Extracting Usability Information from User Interface Events

DAVID M. HILBERT AND DAVID F. REDMILES

University of California at Irvine

Modern window-based user interface systems generate user interface events as natural products of their normal operation. Because such events can be automatically captured and because they indicate 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 voluminos and rich in detail, automated support is generally required to extract information at a level of abstraction that is useful to investigators interested in analyzing application usage or evaluating usability.

This survey examines computer-aided techniques used by HCI practitioners and researchers to extract usability-related information from user interface events. A framework is presented to help HCI practitioners and researchers categorize and compare the approaches that have been, or might fruitfully be, applied to this problem. Because many of the techniques in the research literature have not been evaluated in practice, this survey provides a conceptual evaluation to help identify some of the relative merits and drawbacks of the various classes of approaches. Ideas for future research in this area are also presented.

This survey addresses the following questions: How might user interface events be used in evaluating usability? How are user interface events related to other forms of usability data? What are the key challenges faced by investigators wishing to exploit this data? What approaches have been brought to bear on this problem and how do they compare to one another? What are some of the important open research questions in this area?

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology*

General Terms: Human factors, Measurement, Experimentation

Additional Key Words and Phrases: usability testing, user interface event monitoring, sequential data analysis, human-computer interaction

1. INTRODUCTION

User interface events (UI events) are generated as natural products of the normal operation of window-based user interface systems such as those provided by the Macintosh Operating System [Lewis and Stone 1999], Microsoft Windows [Petzold 1998], the X Window System [Nye and O'Reilly 1992], and the Java Abstract Window Toolkit [Zukowski and Loukides 1997]. Such events indicate user

Authors' address: Department of Information and Computer Science, University of California, Irvine, CA e-mail: {dhilbert,redmiles}@ics.uci.edu.

behavior with respect to the components that make up an application's user interface (e.g., mouse movements with respect to application windows, keyboard presses with respect to application input fields, mouse clicks with respect to application buttons, menus, and lists). Because such events can be automatically captured and because they indicate 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 investigators interested in analyzing application usage or evaluating usability.

While a number of potentially related techniques have been applied to the problem of analyzing sequential data in other domains, this paper primarily focuses on 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 HCI events. However, this survey attempts 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 Method

The fundamental goal of this survey is to construct a framework to help HCI practitioners and researchers categorize, compare, and evaluate the relative 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, we attempt 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 an illuminating comparison can be conducted at the class level and that classification of new instances into existing classes will prove to be unproblematic.

In preparing this survey, we searched the literature in both academic and professional computing forums for papers describing computer-aided techniques for extracting usability-related information from user interface events. We selected and analyzed an initial set of papers to identify key characteristics that distinguish the approaches applied by various investigators.

We then constructed a two-dimensional matrix 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 clusters of related attributes. We then iteratively refined the comparison attributes and classification scheme based on further exploration of the literature. The resulting matrix indicates areas in which further research is needed and suggests synergistic combinations of currently isolated capabilities.

Ideally, an empirical evaluation of these approaches in practice would help elucidate more precisely the specific types of usability questions for which each approach is best suited. However, because many of the approaches have never been realized beyond the research prototype stage, little empirical work has been performed to evaluate their relative strengths and limitations. This survey attempts to provide a conceptual evaluation by distinguishing classes of approaches and illuminating their underlying nature. As a result, this survey should be regarded as a guide to understanding the research literature and not as a guide to selecting an already implemented approach for use in practice.

1.2 Comparison Framework

This subsection introduces the high level categories that have emerged as a result of the survey. We present the framework in more detail in Section 4.

- *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 recordings and coded observation logs. This allows searches in one medium to locate supplementary information in others. In some ways, this is the simplest (i.e., most mechanical) technique for exploiting user interface events in usability evaluation. However, it is quite powerful.

- *Techniques for transforming event streams.* Transformation involves selecting, abstracting, and recoding event streams to facilitate human and automated analysis (including counts, summary statistics, pattern detection, comparison, and characterization). *Selection* involves separating events and sequences of interest from the "noise." *Abstraction* involves relating events to higher-level concepts of interest in analysis. *Recoding* involves generating new event streams based on the results of selection and abstraction so that selected and abstracted events can be subjected to the same types of manual and automated analysis techniques normally performed on raw event streams.
- *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 computed to summarize user behavior, for example, 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.
- *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 are abstractly defined and are supplied by the developers of the technique. In other cases, target sequences are 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 is to perform transformation of the source sequence by abstracting and recoding instances of the target sequence into "abstract" events.
- *Techniques for comparing sequences.* These techniques help investigators compare *source* sequences against concrete or abstractly defined *target* sequences indicating the extent to which the sequences match one another. Some techniques attempt to detect divergence between an abstract model representing the target sequence and a source sequence. Others attempt to detect divergence between a concrete target sequence produced, for example, 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" or expected sequences to identify potential usability issues.
- *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 in order to produce process models with probabilities associated with transitions. Others construct grammatical

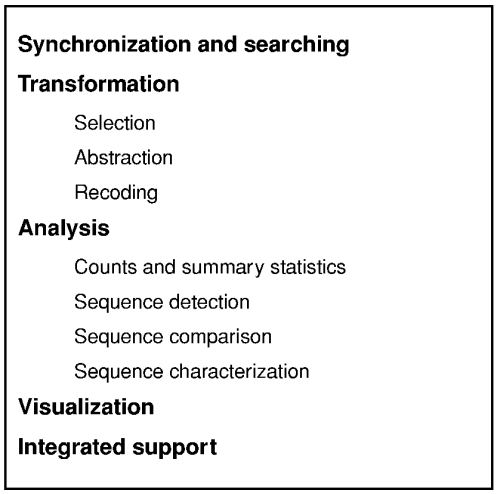


Fig. 1. Comparison framework.

models or finite state machines to characterize the grammatical structure of events in the source sequences.

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

Figure 1 illustrates how the framework might be arranged as a hierarchy. At the highest level, the surveyed techniques are concerned with: synchronization and searching, transformation, analysis, visualization, or integrated support. Transformation can be achieved through selection, abstraction, and recoding. Analysis can be performed using counts and summary statistics, sequence detection, sequence comparison, and sequence characterization.

1.3 Organization of the Survey

Section 2 establishes background and presents definitions of important terms. 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. Section 4 presents a comparison of the approaches based on the framework outlined above. For each class of techniques, the following is provided: a brief description, examples, related work where appropriate, and strengths and limitations. Section 5 summarizes the most important points of the survey and outlines some directions for future research. Section 6 presents conclusions.

