = built-in visualizations; (Plotting) = use of standard database/spreadsheet plotting facilities.
Data Manage	(X) = built-in domain-specific management of data and evaluation files.

Synch and Search techniques seek to combine the advantages of UI event data with the advantages provided by more semantically rich observational data, such as experimenters' observations or video recordings.

By synchronizing UI events with other sources of data such as coded observations and/or video, searches in one medium can be used to locate supplementary information in others. Therefore, if an investigator wishes to review all segments of a video in which a user uses the help system or invokes a particular command, it is not necessary to manually search the entire recording. The investigator can: (a) search through the log of UI events for particular events of interest and use the timestamps associated with those events to automatically cue up the video recording, or (b) search through a log of observations (that were entered by the investigator either during or after the time of the recording) and use the timestamps associated with those observations to cue up the video. Similarly, segments of interest in the video can be used to locate the detailed user interface events associated with those episodes.

4.1.2 Examples

Playback (Neal & Simmons, 1983) is an early example of a system employing synch and search capabilities. In Playback, UI events are collected automatically and experimenters enter observation codes and comments, during or after the evaluation session, which are automatically time-synchronized with the captured events. Instead of using video, Playback allows recorded events to be played back through the application interface to re-trace the user's actions. The evaluator can step through the playback based on events or coded observations as if using an interactive debugger. A handful of simple analyses are built-in that automatically calculate counts and summary statistics. This technique captures less information than video-based techniques since video can also be used to record off-line behavior such as facial gestures, off-line documentation use, and verbalizations. Also, there can be problems associated with replaying user sessions accurately in applications where behavior can be affected by events other than user interface events, such as database failures, for example.

DRUM, the Diagnostic Recorder for Usability Measurement, is an integrated evaluation environment that supports video-based usability evaluation (Macleod et al. 1993). DRUM was developed at the National Physical Laboratory (NPL) as part of the ESPRIT Metrics for Usability Standards in Computing (MUSiC) Project. DRUM features a module for recording and synchronizing events, observations, and video (Recording Logger), a module for defining and managing observation coding schemes (Scheme Manager), a module for calculating pre-defined counts and summary statistics (Log Processor), and a module for managing and manipulating evaluation-related information regarding subjects, tasks, recording plans, logs, videos, and results of analysis (Evaluation Manager).

Usability specialists at Microsoft, Apple, and SunSoft all report the use of tools that provide synch and search capabilities (Weiler et al., 1993, Hoiem & Sullivan, 1994).

The tools used at Microsoft include a tool for logging observations (Observer), a tool for tracking UI events (Tracker), and a tool for synchronizing and reviewing data from the multiple sources (Reviewer). The tools used at Apple and SunSoft are essentially similar. All tools support some level of event filtering as part of the capture process. Apple's filtering appears to be user-definable while Microsoft and SunSoft's filtering appear to be programmed into the capture tools. Scripts and general purpose analysis programs, such as Excel, are used to perform counts and summary statistics after capture. All tools support video annotations to produce "highlights" videos. Microsoft's tools provide an API to allow applications to report application-specific events, or events not readily available in the event stream.

I-Observe, the Interface OBServation, Evaluation, Recording, and Visualization Environment developed at Georgia Tech (Badre, et al. 1995) also provides synch and search capabilities. I-Observe is a set of loosely integrated tools for collecting, selecting, analyzing, and visualizing event data. Searches are performed by specifying predicates over the fields contained within a single event record. Patterns of events can then be located by stringing together a set of such search specifications into a regular expression. The intervals (identified by begin and end events) matched by the regular expressions can be used to automatically select data for visualization, or to drive the display of corresponding segments of the video tape.

4.1.3 Strengths

The strengths of these techniques lie in their ability to integrate data sources with complementary strengths and weaknesses, and to allow searches in one medium to locate related information in the others. UI events provide detailed performance information that can be searched, counted, and analyzed using automated techniques, however, UI events often leave out higher level contextual information that can more easily be captured using coded observations and/or video recordings.

4.1.4 Limitations

Techniques relying on synchronizing UI events with video and coded observations require the use of video recording equipment and the presence of observers. Video recording equipment is not always available, can be expensive to purchase, and usually implies the presence of an operator with expertise in recording. The use of video equipment and the presence of observers can make subjects self-conscious and affect performance, may not be allowed in certain circumstances, and tends to produce massive amounts of data that can be expensive to store and analyze. These issues can all be serious limiting factors on evaluation size, location, and duration.

4.2 Transformation

4.2.1 Purpose

These techniques combine filtering, abstraction, and recoding to transform event streams for various purposes, such as facilitating human pattern detection, comparison,

and characterization, or to prepare data for input into automatic techniques for performing these functions.

Filtering essentially operates by subtracting information, allowing events and sequences of interest to emerge from the “noise”. Filtering involves selecting a subset of events by specifying constraints on event attributes. For instance, one may elect to filter out all events associated with mouse movements in order to focus analysis on higher level actions such as button presses and menu selections. One might also wish to remove extraneous information from events that unnecessarily distinguish them from one another. For example, one might elect to remove the pointer coordinate values associated with button press events if this information unnecessarily distinguishes button presses within the same button.

Abstraction operates essentially by synthesizing new events based on information in the event stream. For example, a pattern of events indicating that an input field had been edited, that a value had been provided, and that the user’s editing attention had since shifted to another component might indicate the abstract event “VALUE_PROVIDED”, which is not signified by any single event in the event stream. The same abstract event might be indicated by different patterns of events in different UI components, e.g., a selection event occurring in an option menu. Also, if the window in which a button is located cannot be determined based on the attributes of button press events, then abstraction will be required to produce such abstract events as “PrintDialogCanceled” or “PrintDialogOkayed”.

Recoding involves producing new event streams based on filtering and/or abstraction. This allows the same manual and/or automated analysis techniques normally applied to raw event streams to be applied to filtered and abstracted event streams, potentially leading to different results. For instance, consider the example in Section 3. If the sequences representing four sequential print job activations were embedded within the context of a larger sequence (or set of sequences), they might not be identified as being similar subsequences, particularly by automated techniques such as those presented below. However, after performing abstraction based on the grammar in that example, each of these sequences could be recoded as “AAAA”, making them much more likely to be identified as common subsequences by automatic techniques.

4.2.2 Examples

Chen (Chen, 1990) presents an approach to user interface event monitoring that filters events based on the notion of “incidents”. Incidents are defined as only those events that actually trigger some response from the application, and not just the user interface system. This technique was demonstrated by modifying the Xt Intrinsics Toolkit to report data regarding events triggering callback procedures registered by applications. This allows events not handled by the application to be ignored altogether. Investigators may further filter event reporting by selectively specifying incidents of interest to be reported. Widgets in the user interface toolkit were modified to provide a query

procedure to return limited contextual information when an event associated with a given widget triggers a callback.

Hartson and colleagues (Hartson, et al., 1996) report an approach to remote usability evaluation that relies on the user to filter events. Users identify potential usability problems that arise during the course of interacting with an application, and report information regarding these “critical incidents” by pressing a “report” button that is supplied in the interface. Traces of the UI events leading up to and following the incident are reported via email along with contextual information that is provided by the user. In this case, filtering is achieved by only reporting the n events leading up to, and m events following, user-identified critical incidents (where n and m are parameters that can be set in advance by investigators).

CHIME (Badre & Santos, 1991a) is a computer-human interaction monitoring engine developed at Georgia Tech that is similar, in some ways, to Chen’s approach. CHIME allows investigators to specify ahead of time which events to report and which events to filter. An important difference is that CHIME supports a limited notion of abstraction that allows a level of indirection to be built on top of the window system. The basic idea is that abstract “interaction units” (IUs) are defined that translate window system events into platform independent events upon which further monitoring infrastructure is built. The events to be recorded are then specified in terms of these platform independent IUs.¹

Hawk (Guzdial 1993) is an environment for filtering, abstracting, and recoding log files of UI events. Hawk’s main functionality is provided by a variant of the AWK programming language (Aho et al. 1988), and an environment for managing data files is provided by HyperCard. Events appear in the event log one per line, and AWK pattern-action pairs are used to specify what is to be matched in each line of input (the pattern) and what is to be printed as output (the action). This allows fairly flexible filtering, abstraction, and recoding to be performed.