Many of the authors are not explicit regarding the event representations assumed by their approaches. We assume that UI events are data structures that include attributes indicating an event type (e.g., `MOUSE_PRESSED` or `KEY_PRESSED`), an event target (e.g., an ID indicating a particular button or text field in the user interface), a timestamp, and other attributes including various aspects of the state of input devices when the event was generated. However, to raise the level of abstraction in our discussion, we typically represent event streams as sequences of letters where each letter corresponds to a more detailed event structure as described above. When beneficial, and possible, we provide examples using this notation to illustrate how particular approaches operate.

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 limitations 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

limitations relative to other types. The definitions and frameworks presented here are not new and can be found in standard HCI texts [Preece et al. 1994; 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 why analysis is complicated are presented.

2.1 Definitions

"*Usability*" is often thought of as referring to a single attribute of a system or device. However, it is more accurately 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 with 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 part of the definition is that the values of usability attributes can 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.

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 new terminology, this survey simply assumes that usability evaluations and usability data can address questions of utility as well as questions of usability.

2.2 Types of Usability 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

Table I. Types of Evaluation and Reasons for Evaluating

Reasons for Evaluating	Predictive Evaluation	Observational Evaluation	Participative Evaluation
Understanding user behavior & performance		X	x
Understanding user thoughts & experience		x	X
Comparing design alternatives	x	X	X
Computing usability metrics	x	X	X
Certifying conformance w/standards	X		

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 (potentially resulting in formative implications).

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

Predictive evaluation usually involves making predictions about usability attributes based on psychological modeling techniques (e.g., the GOMS model [John and Kieras 1996a; 1996b] or the Cognitive Walkthrough [Lewis et al. 1992]), or based on design reviews performed by experts equipped with a knowledge of HCI principles and guidelines and past experience in design and evaluation (e.g., Heuristic Evaluation [Nielsen and Mack 1994]). A key strength of predictive approaches is their ability to produce results based on nonfunctioning 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 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 I 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.

The approaches surveyed here are primarily geared toward supporting observational evaluation, although some provide limited support for capturing participative data as well.

Table II. Data Collection Techniques and Usability Indicators

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

2.3 Types of Usability Data

Having situated observational usability evaluation 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 online help usage, range of functions used.
- *Off-line behavior (nonverbal):* 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., posthoc 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 II 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 yet complementary 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. However, some investigators have used UI events to infer features of user knowledge and understanding [Kay and Thomas 1995; Guzdial et al. 1993].

Many of the surveyed approaches focus on event data exclusively. However, some also combine other sources of data including video recordings, coded observations, and subjective user reports.

3. THE NATURE OF UI EVENTS

This section discusses the nature and characteristics of HCI events in general and UI events specifically. We discuss the grammatical nature of UI events including some implications on analysis. We also discuss the importance of contextual information in interpreting the significance of events. Finally, we present a compositional model of UI events to illustrate how these issues manifest themselves in UI event analysis. We use this discussion to ground later discussion and to highlight some of the strengths and limitations of the surveyed approaches.

3.1 Spectrum of HCI Events

Before discussing the specific nature of UI events, this section introduces the broader spectrum of events of interest to researchers and practitioners in HCI. Figure 2, adapted from Sanderson and Fisher [1994], indicates the durations of different types of HCI events.

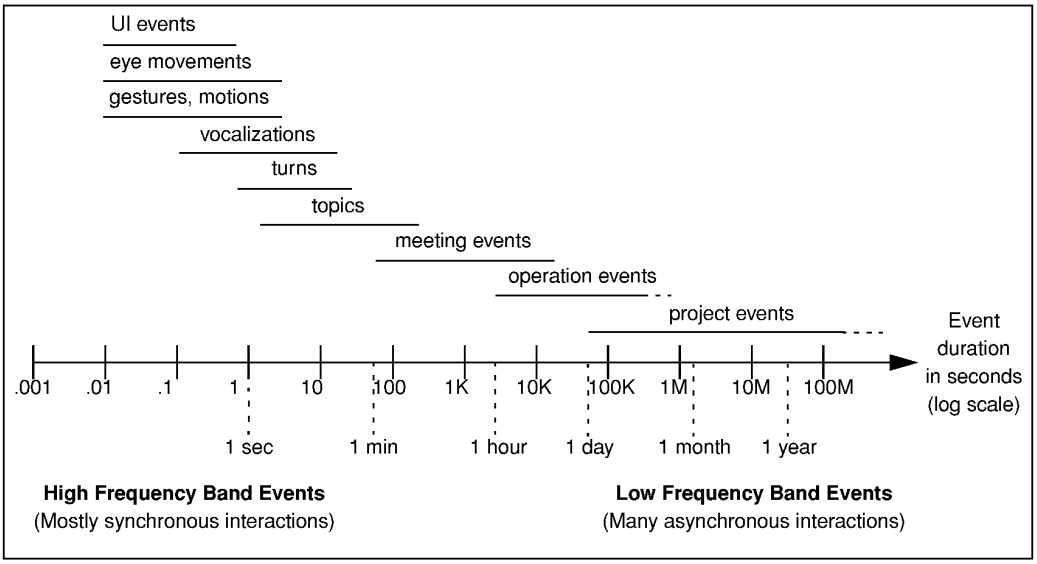


Fig. 2. A spectrum of HCI events. Adapted from [Sanderson and Fisher 1994].

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 and Fisher 1994]. Event types of shorter duration, for example, 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 and 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 and 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 and Fisher 1994]. This compositional structure is also exhibited *within* frequency bands. For instance, user interactions with software applications occur and may be analyzed at multiple levels of abstraction, where events at each level are composed of events occurring at lower levels. (See Hilbert et al. [1997] for an early treatment). This is discussed further below.

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. This too is discussed further below.

3.2 Grammatical Issues in Analysis

UI events are often grammatical in structure. Grammars have been used in numerous disciplines to characterize the structure of sequential data. 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_COMMAND –>
  "MOUSE_PRESSED PrintToolbarButton"or
  (PRINT_DIALOG_ACTIVATED then
  "MOUSE_PRESSED OkButton)

PRINT_DIALOG_ACTIVATED –>
  "MOUSE_PRESSED PrintMenuItem" or
  "KEY_PRESSED Ctrl – P"
```

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 the "OK" button. Rule 2 specifies that the print dialog may be activated by either selecting the print menu item in the "File" menu or by entering a keyboard accelerator, "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 each indicate a series of four consecutive print job activations:

```
AAAA
CDAAA
ABDBDA
BDCDACD
CDBDCDBD
```

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 selecting "OK."

Notice that each of these sequences contains a different number of lexical elements. Some of them have 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 salient differences between these sequences at the lexical level. Techniques for automatically detecting, comparing, and characterizing sequences are typically sensitive to such differences. Unless the data is transformed based on the

above grammar, the fact that these sequences are semantically equivalent (in the sense that each indicates a series of four consecutive print job activations) will most likely go unrecognized, and even simple summary statistics such as "# of print jobs per session" may be difficult to compute.

Techniques for extracting usability-related information from UI events should take into consideration the grammatical relationships between lower level events and higher level events of interest.

3.3 Contextual Issues in Analysis

Another set of problems arises in attempting to interpret the significance of UI events based only on the information carried within events themselves. 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 individuals A and B at a museum. The task is to identify A's favorite paintings based on utterances in the transcript.

Example 1: "*The Persistence of Memory*, by Dali, is one of my favorites."

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

Example 2: "*The Persistence of Memory*, by Dali."

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 could have been: "Which is your least favorite painting?"

Example 3: "That is one of my favorites."

In this case, we need the ability to dereference 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 could have been noted had the transcriber been present and 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 dereference an indexical.

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

Example 1′: "`MOUSE_PRESSED` PrintToolbarButton"

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

Example 2′: "`MOUSE_PRESSED` OkButton"

This event does not on its own indicate what action was performed. As in Example 2 previously, this event indicates a response to some prior event, for example, a prior "`MOUSE_PRESSED` PrintMenuItem" event.

Example 3′: "`WINDOW_OPENED` ErrorDialog"

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 dereferencing 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′: "`WINDOW_OPENED` ErrorDialog"

Assuming the error message is "Invalid Command," then the information needed to interpret the significance of this event is not only found by dereferencing 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 — does 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. Therefore, techniques for extracting usability-related information from UI events should take into consideration the fact that context may be spread across multiple events, and that in some cases, important contextual information may need to be explicitly captured during data collection if meaningful interpretation is to be performed.

3.4 Composition of Events

Finally, user interactions may be analyzed at multiple levels of abstraction. For instance, one may be interested in analyzing low-level mouse movement and timing information, or one may be more interested in higher-level information regarding the steps taken by users in completing tasks, such as placing an order or composing a business letter. Techniques for extracting usability information from UI events should be capable of addressing events at multiple levels of abstraction.

Figure 3 illustrates a multilevel model of events originally presented in Hilbert et al. [1997]. 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 user interface system, but may be computed based on

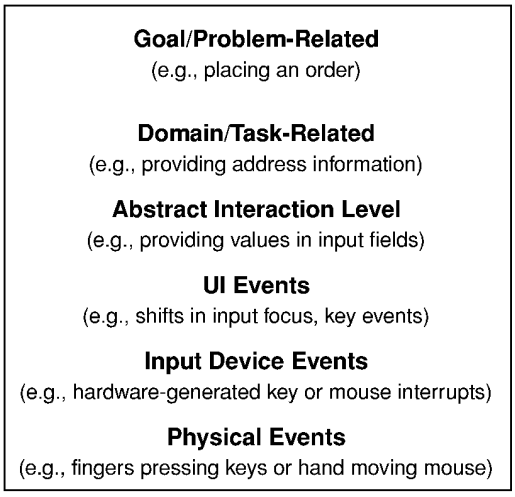


Fig. 3. Levels of abstraction in user interactions.

UI events and other contextual information such as UI state. Abstract interaction events are indicated by recurring, idiomatic patterns of UI events and indicate higher level concepts such as shifts in users' editing attention or the act of providing values to an application by manipulating application components.

Consider the example of a user editing an input field at the top of a form-based interface, then pressing tab repeatedly to edit a field at the bottom of the form. In terms of UI events, input focus shifted several times between the first and last fields. In terms of abstract interaction events, the user's editing attention shifted directly from the top field to the bottom field. Notice that detecting the occurrence of abstract interaction events such as "`GOT_EDIT`" and "`LOST_EDIT`" requires the ability to keep track of the last edited component and to notice subsequent editing events in other components.

Another type of abstract interaction event might be associated with the act of providing a new value to an application by manipulating user interface components. In the case of a text field, this would mean that the field had received a number of key events, was no longer receiving key events, and now contains a new value. The patterns of window system events that indicate an abstract interaction event such

as "VALUE_PROVIDED" will differ from one type of interface component to another, and from one application to another, but will typically remain fairly stable within a given application. Notice that detecting the occurrence of an abstract interaction event such as "VALUE_PROVIDED" requires the ability to access user interface state such as the component value before and after editing events.

Domain/task-related and *Goal/problem-related events* are at the highest levels. Unlike other levels, these events indicate progress in the user's tasks and goals. Inferring these events based on lower level events can be straightforward when the user interface provides explicit support for structuring tasks or indicating goals. For instance, Wizards in Microsoft Word™ [Rubin 1999] lead users through a sequence of steps in a predefined task. The user's progress can be recognized in terms of simple UI events such as button presses on the "Next" button. In other cases, inferring task and goal related events might require more complicated composite event detection. For instance, the goal of placing an order includes the task of providing address information. The task-related event "ADDRESS_PROVIDED" may be recognized in terms of "VALUE_PROVIDED" abstract interaction events occurring within each of the required fields in the address section of the form. Finally, in some cases, it may be impossible to infer events at these levels based only on lower level events.

Techniques for extracting usability-related information from UI events should be sensitive to the fact that user interactions can occur and be analyzed at multiple levels of abstraction.

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 following subsections discuss the features that distinguish each class of approaches and provide examples of some of the approaches in each class. We mention related work where appropriate and discuss the relative strengths and limitations of each class. Figures 4 through 11 provide pictorial representations of the key features underlying each class. Table III (located at the end of this section) presents a categorization and summary of the features belonging to each of the surveyed techniques.