MacSHAPA (Sanderson et al. 1994), which is discussed further below, supports filtering and recoding in the form of a database query and manipulation language which allows event records to be selected based on attributes, and new streams defined based on the results of queries. Abstraction can be performed manually by entering new records representing abstract events and associating them with existing event records.

EDEM, expectation-driven event monitoring (Hilbert & Redmiles, 1998) is a system for capturing UI events developed at UC Irvine that supports both filtering and abstraction. Filtering is achieved by specifying ahead of time which events to report as well as by a critical incident

1. The paper also alludes to the possibility of allowing higher level IUs to be hierarchically defined in terms of lower level IUs (using a context-free grammar and pre-conditions) to provide a richer notion of abstraction. However, this appears to not have been implemented (Badre & Santos, 1991a & b).

reporting mechanism akin to that reported in (Hartson, et al., 1996). One important difference, however, is that EDEM provides facilities to allow evaluators to define automated agents to help in the detection of “critical incidents”, thereby lifting some of the burden from users who often do not know when their actions are violating expectations about proper usage (Smilowitz et al., 1994). Furthermore, abstraction and recoding is achieved by allowing new events to be defined in terms of patterns of existing events which may then be used in further hierarchical event processing.

4.2.3 Strengths

The main strength of these approaches lies in their explicit support for filtering, abstraction, and recoding which are essential steps in preparing UI events for most types of analysis. Chen and CHIME address issues of filtering prior to reporting. Hartson and colleagues add to this a technique for accessing contextual information via the user. EDEM adds to these automatic detection of critical incidents and abstraction that is performed in context. All these techniques might potentially be used to collect UI events remotely.

Hawk and MacSHAPA, on the other hand, do not address event collection but provide powerful and flexible environments for transforming and analyzing UI events.

4.2.4 Limitations

The techniques that filter, abstract, and recode events while collecting them run the risk of dropping data that might have been useful in analysis. Techniques that rely exclusively on users to filter events are even more likely to drop useful information. Most of the approaches that support flexible abstraction and recoding do not do so during capture, meaning contextual information that can be critical in abstraction is not available.

4.3 Counts and Summary Statistics

4.3.1 Purpose

As noted above, one of the key benefits of UI events is how readily details regarding on-line behavior can be captured and manipulated using automated techniques. Most investigators rely on general purpose analysis programs such as spreadsheets and statistical packages to compute counts and summary statistics based on collected data (e.g. feature use counts or error frequencies). However, some investigators have proposed specialized facilities for performing and reporting such calculations. This section provides examples of some of the systems boasting built-in functions for calculating usability-related metrics.

4.3.2 Examples

The MIKE user interface management system is an early example of a system offering built-in facilities for calculating and reporting metrics (Olsen & Halversen 1988). Because MIKE controls all aspects of input and output activity, and because it has an abstract description of the user interface and application “commands”, MIKE is in a uniquely good position to monitor UI events and associate them with responsible interface components and application commands. Example metrics include:

- Performance time: How much time is spent completing tasks such as specifying arguments for commands?
- Mouse travel: Is the sum of the distances between mouse clicks unnecessarily high?
- Command frequency: Which commands are used most frequently or not at all?
- Command pair frequency: Which commands are used together frequently? Can they be combined or placed closer to one another?
- Cancel and undo: Which dialogs are frequently canceled? Which commands are frequently undone?
- Physical device swapping: Is the user switching back and forth between keyboard and mouse unnecessarily? Which features are associated with high physical swapping counts?

MIKE logs all UI events and associates them with the interface components triggering them and the application commands triggered by them. Event logs are written to files that are later read by a metric collection and report generation program. This program uses the abstract description of the interface to interpret the log and to generate human readable reports summarizing the metrics.

MacSHAPA (Sanderson et al. 1994) also includes numerous built-in features to support computation and reporting of counts and summary statistics.

Finally, Automatic Usability Software (AUS) (Chang & Dillon 1997) is reported to provide a number of automatically computed metrics such as help system use, use of cancel and undo, mouse travel, and mouse clicks per window.

4.3.3 Related

Aqueduct Profiler (Aqueduct Software, 1997) is a system for computing counts and summary statistics to characterize the usage of applications in beta tests. It has an API that applications use to report the occurrence of events of interest such as feature usage. This information is then sent via email to developers’ computers where it is stored in a database and plotted using standard database plotting facilities.

4.3.4 Strengths

With the number of possible metrics, counts, and summary statistics that might be computed and that might be useful in usability evaluation, it is nice that some systems provide built-in facilities to perform and report such metrics automatically.

4.3.5 Limitations

With the exception of MacSHAPA, the systems described above do not provide facilities to allow evaluators to modify built-in counts, statistics, and reports or to add new ones of their own. Also, the computation of useful metrics is greatly simplified when the system computing the metrics has a model of the user interface and application commands, as in the case of MIKE, or when the application code is instrumented to report the events to be analyzed, as in the case of Aqueduct Profiler. AUS does not address

application-specific features and thus is limited in its ability to relate metrics results with application features.

4.4 Sequence Detection

4.4.1 Purpose

These techniques are used to detect occurrences of concrete or abstractly defined “target” sequences within “source” sequences. In some cases these target sequences are quite abstractly defined, and are supplied by the developers of the technique (e.g. Fisher’s cycles, lag sequential analysis, multiple repeating pattern analysis, and automatic chunk detection). In other cases, target sequences may be more specific to a particular application and are supplied by the users of the technique (e.g. TOP/G, expectation agents, and expectation-driven event monitoring). Sometimes the purpose is to generate a list of matched source subsequences for perusal by the investigator (e.g. Fisher’s cycles, maximal repeating pattern analysis, and automatic chunk detection). Other times the purpose is to recognize sequences of UI events that violate particular expectations about proper UI usage (e.g. TOP/G, expectation agents, and expectation-driven event monitoring). Finally, in some cases the purpose may be to perform abstraction and recoding of the source sequence based on matches of the target sequence (e.g. expectation-driven event monitoring).

4.4.2 Examples

TOP/G (Hoppe 1988) is a task-oriented parser/generator that parses sequences of commands from a command-line simulation and “infers” the higher level tasks that are being performed. Expected tasks are modeled in a notation based on Payne and Green’s task-action grammars (Payne & Green, 1986) and are represented in a Prolog database as rewrite/production rules. Composite tasks are defined hierarchically in terms of elementary tasks which are in turn decomposed into “triggering rules” that map keystroke level events into elementary tasks. Rules may also be defined to recognize “suboptimal” user behaviors that might be abbreviated by simpler compositions of commands. A later version attempted to use information about the side effects of commands in the environment to recognize when a longer sequence of commands might be replaced by a shorter sequence. In both cases, the generator functionality of TOP/G could be used to output the shorter, or more “optimal”, command sequence.

A number of techniques for detecting abstractly defined patterns in sequential data have been applied by researchers involved in exploratory sequential data analysis (ESDA). For an in-depth treatment see (Sanderson & Fisher, 1994). These techniques can be subdivided into two basic categories:

Techniques sensitive to sequentially separated patterns of events, e.g.:

- Analysis of cycles
- Lag sequential analysis (LSA)

Techniques sensitive to strict transitions between events, e.g.:

- Maximal Repeating Patterns
- Log linear analysis
- Time-series analysis

Fisher’s cycles (Fisher, 1991) allow investigators to specify beginning and ending events of interest that are then used to automatically identify all occurrences of subsequences beginning and ending with those events (excluding those with further internal occurrences of those events). For example, assume an investigator is faced with a source sequence of events encoded using the letters of the alphabet, such as: ABACDACDBADBCACCCD. Suppose further that the investigator wishes to find out what happened between all occurrences of ‘A’ (as a starting point) and ‘D’ (as an ending point), Fisher’s cycles produces the following analysis:

```
Source sequence:      ABACDACDBADBCACCCD
Begin event:         A
End event:           D
Output:
  ABACDACDBADBCACCCD
  ABACDADBADBCACCCD
  ABACDACDBADBCACCCD
  ABACDACDBADBCACCCD
```

Cycle #	Frequency	Cycle
1	2	ACD
2	1	AD
3	1	ACCCD

The investigator could then note that there were clearly no occurrences of B in any A->D cycle. Furthermore, the investigator might use a grammatical technique to recode repetitions of the same event into a single event, thereby revealing that the last cycle (ACCCD) is essentially equivalent to the first two (ACD). This is one way of discovering similar subsequences in “noisy” data.