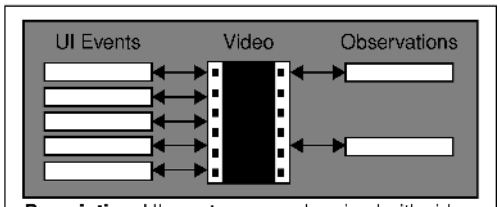
4.1 Synchronization and Searching

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

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 video recordings and experimenters' observations.

By synchronizing UI events with other sources of data such as video or coded observations, 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.

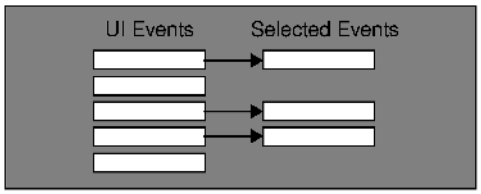
4.1.2 Examples. Playback is an early example of a system employing synch and



Description: UI events are synchronized with video and coded observations. Searches in one medium are used to locate supplementary information in others.

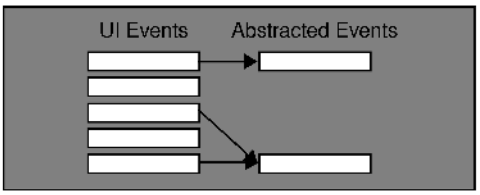
Examples: Playback; Microsoft, Apple, and SunSoft Labs; DRUM; MacSHAPA; I-Observe.

Fig. 4. Synchronization and searching.



Description: Selection is the process of separating events of interest from the rest of the event stream. Recoding is the process of generating a new event stream based on selected events.

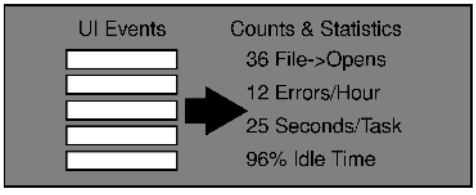
Examples: Incident Monitoring; CHIME; Hawk; MacSHAPA; User-Identified CIs; EDEM.



Description: Abstraction is the process of generating new events based on existing, or patterns of existing, events. Recoding is the process of generating a new event stream based on abstracted events.

Examples: CHIME; Hawk; MacSHAPA; User-Identified CIs; EDEM.

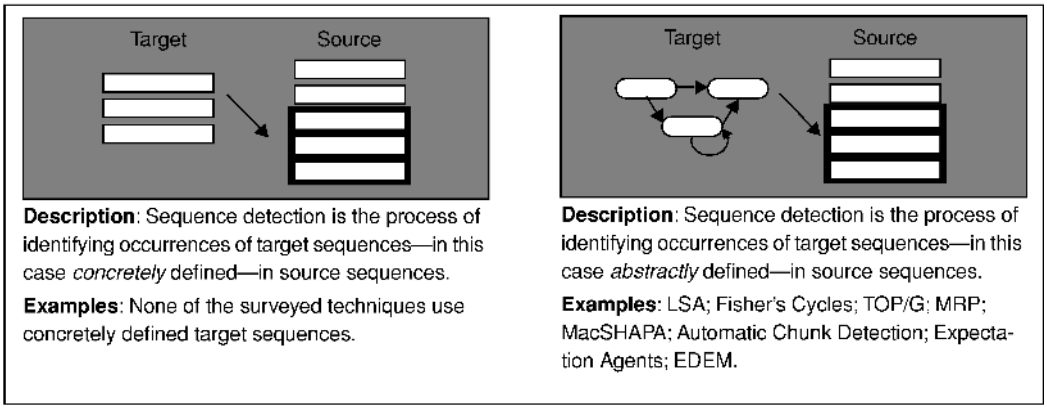
Fig. 5. Transformation.



Description: Counts and summary statistics are numeric values calculated based on UI events to characterize user behavior.

Examples: MIKE UIMS; KRI/AG; MacSHAPA; Long Term Monitoring; AUS.

Fig. 6. Counts and summary statistics.



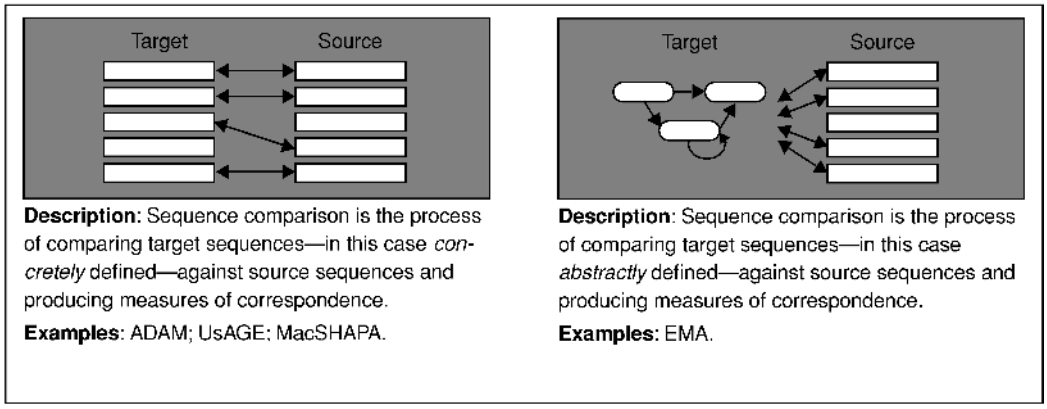
Description: Sequence detection is the process of identifying occurrences of target sequences—in this case *concretely* defined—in source sequences.

Examples: None of the surveyed techniques use concretely defined target sequences.

Description: Sequence detection is the process of identifying occurrences of target sequences—in this case *abstractly* defined—in source sequences.

Examples: LSA; Fisher's Cycles; TOP/G; MRP; MacSHAPA; Automatic Chunk Detection; Expectation Agents; EDEM.

Fig. 7. Sequence detection.



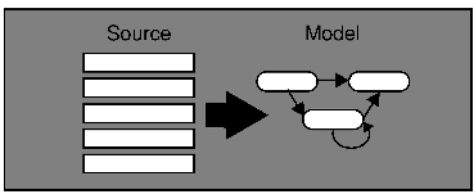
Description: Sequence comparison is the process of comparing target sequences—in this case *concretely* defined—against source sequences and producing measures of correspondence.

Examples: ADAM; UsAGE; MacSHAPA.

Description: Sequence comparison is the process of comparing target sequences—in this case *abstractly* defined—against source sequences and producing measures of correspondence.

Examples: EMA.

Fig. 8. Sequence comparison.



Description: Sequence characterization is the process of analyzing source sequences and generating abstract models to characterize the sequential structure of those sequences.

Examples: Markov-based; Grammar-based.

Fig. 9. Sequence characterization.

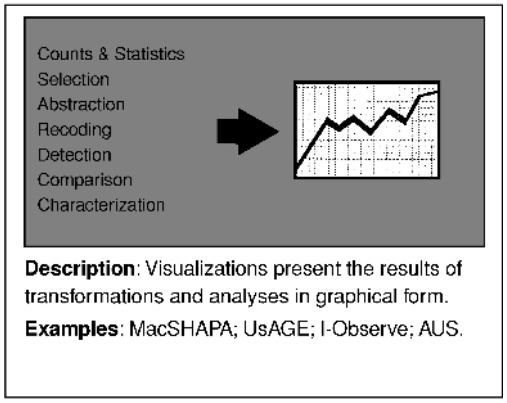


Fig. 10. Visualization.

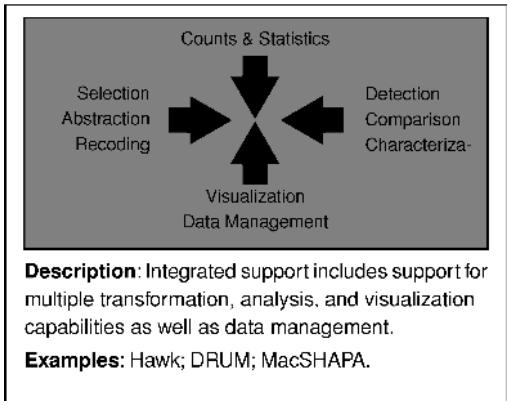


Fig. 11. Integrated Support.

search capabilities [Neal and Simmons 1983]. Playback captures UI events automatically and synchronizes them with coded observations and comments that are entered by experimenters either during or after the evaluation session. Instead of using video, Playback allows recorded events to be played back through the application interface to retrace the user's actions. The evaluator can step through the playback based on events or coded observations as if using an interactive debugger. There are a handful of simple built-in analyses to 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 is affected by events outside of user interactions. For instance, the behavior of some applications can vary depending on the state of networks and persistent data stores.

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 as part of the ESPRIT Metrics for Usability Standards in Computing Project (MUSiC). DRUM features a module for recording and synchronizing events, observations, and video, a module for defining and managing observation coding schemes, a module for calculating predefined counts and summary statistics, and a module for managing and manipulating evaluation-related information regarding subjects, tasks, recording plans, logs, videos, and results of analysis.

Usability specialists at Microsoft, Apple, and SunSoft all report the use of tools that provide synch and search capabilities [Weiler 1993; Hoiem and Sullivan 1994]. The tools used at Microsoft include a tool for logging observations, a tool for tracking UI events, and a tool for synchronizing and reviewing data from the multiple sources. The tools used at Apple and SunSoft are essentially similar. All tools support some level of event selection as part of the capture process. Apple's selection appears to be user-definable while Microsoft and SunSoft's selection appear to be programed into the capture tools. Scripts and general-purpose analysis programs, such as Microsoft Excel™ [Dodge and Stinson 1999], 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 UI event stream.

I-Observe, the "Interface Observation, Evaluation, Recording, and Visualization Environment," also provides synch and search capabilities [Badre et al. 1995].

Table III. 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	Select/ Recode	Abstract/ Recode	Counts & Stats	Sequence Detect	Sequence Compare	Sequence Character	Visualize	Data Manage	UI Platform
Section 4.1	Synch & Search	Playback	[Neal & Simmons 1983]	Yes	Obs.				Built-in						Unknown
		Apple Lab	[Weiler 1993]	Yes	Obs.+Vid.		Built-in								MacOS
		SunSoft Lab	[Weiler 1993]	Yes	Obs.+Vid.		Built-in								X Windows
		Microsoft Lab	[Hoiem & Sullivan 1994]	Yes	Obs.+Vid.		Built-in								MS Windows
		I-Observe	[Badre et al. 1995]	Yes	Vid.					Model			Built-in		X Windows
Section 4.2	Transform	Incident Monitoring	[Chen 1990]	Yes		UI	Built-in								X Windows
		User Identified CIs	[Hartson et al. 1996]	Yes		User	User	User							Unknown
		CHIME	[Badre & Santos 1991]	Yes		UI	Model	Model							X Windows
		EDEM	[Hilbert & Redmiles 1997]	Yes		UI+User	Model+User	Model+User	Built-in	Model			Built-in		Java AWT
Section 4.3	Counts & Stats	UIMS	[Buxton et al. 1983]	Yes					Built-in				Built-in		UIMS
		MIKE	[Olsen & Halversen 1988]	Yes					Built-in						UIMS
		KRI/AG	[Lowgren & Nordqvist 1992]	Yes					Built-in						UIMS
		Long-Term Monitoring	[Kay & Thomas 1995]	Instr.					Programs				Graphs		API
		AUS	[Chang & Dillon 1997]	Yes					Built-in				Built-in		MS Windows
		Aqueduct AppScope	[Aqueduct Software 1998]	Instr.					Database				Database		API
		Full Circle Talkback	[Full Circle Software 1998]	Instr.					Database				Database		API
Section 4.4	Sequence Detect	LSA	[Sackett 1978]							Built-in					
		Fisher's Cycles	[Fisher 1991]							Built-in					
		TOP/G	[Hoppe 1988]	Sim.		Sim.				Model					Simulation
		MRP	[Siochi & Hix 1991]							Built-in					
		Expectation Agents	[Girgensohn et al. 1994]	Yes		UI+User	User	User		Model					OS/2
		EDEM	[Hilbert & Redmiles 1997]	Yes		UI+User	Model+User	Model+User	Built-in	Model					Java AWT
		USINE	[Lecerof & Paterno 1998]	Yes					Built-in	Model	Model		Built-in		X Windows
		TSL	[Rosenblum 1991]							Model					N/A
		Amadeus	[Selby et al. 1991]							Model					N/A
		YEAST	[Krishnamurthy & Rosenblum 1995]					Model		Model					N/A
		EBBA	[Bates 1995]			App		Model		Model					N/A
		GEM	[Mansouri-Samani & Sloman 1997]			App	Model	Model		Model					N/A