Lag sequential analysis (LSA) is another popular technique that identifies the frequently with which two events occur at various “removes” from one another (Sackett, 1978, Allison & Liker, 1982, Faraone & Dorfman, 1987, Sanderson & Fisher, 1994). LSA takes one event as a ‘key’ and another event as a ‘target’ and reports how often the target event occurs at various intervals before and after the key event. If ‘A’ were the key and ‘D’ the target in the previous example, LSA would produce the following analysis:

```
Source sequence:      ABACDACDBADBCACCCD
Key event:           A
Target event:        D
Lag(s):              -4 through +4
Output:
```

Lag	-4	-3	-2	-1	1	2	3	4
Occurrences	0	1	1	1	1	2	0	1

The count of 2 at Lag = +2 corresponds to the ACD cycles identified by Fischer's cycles above. Assuming the same recoding operation performed above to collapse multiple occurrences of the same event into a single event, this count would increase to 3. The purpose of LSA is to identify correlations between events (that might be causally related to one another) that might otherwise have been missed by techniques more sensitive to the strict transitions between events.

An example of a technique that is more sensitive to strict transitions is the Maximal Repeating Pattern (MRP) analysis technique (Siochi & Hix, 1991). MRP operates under the assumption that repetition of user actions can be an important indicator of potential usability problems. MRP identifies all patterns occurring repeatedly in the input sequence and produces a listing of those patterns sorted by length first followed by frequency of occurrence in the source sequence. MRP applied to the sequence above would produce the following analysis:

Source Sequence: ABACDACDBADBCACCCD

Output:

Pattern #	Frequency	Pattern
1	2	ACD
2	3	AC
3	3	CD
4	2	BA
5	2	DB

MRP is similar in spirit to Fisher's cycles and LSA, however, the investigator does not specify particular events of interest. Notice that the ACCCD subsequence identified in the previous examples is not identified by MRP.

Markov-based techniques can be used to compute the transition probabilities from one or more events to the next event. Statistical tests can be applied to determine whether the probabilities of these transitions is greater than would be expected by chance (Sanderson & Fisher, 1994). Other related techniques include log linear analysis (Gottman & Roy, 1990) and formal time-series analysis (Box & Jenkins, 1976). All of these techniques attempt to find strict sequential patterns in the data that occur more frequently than would be expected by chance.

Santos and colleagues have proposed an algorithm for detecting users' "mental chunks" based on pauses and flurries of activity in human computer interaction logs (Santos, et al. 1994). The algorithm is based on an extension of Fitts' law (Fitts, 1964) that predicts the expected time between events generated by a user who is not engaged in problem solving activity. For each event transition in the log, if the pause in interaction cannot be justified by the predictive model, then the lag is assumed to signify a transition from "plan execution phase" to "plan acquisition phase" (Santos, et al. 1994). The results of the algorithm are used to segment the source sequence into plan execution chunks and chunks most probably associated with problem solving and planning activity. The

assumption is that expert users tend to have longer, more regular execution chunks than novice users, so user expertise might be inferred on the basis of the results of this chunking algorithm.

Finally, work done by Redmiles and colleagues on expectation agents (EAs) (Girgensohn, et al. 1994) and expectation-driven event monitoring (EDEM) (Hilbert & Redmiles, 1998) use sequence detection techniques to trigger reactions to particular patterns of UI events. These approaches employ an event pattern language to specify ordering constraints on events. When a pattern of interest is detected, contextual information may be queried before action is taken. Possible actions include notifying the user and/or investigator that a particular pattern was detected, collecting user feedback, and reporting interface state and events leading up to violations. In the case of EDEM, the detection of a particular pattern may also result in abstraction and recoding of the event stream to indicate the occurrence of an abstract event.

4.4.3 Related

EBBA (Bates 1995) is a distributed debugging system that attempts to match the behavior of a distributed program against partial models of expected behavior. EBBA is similar to EDEM, particularly in its ability to abstract and recode the event stream based on hierarchically defined abstract events.¹

Amadeus (Selby 1991) and YEAST (Rosenblum 1995) are event-action systems used to detect and take actions based on patterns of events in software processes that are also similar in spirit to expectation agents and EDEM. Techniques that have been used to specify behavior of concurrent systems, such as the Task Sequencing Language (TSL) as described in (Rosenblum, 1991) are also related. Some of the same techniques used for specifying and detecting patterns of events in these approaches may be usefully applied to the problem of specifying and detecting patterns of UI events.

4.4.4 Strengths

The strength of these approaches lies in their ability to help investigators detect patterns of interest in events, and not just perform analysis on isolated occurrences of events. The techniques associated with ESDA allow patterns that may not have been anticipated to be discovered. Languages for detecting patterns of interest in UI events based, for example, on extended regular expressions (Sanderson & Fisher, 1994), or on more grammatically inspired techniques (e.g. Hilbert & Redmiles, 1998) can be used to locate patterns of interest and to transform event streams by recoding abstract patterns of events into "abstract" events.

¹EBBA is sometimes characterized as a sequence comparison system since the information carried in a partially matched model can be used to help the investigator better understand where the program's behavior has gone wrong (or where a model is inaccurate). However, EBBA does not directly indicate that partial matches have occurred or provide any diagnostic measures of correspondence. Rather, the user must notice that a full match has failed, and then manually inspect the state of the pattern matching mechanism to see which events were matched and which were not.

4.4.5 Limitations

The ESDA techniques described above tend to produce large amounts of output that can be difficult to interpret, and that frequently do not lead to identification of usability problems (D.L. Cuomo, 1994). The non-ESDA techniques require investigators to know how to specify the patterns they are looking for, and to define them (sometimes painstakingly) before analysis can be performed.

4.5 Sequence Comparison

4.5.1 Purpose

These techniques compare a “source” sequence against concrete or abstractly defined “target” sequences indicating *partial* matches between the two. Some techniques attempt to detect divergence between an abstract model of the target sequence and a source sequence (e.g. EMA). Others attempt to detect divergence between a concrete target sequence produced, e.g., by an expert user and a source sequence produced by some other user (e.g. ADAM and UsAGE). Some produce diagnostic measures of distance to characterize the correspondence between target and source sequences (e.g. ADAM). Others attempt to perform the best possible alignment of events in the target and source sequences and present the results visually (e.g. UsAGE and MacSHAPA). Still others use points of deviation between the target and input sequences to automatically indicate potential “critical incidents” (e.g. EMA). In all cases, the purpose is to compare actual usage against some model or trace of “ideal” or expected usage to identify potential usability problems.¹

4.5.2 Examples

ADAM (Finlay & Harrison, 1990), is an advanced distributed associate memory that compares fixed length source sequences against a set of target sequences that were used to “train” the memory. Target sequences used to train the memory are associated with “classes” of patterns by the investigator. When a source sequence is input, the associative memory identifies the class that most closely matches the source sequence, and two diagnostic measures are output: a “confidence” measure that is 100% only when the source sequence is identical to one of the trained target sequences, and a “distance” measure, indicating how far the source pattern is from the next “closest” class. Investigators then use these measure to determine whether a source sequence is different enough from the trained sequences to be judged as a possible “critical incident”. ADAM might also be trained on examples of “expected” critical incidents so that these might be detected directly.

MacSHAPA (Sanderson & Fisher, 1994) provides techniques for aligning two sequences of events as optimally as possible based on maximal common

subsequences (Hirschberg, 1975). The results are presented visually as cells in adjacent spreadsheet columns with aligned events appearing in the same row, and missing cells indicating events in one sequence that could not be aligned with events in the other.

A related technique is applied in UsAGE (Ueling & Wolf, 1995) where a source sequence of UI events (related to performance of some task) is aligned as optimally as possible with a target sequence (produced by an “expert” performing the same task), and presented in visual form.

Finally, EMA, an automatic analysis mechanism for the evaluation of user interfaces (Balbo, 1996), requires that investigators provide a grammar-based model describing all the expected paths through a particular user interface. An evaluation program is then used to compare a log of UI events generated by use of the interface against the model, indicating in the log and the model where the user has taken “illegal” paths. Other simple patterns, e.g., the use of cancel, are also detected and reported. This information can then be used by the evaluator to identify problems in the interface (or problems in the model).

4.5.3 Related

Process validation techniques (Cook & Wolf, 1997) are related in that they compare actual traces of events generated by a software process against an abstract model of the intended process. A diagnostic measure of distance is computed to indicate the correspondence between the trace and the closest acceptable trace produced by the model. Techniques for performing error correcting parsing are also related. See (Cook & Wolf, 1997) for more related techniques.

4.5.4 Strengths

The strengths of these approaches lie their ability to compare actual traces of events against expected traces, or models of expected traces, in order to identify potential usability issues. This is particularly appealing when expected traces can be specified “by demonstration” as in the case of ADAM and UsAGE.

4.5.5 Limitations

Unfortunately, all of these techniques have significant limitations.

A key limitation of any approach that compares whole sequences is the underlying (unrealistic) assumption that: (a) sequences can be easily segmented for piecemeal comparison, as in the case of ADAM, or (b) that whole interaction sequences will actually exhibit reasonable correspondence, as in the case of UsAGE.

Furthermore, the output of all these techniques, (except in the case of perfect matches) requires expert human interpretation to determine whether the sequences are interestingly similar or different. This is one reason why completely matching patterns that directly indicate violations of expected patterns (e.g. as in EDEM) can be superior to techniques that simply produce output to the effect, “this sequence is similar to a sequence you specified, but different; the measure of correspondence is 61%”.

1. TOP/G, expectations agents, and EDEM (discussed above) are also intended to detect deviations between actual and expected usage to identify potential usability problems. However, these approaches are better characterized as detecting complete matches between source sequences and (“negatively defined”) target patterns that indicate unexpected, or suboptimal behavior, as opposed to partially matching, or comparing, source sequences against (“positively defined”) target patterns.

A key limitation of any technique comparing sequences against abstract models (e.g. EMA and the process validation techniques described by Cook and Wolf) is that in order to reliably categorize a source sequence as being a poor match, the model used to perform the comparison must be relatively complete in its ability to describe all possible, or rather, expected paths. This is all but impossible in most non-trivial interfaces. Furthermore, the models must account for all manner “noise”, otherwise low level events, such as mouse movements, can mask otherwise significant correspondence between source sequences and the abstract model. Because these techniques typically have no built-in facilities for performing transformations on input traces, this implies that either the event stream has already been transformed, perhaps by careful hand instrumentation of the application (as with EMA), or great complexity must be introduced into the model to avoid sensitivity to “noise”. In contrast, techniques such as EDEM and EBBA use abstraction to pick out patterns of interest from the noise. The models need not be complete in any sense, and may entirely ignore events that are not of interest.

4.6 Sequence Characterization

4.6.1 Purpose

These techniques take “source” sequences as input and attempt to construct an abstract model to summarize, or characterize, interesting sequential features of those input sequences. Some techniques produce a process model with probabilities associated with transitions (e.g. Guzdial, 1993). Others may be used to construct models that characterize the grammatical structure of events in the input sequences (e.g. Olson et al., 1993).

4.6.2 Examples

Guzdial (Guzdial, 1993) describes a technique based on Markov Chain analysis that can be used to produce process models with probabilities assigned to transitions to characterize user behavior with interactive applications. First, abstract stages, or states, of application use are identified. In Guzdial’s example, a simple design environment was the object of study. Facilities provided by the design tool supported the following stages in a design process: “initial review”, “component definition”, “component composition”, “debugging”, and “final review”. Each of the operations in the interface could be mapped to one or another of these abstract design stages. For instance, all debugging related commands, which incidentally all appeared in a single “debugging” menu, could be mapped to the “debugging” stage. The stream of events was then abstracted and recoded to replace low level events with the abstract stages associated with them (presumably dropping events not associated with stages). The observed probability of entering any stage from the stage immediately before it was then computed to yield a transition matrix. The matrix can then be used to create a process diagram with probabilities associated with transitions. For example, one subject was observed to have transitioned from “debugging” to “component composition” more often (52% of all transitions out of

“debugging”) than to “component definition” (10%). A steady state vector was then computed to reflect the probability of any event chosen at random belonging to each particular stage. This could then be compared to an expected probability vector computed by simply calculating the percentage of commands associated with each stage. This comparison could then be used to indicate user “preference” for classes of commands.

Olson and colleagues (Olson et al., 1993) describe an approach based on statistical and grammatical techniques for characterizing the sequential structure of verbal interactions between participants in design meetings. Verbalizations were encoded into categories and subjected to statistical techniques, including log linear modeling and lag sequential analysis, to identify potential dependencies between events. These patterns were then used to suggest rules that might be included in a definite clause grammar used to summarize, or characterize some of sequential structure of the meeting interactions. These grammar rules were then used to rewrite some of the patterns embedded in the sequence (abstraction and recoding). The sequence was again subjected to statistical techniques and the grammar refined further in an iterative fashion. The result was a set of grammar rules that provided insight into the sequential structure of the meeting interactions.

4.6.3 Related

Techniques for process discovery (Cook & Wolf, 1996) are related in that they attempt to automatically generate a process model, in the form of a finite state machine, that accounts for a trace of events produced by a particular software process. It is not clear how well these techniques would perform with data as noisy as UI events. A more promising approach might be to perform filtering, abstraction, and recoding on the event stream before submitting it for analysis.

4.6.4 Strengths

The strength of these techniques lies in their ability to provide investigators with insight into the sequential structure of events embedded within sequences.

4.6.5 Limitations

The techniques described by Guzdial and Olson require extensive human involvement and can be quite time-consuming, particularly the grammar-based technique. On the other hand, the automated techniques suggested by Cook and Wolf appear to be fairly sensitive to noise, and are less likely to produce models that make sense to investigators (Olson et al., 1994).

Markov based models, while relying on over-simplifying assumptions, appear to be more likely than grammar-based techniques to tell investigators something about user interactions that is useful. Investigators often have an idea of the grammatical structure of interactions that may arise from the use of (at least portions of) a particular interface. Grammars are thus useful in transforming low level UI events into higher level patterns of interest, or to detect when actual usage patterns violate expected patterns. However, the value of generating a grammatical (or FSM-

based) model after the fact, is not entirely clear. More often than not, a grammar- or FSM-based model generated on the basis of multiple traces will be vacuous in that it will describe all the possible (or common) patterns of usage of an interface. While this may be useful in defining paths for UI regression testing, investigators interested in locating usability problems will be more interested in seeing actual counter-examples to what was expected, or just seeing actual instances of patterns, whatever they are, than seeing a grammar to summarize them all.

4.7 Visualization

4.7.1 Purpose

These techniques present the results of transformations and analyses in forms allowing humans to exploit their innate visual analysis capabilities to interpret results. These techniques are also particularly useful in linking results of analysis back to features of the interface.

4.7.2 Examples

A number of techniques have been used to visualize data based on UI events. For a survey of such techniques see (Guzdial et al, 1994). Below are few of examples of techniques that have been used in support of the analysis approaches described above.

Transformation:

The results of performing filtering or abstraction on an event stream can sometimes be visualized using a timeline in which bands of colors indicate different selections of events in the event stream. For example, one might use red to highlight the use of “Edit” menu operations and blue to highlight the use of “Font” menu operations in the evaluation of a word processor (Figure 3).