Section 4.5	Sequence Compare	ADAM	[Finlay & Harrison 1990]								Concrete				Unknown
		UsAGE	[Ueling & Wolf 1995]	Yes							Concrete		Built-in		UIMS
		EMA	[Balbo 1996]	Instr.							Model				API
		USINE	[Lecerof & Paterno 1998]	Yes					Built-in	Model	Model		Built-in		X Windows
		Process Validation	[Cook & Wolf 1997]								Model				N/A
Section 4.6	Sequence Character	Markov-based	[Guzdial 1993]					Model				Manual			
		Grammer-based	[Olson et al. 1994]					Model				Manual			
		Process Discovery	[Cook & Wolf 1995]									Auto			N/A
Section 4.7	Integrated Support	Hawk	[Guzdial 1993]				Script	Script	Script	Script				Built-in	
		DRUM	[Macleod & Rengger 1993]	Yes	Obs.+Vid.				Built-in					Built-in	MacOS
		MacSHAPA	[Sanderson et al. 1994]		Obs.+Vid.		Database	Database	Built-in	Built-in+DB	Concrete	Manual	Built-in	Built-in	
		Ergolight EORS/EUVS	[Ergolight Usability Software 1998]	Yes					Built-in	Model	Model?		Built-in	Built-in	MS Windows

I-Observe is a set of loosely integrated tools for collecting, selecting, analyzing, and visualizing event data that has been synchronized with video recordings. Investigators can perform searches by specifying predicates over the attributes contained within a single event record. Investigators can then locate patterns of events by stringing together multiple search specifications into regular expressions. Investigators can then use intervals matched by such regular expressions (identified by begin and end events) to select data for visualization or to display the corresponding segments of the video recording.

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 video recordings and coded observations.

4.1.4 Limitations. Techniques relying on synchronizing UI events with video and coded observations typically require the use of video recording equipment and the presence of observers. The use of video equipment and the presence of observers can make subjects self-conscious and affect performance and may not be practical or permitted in certain circumstances. Furthermore, video-based evaluations tend to produce massive amounts of data that can be expensive to analyze. The ratio of the time spent in analysis versus the duration of the sessions being analyzed has been known to reach 10:1 [Sanderson and Fisher 1994; Nielsen 1993; Sweeny et al. 1993]. These matters are all serious limiting factors on evaluation size, scope, location, and duration.

Some researchers have begun investigating techniques for performing collaborative remote usability evaluations using video-conferencing software and application sharing technologies. Such techniques may help lift some of the limitations on evaluation location. However, more work must be done in the area of automating data collection and analysis if current restrictions on evaluation size, scope, and duration are to be addressed.

4.2 Transformation

4.2.1 Purpose. These techniques combine selection, 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.

Selection operates by subtracting information from event streams, allowing events and sequences of interest to emerge from the "noise." Selection involves specifying constraints on event attributes to indicate events of interest to be separated from other events or to indicate events to be separated from events of interest. For instance, one may elect to disregard all events associated with mouse movements in order to focus analysis on higher level actions such as button presses and menu selections. This can be accomplished by "positively" selecting button press and menu selection events or by "negatively" selecting, or filtering, mouse movement events.

Abstraction operates by synthesizing new events based on information in the event stream, supplemented (in some cases) by contextual information outside of the event stream. For instance, a pattern of events indicating that an input field had been edited, that a new 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. Furthermore, the same abstract event might be indicated by different events in different UI components, for example, mouse events typically indicate

editing in nontextual components while keyboard events typically indicate editing in textual components. One may also wish to synthesize events to relate the use of particular UI components to higher level concepts such as the use of menus, toolbars, or dialogs to which those components belong.

Recoding involves producing new event streams based on the results of selection and abstraction. This allows the same manual or automated analysis techniques normally applied to raw event streams to be applied to selected and abstracted events, potentially leading to different results. Consider the example presented in Section 3. If the sequences representing four sequential print job activations were embedded within the context of a larger sequence, 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 presents an approach to user interface event monitoring that selects events based on the notion of "incidents" [Chen 1990]. 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 X Toolkit Intrinsics [Nye and O'Reilly 1992] to report events that trigger callback procedures registered by applications. This allows events not handled by the application to be selected "out" automatically. Investigators may further constrain event reporting by selecting specific incidents of interest to report. 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 report an approach to remote collaborative usability evaluation that relies on users to select events [Hartson et al. 1996]. 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. The approach uses E-Mail to report digitized video of the events leading up to and following critical incidents along with contextual information provided by users. In this case, selection 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).[1]

CHIME, the "Computer-Human Interaction Monitoring Engine," is similar, in some ways, to Chen's approach [Badre and Santos 1991a]. CHIME allows investigators to select ahead of time which events to report and which events to filter. An important difference is that CHIME also 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.[2]

Hawk is an environment for selecting, abstracting, and recoding events in log files [Guzdial 1993]. 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™ [Goodman 1998]. Events appear in the

[1] This approach actually focuses on capturing video and not events. However, the ideas embodied by the approach can equally well be applied to the problem of selecting events of interest surrounding user-identified critical incidents.

[2] 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 never have been implemented [Badre and Santos 1991a; 1991b].

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 selection, abstraction, and recoding to be performed.

MacSHAPA, which is discussed further later, supports selection and recoding via a database query and manipulation language that allows investigators to select event records based on attributes and define new streams based on the results of queries [Sanderson et al. 1994]. Investigators can also perform abstraction by manually entering new records representing abstract events and visually aligning them with existing events in a spreadsheet representation (see Figure 13).