MacSHAPA (Sanderson & Fisher, 1994) visualize events as occupying cells in a spreadsheet. Event streams are listed vertically (in adjacent columns) and correspondence of events in one stream with events in adjacent streams is indicated by horizontal alignment (across rows). A large cell in one column may correspond to a number of smaller cells in another column indicate abstraction relationships (Figure 4).

Counts and summary statistics:

There are a number of visualizations that may be used to represent the results of counts and summary statistics, including static 2D and 3D graphs, static 2D effects superimposed on a coordinate space representing the interface, and static and dynamic 2D and 3D effects superimposed on top of an actual visual representation of the interface.

Static 2D and 3D graphs:

- Graph of keystrokes per window (e.g. Chang & Dillon, 1997).
- Graph of mouse clicks per window (e.g. Chang & Dillon, 1997).
- Graph of relative command frequencies (Figure 5) (Kay & Thomas, 1995).

- Graph of relative command frequencies as they vary over time (Figure 6) (Kay & Thomas, 1995).

Static 2D effects superimposed on an abstract coordinate space representing the interface:

- Location of mouse clicks (e.g. Guzdial, et al., 1994, Chang & Dillon, 1997).
- Mouse travel patterns between clicks (e.g. Buxton et al., 1983, Chang & Dillon, 1997).

Static and dynamic 2D and 3D effects superimposed on top of an actual visual representation of the interface (Guzdial, et al., 1994):

- Static highlighting to indicate location and density of mouse clicks.
- Dynamic highlighting of mouse click activity as it varies over time.
- 3D representation of mouse click density (Figure 8).

Sequence detection:

The results of selecting subsequences of UI events based on sequence detection techniques can be visualized using the same technique illustrated in (Figure 3).

EDEM provides a dynamic visualization that indicates the occurrence of abstract sequences of events by highlighting entries in a list of agents responsible for detecting those events. The same visualization is used to dynamically visualize (non-abstract) user interface events as they occur in user interface components. These visualizations are used by investigators to inspect the dynamic behavior of events to facilitate specification of the temporal patterns they are looking for (Hilbert & Redmiles, 1997).

Sequence comparison:

As described above, MacSHAPA (Sanderson & Fisher, 1994) provides facilities for aligning two sequences of events as optimally as possible and presenting the results visually as cells in adjacent spreadsheet columns with aligned events appearing in the same row, and missing cells indicating events in one sequence that could not be aligned with events in the other (Figure 7).

UsAGE (Ueling & Wolf, 1995) provides a similar visualization for comparing sequences based on drawing a connected graph of nodes. The “expert” series of actions is displayed linearly as a sequence of nodes across the top of the graph. The “novice” series of actions are indicated by drawing directed arcs connecting the nodes to represent the order in which the actions were performed by the novice. Out of sequence actions are indicated by arcs that skip expert nodes in the forward direction, or that point backwards in the graph. Unmatched actions taken by the novice appear as nodes (with a different color) placed below the last matched expert node.

Sequence characterization:

Guzdial (Guzdial, 1993) uses a connected graph visualization of the results of his Markov based analysis. The result is a process model with nodes representing

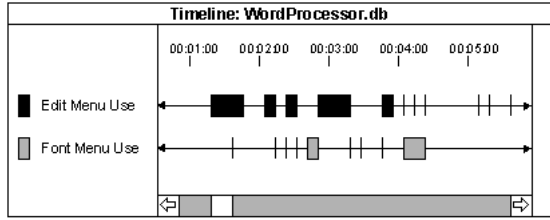


Figure 3. A timeline visualization depicting use of “Edit” menu operations (black) and “Font” menu operations (grey); used to display results of filtering or sequence detection.

OrderForm.db		
UI Events	Abs. Interaction Events	Task-Related. Events
⋮	⋮	⋮
GotFocus(Name)		
Key(Name, 'D')	GotEdit(Name)	AddressSection Started()
Key(Name, 'a')		
Key(Name, 'v')		
Key(Name, 'i')		
Key(Name, 'd')		
Lost Focus(Name)		
GotFocus(Street)		
	Lost Edit(Name)	
Key(Street, '1')	GotEdit(Street)	
	Value Provided(Name, "David")	Name Provided("David")
Key(Street, '3')		
Key(Street, '8')		
⋮	⋮	⋮
Lost Focus(ZIP)		
GotFocus(Quantity)		
	Lost Edit(ZIP)	
Key(Quantity, '1')	GotEdit(Quantity)	
	Value Provided(ZIP, "90740")	ZIP Provided("90740")
		AddressSection Completed()

Figure 4. Correspondence between events at different levels of abstraction are indicated by horizontal alignment; single “Key” events in large cells correspond to “LostEdit”, “GotEdit”, and “ValueProvided” abstract interaction events in smaller, horizontally aligned cells.

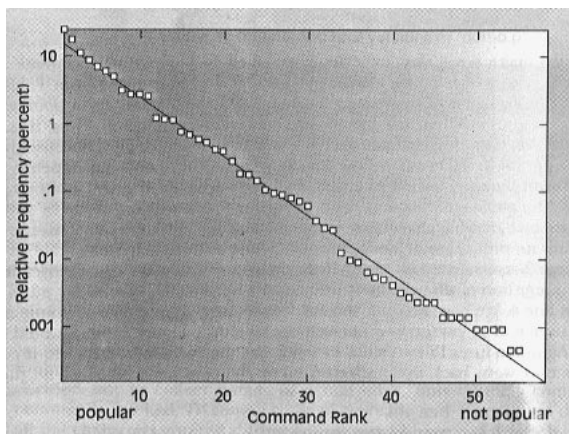


Figure 5. A graph of relative command frequencies ordered by popularity.

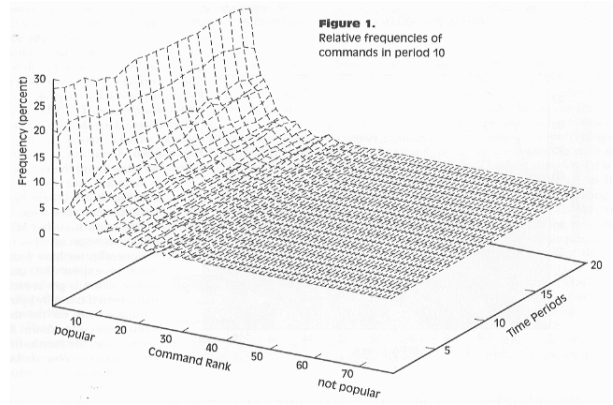


Figure 6. A Graph of relative command frequencies over time.

Meeting.db		
MeetingA	MeetingB	
Amplify	Amplify	
Resolve Issue	Resolve Issue	
Identify Problem	Identify Problem	
Amplify	Digress	
Digress	Amplify	
Recapitulate	Recapitulate	
Identify Issue	Identify Issue	
Amplify	Identify Issue	
Resolve	Identify Issue	
Identify	Identify Issue	
Resolve	Identify Issue	
Identify	Identify Issue	
Amplify	Identify Issue	
Recapitulate	Identify Issue	
Resolve	Identify Issue	
Amplify	Identify Issue	
Digress	Identify Issue	
	Identify Issue	

Figure 7. Results of automatic alignment of two event streams; horizontal alignment indicates correspondence; black spaces indicate where alignment was not possible.

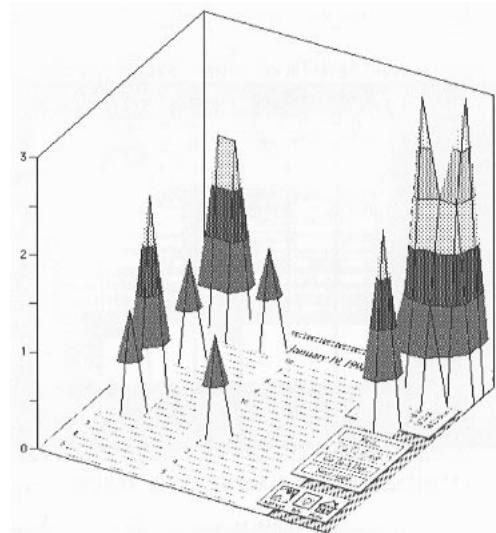


Figure 8. A 3D representation of mouse click density superimposed over a graphical representation of an electronic calendar application interface.

process steps and arcs indicating the observed probabilities of transitions between process steps.

4.7.3 Strengths

The strengths of these techniques lie in their ability to present the results of analysis in forms allowing humans to exploit their innate visual analysis capabilities to interpret results. Particularly useful are the techniques that link results of analysis back to features of the interface, such as the techniques superimposing graphical representations of behavior over actual representations of the interface.

4.7.4 Limitations

With the exception of simple graphs (which can typically be generated using standard graphing capabilities provided by spreadsheets and statistical analysis packages), most of the visualizations above must be produced by hand. Techniques for accurately superimposing graphical effects over actual representations of the interface can be particularly problematic.

4.8 Integrated Support

4.8.1 Purpose

Integrated support is provided by environments that facilitate flexible composition of various transformation, analysis, and visualization capabilities. Some environments also provide built-in support for managing domain-specific artifacts such as data regarding evaluations, subjects, tasks, and results of analysis.

4.8.2 Examples

MacSHAPA (Sanderson et al., 1994) is perhaps the most comprehensive environment designed to support all manner of exploratory sequential data analysis (ESDA). Features include: data import and export; video and coded observation synch and search capabilities; facilities for performing filtering, abstraction, and recoding; a number of built-in counts and summary statistics; features supporting sequence detection, comparison, and characterization; a general purpose database query and manipulation language; and a number of visualizations and reports.

DRUM (Macleod et al., 1993) provides integrated features for synchronizing events, observations, and video; for defining and managing observation coding schemes; for calculating pre-defined counts and summary statistics; and for managing and manipulating evaluation-related artifacts regarding subjects, tasks, recording plans, logs, videos, and results of analysis.

Hawk (Guzdial, 1993) provides flexible support for creating, debugging, and executing scripts to automatically filter, abstract, and recode event logs. Management facilities are also provided to organize and store event logs and analysis scripts.

4.8.3 Strengths

The task of extracting usability-related information from UI events typically requires the management of numerous files and media types as well as the creation and composition of various analysis techniques. Environments supporting the

integration of such activities can significantly reduce the burden of data management and integration.

4.8.4 Limitations

All the environments above display some rather important limitations.

While MacSHAPA is perhaps the most comprehensive integrated environment for analyzing sequential data, it is not specifically designed for analysis of UI events, and as a result, lacks support for UI event collection and focuses too much on analysis techniques that require extensive human intervention and interpretation to extract useful information. MacSHAPA provides many of the basic building blocks required for an “ideal” environment for capturing and analyzing UI events, but requires too much human effort to perform filtering, abstraction, and recoding that should be performed in an automated fashion. Furthermore, because the powerful features of MacSHAPA cannot be used *during* event collection, contextual information that might be useful in filtering and abstraction is not available.

While providing nice features for managing and analyzing UI events, coded observations, video data, and evaluation artifacts, DRUM provides no features for filtering, abstracting, and recoding data.

Finally, while Hawk addresses the problem of providing automated support for filtering, abstraction, and recoding, like MacSHAPA, it does not address UI event capture, and as a result, contextual information cannot be used to guide filtering and abstraction.

5 DISCUSSION

5.1 On the Importance of Transformation

Transformation is a critical prerequisite to meaningful analysis. Computing counts and summary statistics, e.g., feature usage, feature invocation techniques, cancellations, undos, and errors depends on it. Likewise, sequence analysis techniques, e.g. tracing, detecting, comparing, and characterizing sequences of interest also depend on it. However, meaningful transformation depends on at least three types of information not directly available in the event stream.

1. The first kind of information needed in transformation that is not available in the event stream regards mappings between UI events and application features. Features provide a means for subdividing and reasoning about application functionality. Such a mapping is critical in linking results of analysis back to the application and user interface being evaluated. For instance, computing the relative frequencies of feature use and feature invocation techniques as well as associating cancellations, undos, and errors with appropriate application features depends on such a mapping. As illustrated in Section 3.2, user interfaces often provide numerous techniques for invoking a single application feature (e.g. printing). An abstract model, such as the grammar presented in that section, can be indispensable in mapping between UI events and the application features with

which they are associated.

2. A second kind of information important in meaningful transformation involves mappings between lower level events and higher level events of interest. For instance, in order to analyze the sequence in which on-line forms are completed, modeling abstract events such as "GotEdit", "LostEdit" and "ValueProvided" can be indispensable in simplifying pattern specification and detection.
3. Finally, contextual information not available in the event stream but available in the user interface is often critical in properly interpreting the significance of UI events. For instance, the ability to query error dialogs for their error messages may be necessary in properly characterizing the types and relative frequencies of errors. Also, the ability to query user interface component values may be necessary to determine the causes for certain kinds of failures, for example, problems resulting from incompatible parameters being specified in application dialogs.

5.2 Current Approaches Fall Short

It is not entirely clear how these obstacles are to be overcome. It is natural to think that building data collection directly into a user interface management system (UIMS) or requiring applications to report events themselves are the most straightforward ways to side-step these issues. However, both of these approaches has important limitations.

User interface management systems (UIMS's) model the relationships between application features and UI events explicitly, so reasonable data collection mechanisms might be built directly in, as in the MIKE UIMS (Olsen & Halversen, 1988). Because UIMS's have dynamic access to most aspects of the user interface, contextual information important in transformation is also available. However, most developers do not use UIMS's, so a more general technique that does not depend on use of a UIMS must be devised.

An event-reporting API that allows applications to report events directly is certainly useful, particularly when events of interest cannot be inferred from UI events. Such an approach allows feature-related events to be reported by applications themselves and provides a more general solution than a UIMS-based approach. However, there is much useful usability-related information not directly available to applications that is easily captured in the event stream, e.g., shifts in input focus, mouse movements, and the particular UI actions used to invoke application features.

Finally, neither of these approaches directly addresses problems of analyzing the timing and sequencing of events.

5.3 Requirements for a Successful Solution

What is needed is an approach that assumes no more than a typical event-based user interface toolkit, e.g., those provided by MS Windows, the Macintosh OS, the X Window System, or the Java Abstract Window Toolkit (AWT). Developers should not be required to adopt a

particular UIMS and should not be required to call an API to report every potentially interesting event. This approach would need to provide:

- a means for modeling the relationships between UI actions and application features;
- a means for modeling relationships between lower level events and higher level events of interest;
- and a means for performing transformation in-context, so that contextual information can be used in interpretation.

5.4 Implications for Future Research

Before such an approach can actually be used to extract meaningful usability-related information from UI events automatically, more research must address:

- how to establish a reasonable mapping between UI events and application features that is maintainable as the application interface and application functionality evolves.
- how to establish a reasonable mapping between UI events and higher level events of interest that is also maintainable as the application interface and functionality evolves.
- how to allow data to be collected in-context without letting the evolution of data collection mechanisms to impact the evolution and deployment of applications.
- how to make transformation and analysis mechanisms easily adaptable to particular purposes by investigators wishing to employ them.

Performing transformation and analysis automatically and in-context has the potential of greatly reducing the amount of data that must be reported. It is an open question whether such an approach might be scaled up to large-scale and ongoing use over the Internet. If so, such an approach could lift current restrictions on evaluation size, location, and duration, and could help developers base design decisions on empirical data reflecting actual application use.