EDEM, an "Expectation-Driven Event Monitoring" system, captures UI events and supports automated selection, abstraction, and recoding [Hilbert and Redmiles 1998a]. Selection is achieved in two ways: investigators specify ahead of time which events to report, and users can also cause events to be selected via a critical incident reporting mechanism akin to that reported in Hartson et al. [1996]. However, one important difference is that EDEM also allows investigators 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, investigators can use EDEM to define abstract events in terms of patterns of existing events. When EDEM detects a pattern of events corresponding to a predefined abstract event, it generates an abstract event and inserts it into the event stream. Investigators can configure EDEM to perform further hierarchical event abstraction by simply defining higher level abstract events in terms of lower level abstract events. All of this is done in context, so that contextual information can be used in selection and abstraction. EDEM also calculates a number of simple counts and summary statistics.

4.2.3 Strengths. The main strength of these approaches lies in their explicit support for selection, abstraction, and recoding which are essential steps in preparing UI events for most types of analysis (as illustrated in Section 3). Chen and CHIME address issues of selection 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 already captured UI events.

4.2.4 Limitations. The techniques that select, abstract, and recode events while collecting them run the risk of throwing away data that might have been useful in analysis. Techniques that rely exclusively on users to select events are even more likely to drop useful information. With the exception of EDEM, the approaches that support flexible abstraction and recoding do so only after events have been captured, meaning contextual information that can be critical in abstraction is not available.

4.3 Counts and Summary Statistics

4.3.1 Purpose. As noted previously, 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 systems with specific built-in facilities for performing and reporting such calculations. This section provides examples of some of the systems boasting specialized facilities for calculating usability-related metrics.

4.3.2 Examples. The MIKE user interface management system (UIMS) is an early example of a system offering built-in facilities for calculating and reporting metrics [Olsen and Halversen 1988]. Because MIKE controls all aspects of input and output activity, and because it has an abstract description that links user interface components to the application commands they trigger, 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 also includes numerous built-in features to support computation and reporting of simple counts and summary statistics, including, for instance, the frequencies and durations of any selection of events specified by the user [Sanderson et al. 1994]. MacSHAPA's other more powerful analysis features are described in following sections.

Automatic Usability Software (AUS) 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 [Chang and Dillon 1997].

Finally, ErgoLight Operation Recording Suite (EORS) and Usability Validation Suite (EUVS) [ErgoLight Usability Software 1998] provide a number of built-in counts and summary statistics to characterize user interactions captured by the tools, either locally in the usability lab, or remotely over the Internet.

4.3.3 Related. A number of commercial tools such as Aqueduct AppScope™ [Aqueduct Software 1998] and Full Circle Talkback™ [Full Circle Software 1998] have recently become available for capturing data about application crashes over the Internet. These tools capture metrics about the operating system and application at the time crashes occur, and Talkback allows users to provide feedback regarding the actions leading up to crashes. These tools also provide APIs that application developers can use to report events of interest, such as application feature usage. These tools then send captured data via E-mail 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 calculations 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 linking user interface components to application

commands, as in the case of MIKE, or when the application code is manually instrumented to report the events to be analyzed, as in the case of AppScope and Talkback. AUS does not address application-specific features and thus is limited in its ability to relate metrics results to application features.

4.4 Sequence Detection

4.4.1 Purpose. These techniques detect occurrences of concrete or abstractly defined *target* sequences within *source* sequences.[3] In some cases target sequences are 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 are more specific to a particular application and are supplied by the investigators using the technique (e.g., TOP/G, Expectation Agents, and EDEM). 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 EDEM). 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., EDEM).

4.4.2 Examples. TOP/G, the "Task-Oriented Parser/Generator," parses sequences of commands from a command-line simulation and attempts to infer the higher level tasks that are being performed [Hoppe 1988]. Users of the technique model expected tasks in a notation based on Payne and Green's task-action grammars [Payne and Green 1986] and store this information as rewrite or production rules in a Prolog database. Investigators can define composite tasks hierarchically in terms of elementary tasks, which they must further decompose into "triggering rules" that map keystroke level events into elementary tasks. Investigators may also define rules 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, TOP/G's generator functionality could be used to generate the shorter, or more "optimal," command sequence.

Researchers involved in exploratory sequential data analysis (ESDA) have applied a number of techniques for detecting abstractly defined patterns in sequential data. For an in-depth treatment see [Sanderson and Fisher 1994]. These techniques can be subdivided into two basic categories:

Techniques sensitive to sequentially separated patterns of events, for example:

- Fisher's cycles
- Lag sequential analysis (LSA)

Techniques sensitive to strict transitions between events, for example:

- Maximal Repeating Pattern Analysis (MRP)
- Log linear analysis
- Time-series analysis

Fisher's cycles 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) [Fisher 1991]. 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

[3] The following sentences include names of approaches in parentheses to indicate how the examples explained in the next subsection fall into finer subcategories of the broader "sequence detection" category.

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
ABACDACDBADBCACCCD
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 frequency with which two events occur at various "removes" from one another [Sackett 1978; Allison and Liker 1982; Faraone and Dorfman 1987; Sanderson and 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 and 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 since it only occurs once in the source sequence.

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 and Fisher 1994]. Other related techniques include log linear analysis [Gottman and Roy 1990] and formal time-series analysis [Box and