6 RELATED WORK

There are a number of related techniques that could provide useful insights into how to more effectively extract usability-related information from UI events:

- Model-based debugging and testing techniques (e.g. EBBA & TSL)
- Beta test data collection (e.g. Aqueduct Profiler)
- API usage monitoring (e.g. Hewlett Packard/Tivoli Systems Application Response-Time Measurement API)
- Event histories and undo mechanisms (e.g. Kosbie & Myers, 1994)
- Layered protocol models of interaction (e.g. Nielsen, 1986, Taylor, 1988)
- Command language grammars (CLG's) (Moran, 1981) and task-action grammars (TAG's) (Payne & Green, 1986)
- UI testing and debugging (e.g. Mercury Interactive XRunner/WinRunner, SunTest JavaStar)

- Process validation techniques (Cook & Wolf, 1997)
- Process discovery techniques (Cook & Wolf, 1996)

There are also a number of other techniques that are somewhat related, but that are less likely to provide important insights:

- Collaborative remote usability testing (e.g. see the “Remote Evaluation” web page, URL: http://hci.isc.vt.edu/~josec/remote_eval/)
- Simple event capture (e.g. Windows SPY)
- Macro recording/playback (e.g. Windows Macro Recorder)
- Programming by demonstration (e.g. Cypher et al., 1994)
- Task recognition and assisted-completion (e.g. Cypher, 1991)
- Enterprise management (e.g. TIBCO Hawk)
- Product condition monitoring
- Firewall monitoring and network intrusion detection
- Network monitoring and diagnostics
- Trade monitoring

7 CONCLUSIONS

A number of computer-aided techniques for extracting usability information from UI events have been surveyed and classified. The classification scheme includes the following categories: synch and search techniques, transformation techniques, techniques for performing simple counts and summary statistics, techniques for performing sequence detection, comparison, and

characterization, visualization techniques, and finally, techniques providing integrated evaluation support.

Transformation emerges as an absolutely critical subprocess in the overall process of extracting useful information from UI events and relating results of analysis back to the application and user interface being evaluated. Three types of information not available in the user interface event stream are identified as critical to meaningful analysis: (1) mappings between UI events and application features, (2) mappings between lower level events and higher level events of interest, and finally (3) contextual information available in the user interface at the time events occur, but not afterwards.

Mechanisms for modeling these relationships and accessing contextual information are proposed as requirements for an automated technique for extracting usability-related information from user interface events. Performing transformation and analysis automatically and in-context has the potential of greatly reducing the amount of data that must be reported. It is an open question whether such an approach might be scaled up to large-scale and ongoing use over the Internet. If so, such an approach could help lift current restrictions on evaluation size, location, and duration. Furthermore, it could help developers and managers actually address such empirical questions as: How thoroughly has beta testing exercised relevant features? How are application features actually being used? Which features warrant more or less development and testing effort? Which features if modified, added, or deleted are most likely to impact application utility and/or usability? Does actual usage match expectations and how might the design be improved to better match actual usage?

REFERENCES

Note: References listed by topic. A small number of references fall into multiple categories. As a result, the total number of references does not correspond exactly to the numbers provided here.

General Usability Evaluation:

1. R.M. Baecker, J. Grudin, W.A.S. Buxton, S. Greenberg (Eds.). *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann, San Mateo, CA, 1995.
2. V. Bellotti. A framework for assessing applicability of HCI techniques. In *Proceedings of INTERACT'90*. 1990.
3. A. Doubleday, M. Ryan, M. Springett, A. Sutcliffe. A comparison of usability techniques for evaluating design. In *Proceedings of DIS'97*. 1997.
4. J. Grudin. Utility and usability: Research issues and development contexts". *Interacting with computers*, Vol. 4, No. 5, 1992.
5. M. Helander (Ed.). *Handbook of human-computer interaction*. Elsevier Science Publishers B.V. (North Holland), 1998.
6. J. Nielsen. *Usability engineering*. Academic Press/AP Professional, Cambridge, MA, 1993.
7. J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-computer interaction*. Addison-Wesley, Wokingham, UK, 1994.
8. E.D. Smilowitz, M. J. Darnell, A.E. Benson. Are we overlooking some usability testing methods? A comparison of lab, beta, and forum tests. *Behaviour and Information Technology, Usability Laboratories Special Issue* (Ed.) J. Nielsen, Vol.13, No.1 & 2, 1994.
9. M. Sweeny, M. Maguire, B. Shackel. Evaluating human-computer interaction: A framework. *International Journal of Man-Machine Studies*, Vol.38, 1993.
10. R.N. Taylor and J. Coutaz. Workshop on Software Engineering and Human-Computer Interaction: Joint Research Issues. In *Proceedings of ICSE'94*. 1994.
11. A. Whitefield, F. Wilson, J. Dowell. A framework for human factors evaluation. *Behaviour and Information Technology*, Vol. 10, No. 1, 1991.

General Sequential Analysis:

12. A. Abbott. A Primer on sequence methods. *Organization Science*, Vol.4, 1990.
13. P.D. Allison, J.K. Liker. Analyzing sequential categorical data on dyadic interaction: A comment on Gottman. *Psychological Bulletin*, 2, 1987.
14. D.L. Cuomo. Understanding the applicability of sequential data analysis techniques for analysing usability data. *Behaviour and Information Technology, Usability Laboratories Special Issue* (Ed.) J. Nielsen,

Vol.13, No.1 & 2, 1994.

15. C. Fisher. Advancing the study of programming with computer-aided protocol analysis. In G. Olson, E. Soloway, S. Sheppard (Eds.), *Empirical Studies of Programmers*, 1987 Workshop. Ablex, Norwood, NJ, 1987.
16. C. Fisher. Protocol Analyst's Workbench: Design and evaluation of computer-aided protocol analysis. Unpublished PhD thesis, Carnegie Mellon University, Department of Psychology, Pittsburgh, PA, 1991.
17. C. Fisher and P. Sanderson. Exploratory sequential data analysis: exploring continuous observational data. *Interactions*, Vol.3, No. 2, ACM Press, Mar. 1996.
18. J.M. Gottman, A.K. Roy. *Sequential analysis: A guide for behavioral researchers*. Cambridge University Press, Cambridge, England, 1990.
19. G.P. Sackett. *Observing behavior* (Vol. 2). University Park Press, Baltimore, MD, 1978.
20. P.M. Sanderson, C. Fisher. Exploratory sequential data analysis: foundations. *Human-Computer Interaction Special Issue on ESDA*, Vol. 9, 1994.

Miscellaneous and Related:

21. A.V. Aho, B.W. Kernighan, P.J. Weinberger. *The AWK programming language*. Addison-Wesley, Reading, MA. 1988.
22. A. Cypher. EAGER: programming by repetitive tasks by example. In *proceedings of CHI'91*. 1991.
23. A. Cypher, D.C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, A. Turransky. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA., 1993.
24. P.M. Fitts. Perceptual motor skill learning. In A.W. Melton (Ed.), *Categories of human learning*. Academic Press, New York, NY, 1964.
25. S.J. Payne, T.R.G. Green. Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction*, Vol. 2, 1986.
26. D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, Vol. 18, 1975.
27. D.S. Kosbie, B.A. Myers. Extending programming by demonstration with hierarchical event histories (CMU-HCII-94-102). Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, 1994.
28. T.P. Moran. *The Command Language Grammar*. *IJMMS*, Vol. 15, 1981.
29. J. Nielsen. A virtual protocol model for computer-human interaction. *IJMMS*, Vol. 24, 1986.
30. S. Payne, T. Green. Task Action Grammars: a model of the mental representation of task languages. *Human-Computer Interaction*, Vol. 2, 1986.
31. M.M. Taylor. Layered protocols for computer-human dialogue I: Principles. *IJMMS*, Vol. 28, 1988.
32. M.M. Taylor. Layered protocols for computer-human dialogue I: Some practical issues. *IJMMS*, Vol. 28, 1988.

Synch and Search:

33. A.N. Badre, M. Guzdial, S. E. Hudson, P.J. Santos. A user interface evaluation environment using synchronized video, visualizations, and event trace data. *Journal of Software Quality*, Vol. 4, 1995.
34. D. E. Hoiem, K. D. Sullivan. Designing and using integrated data collection and analysis tools: challenges and considerations. *Behaviour and Information Technology, Usability Laboratories Special Issue* (Ed.) J. Nielsen, Vol.13, No.1 & 2, 1994.
35. M. Macleod, R. Rengger. The Development of DRUM: A Software Tool for Video-assisted Usability Evaluation. *In Proceedings of HCI'93*. 1993.
36. A.S. Neal, R.M. Simons. Playback: A method for evaluating the usability of software and its documentation. *In Proceedings of CHI'83*. 1983.
37. P. Weiler. Software for the usability lab: a sampling of current tools. *In Proceedings of INTERCHI'93*. 1993.