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 and Badre 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 actively *executing* plans, as opposed to engaging in problem solving and planning 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 approach uses the results of the algorithm 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 and Redmiles 1998b] rely on sequence detection techniques to trigger various actions in response to pre-specified patterns of events. These approaches employ an event pattern language to allow investigators to specify composite events to be detected. When a pattern of interest is detected, contextual information may also 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 UI state and events leading up to detected patterns. Investigators may also configure EDEM to abstract and recode event streams to indicate the occurrence of abstract events associated with pre-specified event patterns.

4.4.3 Related. EBBA is a debugging system that attempts to match the behavior of a distributed program against partial models of expected behavior [Bates 1995]. EBBA is similar to EDEM, particularly in its ability to abstract and recode the event stream based on hierarchically defined abstract events.[4]

Amadeus [Selby et al. 1991] and YEAST [Krishnamurthy and Rosenblum 1995] are event-action systems used to detect and take actions based on patterns of events in software processes. These systems 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. The event specification notations used in these approaches might be applicable 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 events. The techniques associated with ESDA help investigators detect patterns that may not have been anticipated. Languages for detecting patterns of interest in UI events based on extended regular expressions [Sanderson and Fisher 1994] or on more grammatically inspired techniques [Hilbert and Redmiles 1998b] can be used to locate patterns of interest and to transform event streams by recoding patterns of events into abstract events.

4.4.5 Limitations. The ESDA techniques described above tend to produce large

[4] EBBA 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.

amounts of output that can be difficult to interpret and that frequently do not lead to identification of usability problems [Cuomo 1994]. The non-ESDA techniques require investigators to know how to specify the patterns for which they are searching and to define them (sometimes painstakingly) before analysis can be performed.

4.5 Sequence Comparison

4.5.1 Purpose. These techniques compare *source* sequences against concrete or abstractly defined *target* sequences indicating partial matches between the two.[5] Some techniques attempt to detect divergence between an abstract model of the target sequence and the source sequence (e.g., EMA and USINE). Others attempt to detect divergence between a concrete target sequence produced, for example, 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 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 and USINE). 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.[6]

[5] The following sentences include names of approaches in parentheses to indicate how the examples explained in the next subsection, fall into finer subcategories of the broader "sequence comparison" category.

[6] TOP/G, Expectation 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 indicating expected behavior.

4.5.2 Examples. ADAM, an "Advanced Distributed Associative Memory," compares fixed length source sequences against a set of target sequences that were used to *train* the memory [Finlay and Harrison 1990]. Investigators train ADAM by helping it associate example target sequences with "classes" of event patterns. After training, when a source sequence is input, the associative memory identifies the class that most closely matches the source sequence and outputs two diagnostic measures: 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 measures to determine whether a source sequence is different enough from the trained sequences to be judged as a possible "critical incident." Incidentally, ADAM might also be trained on examples of "expected" critical incidents so that these might be detected directly.

MacSHAPA [Sanderson and 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 (see Figure 17).

UsAGE applies a related technique in which a source sequence of UI events (related to performance of a specific task) is aligned as optimally as possible with a target sequence produced by an "expert" performing the same task [Ueling and Wolf 1995]. UsAGE presents its alignment results in visual form.

EMA, an "automatic analysis mechanism for the ergonomic evaluation of user interfaces," requires investigators to provide a grammar-based model describing all the expected paths through a particular user interface [Balbo 1996]. An evaluation program then compares a log of events generated by use of the interface against

the model, indicating in the log and the model where the user has taken "illegal" paths. EMA also detects and reports the occurrence of other simple patterns, for example, the use of cancel or repeated actions. The evaluator can then use this information to identify problems in the interface (or problems in the model).

USINE is a similar technique [Lecerof and Paterno 1998], in which investigators use a hierarchical task notation to specify how lower-level actions combine to form higher-level tasks, and to specify sequencing constraints on actions and tasks. The tool then compares logs of user actions against the task model. All actions not specified in the task model, or actions and tasks performed "out of order" according to the sequencing constraints specified in the task model, are flagged as potential errors. The tool then computes a number of built-in counts and summary statistics including number of tasks completed, errors, and other basic metrics (e.g., window resizing and scrollbar usage) and generates simple graphs.

ErgoLight Usability Validation Suite (EUVS) also compares user interactions against hierarchical representations of user tasks [ErgoLight Usability Software 1998]. EUVS is similar in spirit to EMA and USINE with the added benefit that it provides a number of built-in counts and summary statistics regarding general user interface use in addition to automatically-detected divergences between user actions and the task model.

4.5.3 Related. Process validation techniques are related in that they compare actual traces of events generated by a software process against an abstract model of the intended process [Cook and Wolf 1997]. These techniques compute a diagnostic measure of distance 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 and Wolf [1997] for further discussion and pointers to relevant literature.

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 problems. 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 source sequences against concrete target sequences is the underlying assumption that: (a) source and target sequences can be easily segmented for piecemeal comparison, as in the case of ADAM, or (b) that whole interaction sequences produced by different users 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. In contrast to techniques that completely match patterns that directly indicate violations of expected patterns (e.g., as in the case of EDEM), these techniques produce output to the effect, "the source sequence is similar to a target sequence with a correspondence measure of 61%," leaving it up to investigators to decide on a case by case basis what exactly the correspondence measure means.