Event Stream Transformation:

38. A.N. Badre, P.J. Santos. CHIME: A knowledge-based computer-human interaction monitoring engine. Tech Report GIT-GVU-91-06. 1991.
39. A.N. Badre, P.J. Santos. A knowledge-based system for capturing human-computer interaction events: CHIME. Tech Report GIT-GVU-91-21. 1991.
40. J. Chen. Providing intrinsic support for user interface monitoring. *In Proceedings of INTERACT'90*. 1990.
41. B. Elgin. Subjective usability feedback from the field over a network. *In Proceedings of CHI'95*. 1995.
42. H.R. Hartson, J.C. Castillo, J. Kelso, W.C. Neale. Remote evaluation: the network as an extension of the usability laboratory. *In Proceedings of CHI'96*. 1996.
43. D.M. Hilbert, D.F. Redmiles. An approach to large-scale collection of application usage data over the Internet. *In Proceedings of ICSE'98*. 1998.
44. D.M. Hilbert, D.F. Redmiles. Agents for collecting application usage data over the Internet. *In Proceedings of Autonomous Agents'98*. 1998.

Counts and Summary Statistics:

45. E. Chang, T.S. Dillon. Automated usability testing. *In Proceedings of INTERACT'97*.
46. R. Cook, J. Kay, G. Ryan, R.C. Thomas. A toolkit for appraising the long-term usability of a text editor. *Software Quality Journal*, Vol. 4, No. 2, 1995.
47. J. Kay, R.C. Thomas. Studying long-term system use. *Communications of the ACM*, Vol. 38, No. 7, 1995.
48. J. Lowgren, T. Nordqvist. Knowledge-based evaluation as design support for graphical user interfaces. *In Proceedings of CHI'92*. 1992.
49. D.R. Olsen, B.W. Halversen. Interface usage measure-

ments in a user interface management system. *In Proceedings of UIST'88*. 1988.

Related:

50. Aqueduct Software. Aqueduct Profiler. URL: <http://www.aqueduct.com/>. 1998.
51. Hewlett Packard/Tivoli Systems. Application Response-Time Measurement. URL: <http://www.hp.com/openview/rpm/arm/>. 1998.

Sequence Detection:

52. S.V. Faraone, D.D. Dorfman. Lag sequential analysis: Robust statistical methods. *Psychological Bulletin*, 101, 1987.
 53. A. Girgensohn, D.F. Redmiles, and F.M. Shipman III. Agent-Based Support for Communication between Developers and Users in Software Design. *In Proceedings of the Knowledge-Based Software Engineering Conference '94*. Monterey, CA, USA, 1994.
 54. H.U. Hoppe. Task-oriented parsing: A diagnostic method to be used by adaptive systems. *In Proceedings of CHI'88*. 1988.
 55. D.M. Hilbert, D.F. Redmiles. An approach to large-scale collection of application usage data over the Internet. *In Proceedings of ICSE'98*. 1998.
 56. D.M. Hilbert, D.F. Redmiles. Agents for collecting application usage data over the Internet. *In Proceedings of Autonomous Agents'98*. 1998.
 57. P.M. Sanderson, J.J.P. Scott, T. Johnston, J. Mainzer, L.M. Watanabe, J.M. James. MacSHAPA and the enterprise of Exploratory Sequential Data Analysis (ESDA). *International Journal of Human-Computer Studies*, Vol. 41, 1994.
 58. P.J. Santos, A.N. Badre. Automatic chunk detection in human-computer interaction. *In Proceedings of Workshop on Advanced Visual Interfaces AVI '94*. Also available as Tech Report GIT-GVU-94-4. 1994.
 59. F. Schiele, H. U. Hoppe. Inferring task structures from interaction protocols. *In Proceedings of INTERACT'90*. 1990.
 60. A.C. Siochi, D. Hix. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. *In Proceedings of CHI'91*. 1991.
 61. A.C. Siochi, R.W. Ehrich. Computer analysis of user interfaces based on repetition in transcripts of user sessions. *ACM Transactions on Information Systems*. 1991.
- Related:*
62. P.C. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Transactions on Computer Systems*, Vol. 13, No. 1, 1995.
 63. M.S. Feather, K. Narayanaswamy, D. Cohen, S. Fickas. Automatic monitoring of software requirements.

Research Demonstration in *Proceedings of ICSE'97*. 1997.

64. S. Fickas, M.S. Feather. Requirements monitoring in dynamic environments. *IEEE International Symposium on Requirements Engineering*. 1995.
65. B. Krishnamurthy and D.S. Rosenblum. Yeast: A General Purpose Event-Action System. *IEEE Transactions on Software Engineering*, Vol. 21, No. 10, 1995.
66. M. Mansouri-Samani, M. Sloman. GEM: A generalised event monitoring language for distributed systems. *IEE/BCS/IOP Distributed Systems Engineering Journal*, Vol 4, No 2, 1997.
67. D.S. Rosenblum. Specifying concurrent systems with TSL. *IEEE Software*, Vol. 8, No. 3, 1991.
68. R.W. Selby, A.A. Porter, D.C. Schmidt, and J. Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of ICSE'91*. 1991.
69. TIBCO. HAWK Enterprise Monitor. URL: <http://www.tibco.com/products/products.html>. 1998.

Sequence Comparison:

70. S. Balbo. EMA: Automatic analysis mechanism for the ergonomic evaluation of user interfaces. CSIRO Technical report. 1996.
71. J. Finlay, M. Harrison. Pattern recognition and interaction models. In *Proceedings of INTERACT'90*. 1990.
72. D. L. Uehling, K. Wolf. User Action Graphing Effort (UsAGE). In *Proceedings of CHI'95*. 1995.

Related:

73. J.E. Cook, A.L. Wolf. Toward metrics for process validation. In *Proceedings of ICSP'94*. 1994.
74. J.E. Cook, A.L. Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. Tech Report CU-CS-840-97, Department of Computer Science, University of Colorado at Boulder. 1997.

Sequence Characterization:

75. M. Guzdial. Deriving software usage patterns from log files. Tech Report GIT-GVU-93-41. 1993.

76. M. Guzdial, C. Walton, M. Konemann, E. Soloway. Characterizing process change using log file data. Tech Report GIT-GVU-93-44. 1993.

77. G.M. Olson, J.D. Herbsleb, H.H. Rueter. Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Human-Computer Interaction Special Issue on ESDA*, Vol.9, 1994.

Related:

78. J.E. Cook, A.L. Wolf. Automating process discovery through event-data analysis. In *Proceedings of ICSE'95*. 1995.
79. B.T. Pentland. Grammatical models of organizational processes. *Organization Science*. 1994.
80. B.T. Pentland. A grammatical model of organizational routines. *Administrative Science Quarterly*. 1994.
81. A.L. Wolf and D.S. Rosenblum. A Study in Software Process Data Capture and Analysis. In *Proceedings of the Second International Conference on Software Process*, 1993.

Visualization:

82. W. Buxton, M. Lamb, D. Sheman, K. Smith. Towards a comprehensive user interface management system. In *Proceedings of SIGGRAPH'83*. 1983.
83. M. Guzdial, P. Santos, A. Badre, S. Hudson, M. Gray. Analyzing and visualizing log files: A computational science of usability. Presented at HCI Consortium Workshop, 1994.

Integrated Support:

84. M. Guzdial. Deriving software usage patterns from log files. Tech Report GIT-GVU-93-41. 1993.
85. M. Macleod, R. Rengger. The Development of DRUM: A Software Tool for Video-assisted Usability Evaluation. In *Proceedings of HCI'93*. 1993.
86. P.M. Sanderson, J.J.P. Scott, T. Johnston, J. Mainzer, L.M. Watanabe, J.M. James. MacSHAPA and the enterprise of Exploratory Sequential Data Analysis (ESDA). *International Journal of Human-Computer Studies*, Vol. 41, 1994.