A key limitation of any technique comparing sequences against abstract models (e.g., EMA, USINE, ErgoLight EUVS, 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 nontrivial interfaces. Furthermore, the model must somehow deal with "noise" so that low-level events, such as mouse movements, won't 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 manually instrumenting the application (as with EMA), or complexity must be introduced into the model to avoid sensitivity to "noise." In contrast, techniques such as EDEM and EBBA use selection and abstraction to pick out patterns of interest from the noise. The models need not be complete in any sense and may 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 sequences. Some techniques produce a process model with probabilities associated with transitions [Guzdial 1993]. Others construct models that characterize the grammatical structure of events in the input sequences [Olson et al. 1994].

4.6.2 Examples. Guzdial describes a technique, based on Markov Chain analysis, that produces process models with probabilities assigned to transitions to characterize user behavior with interactive applications [Guzdial 1993]. First, the investigator identifies abstract stages, or states, of application use. In Guzdial's example, a simple design environment was the object of study. The design environment provided functions supporting the following stages in a simple design process: "initial review," "decomposition," "composition," "debugging," and "final review." The investigator then creates a mapping between each of the operations in the interface and one of the abstract stages. For instance, Guzdial mapped all debugging related commands (which incidentally all appeared in a single "debugging" menu) to the "debugging" stage. The investigator then uses the Hawk tool to abstract and record the event stream to replace low level events with the abstract stages associated with them (presumably dropping events not associated with stages). The investigator then uses Hawk to compute the observed probability of entering any stage from the stage immediately before it to yield a transition matrix. The investigator can then use the matrix to create a process diagram with probabilities associated with transitions. In Guzdial's example, one subject was observed to have transitioned from "debugging" to "composition" more often (52% of all transitions out of "debugging") than to "decomposition" (10%) (see Figure 18). Guzdial then computed a steady state vector to reflect the probability of any event chosen at random belonging to each particular stage. He could then compare this to an expected probability vector (computed by simply calculating the percentage of commands associated with each stage) to indicate user "preference" for classes of commands.

Olson and colleagues describe an approach, based on statistical and grammatical techniques, for characterizing the sequential structure of verbal interactions between participants in design meetings [Olson et al. 1994]. They begin by mapping meeting verbalizations into event categories that are then used to manually encode the transcript into an event sequence representation. The investigator then applies statistical techniques, including log linear modeling and lag sequential analysis, to identify potential dependencies between events in the sequence. The investigator then uses the results of these pattern detection techniques to suggest rules that might be included in a definite clause grammar to summarize, or characterize, some of the sequential structure of the meeting interactions. The investigator then uses the resulting grammar rules to rewrite some of the patterns embedded in the sequence (i.e., abstraction and recoding), and the new sequence is subjected to the same statistical techniques leading to further iterative refinement of the grammar. The result is a set of grammar rules

that provide insight into the sequential structure of the meeting interactions.

4.6.3 Related. Process discovery techniques 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 [Cook and Wolf 1995]. 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 selection, abstraction, and recoding of the event stream prior to submitting it for analysis.

4.6.4 Strengths. The strength of these techniques lies in their ability to help investigators *discover* sequential structure within event sequences and to characterize that structure abstractly.

4.6.5 Limitations. The technique described by Olson and colleagues requires extensive human involvement and can be very time-consuming [Olson et al. 1994]. On the other hand, the automated techniques suggested by Cook and Wolf appear to be sensitive to noise and are less likely to produce models that make sense to investigators [Olson et al. 1994].

In our opinion, Markov-based models, while relying on overly simplifying assumptions, are more likely than grammar-based techniques to tell investigators something about user interactions that they don't already know. 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 events of interest, or to detect when actual usage patterns violate expected patterns. However, the value of generating a grammatical or FSM-based model to summarize use is more limited. 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 observed patterns of usage of an interface without indicating which are most common. While this may be useful in defining paths for UI regression testing, investigators interested in locating usability problems will more likely be interested in identifying and determining the frequency of specific observed patterns than in seeing a grammar to summarize them all.

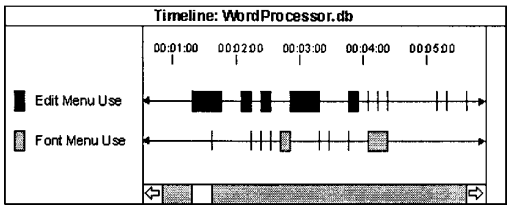


Fig. 12. Use of "Edit" menu operations is indicated in black. Use of "Font" menu operations is indicated in grey. Used to display results of transformations or sequence detection.

4.7 Visualization

4.7.1 Purpose. These techniques present the results of transformation and analysis in forms allowing humans to exploit their innate visual analysis capabilities to interpret results. Some of these techniques are helpful in linking results of analysis back to features of the interface.

4.7.2 Examples. Investigators have proposed a number of techniques to visualize data based on UI events. For a survey of such techniques see [Guzdial et al. 1994]. Below are few examples of techniques that have been used in support of the analysis approaches described above.

Transformation: The results of performing selection 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 12).

MacSHAPA [Sanderson and Fisher 1994] visualizes events as occupying cells in a spreadsheet. Event streams are listed vertically (in adjacent columns) and

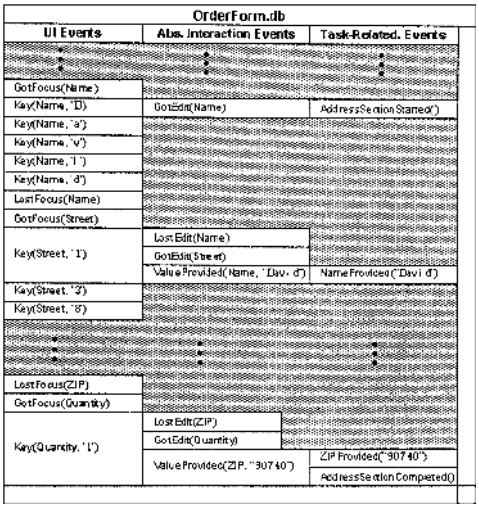


Fig. 13. Correspondence between events at different levels of abstraction is indicated by horizontal alignment. Single "Key" events in large cells correspond to "LostEdit," "GotEdit," and "ValueProvided" abstract interaction events in smaller, horizontally aligned cells.

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 to indicate abstraction relationships (Figure 13).

Counts and summary statistics: There are a number of visualizations that can 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.

The following are examples of static 2D and 3D graphs:

- Graph of keystrokes per window [Chang and Dillon 1997].
- Graph of mouse clicks per window [Chang and Dillon 1997].
- Graph of relative command frequencies [Kay and Thomas 1995] (Figure 14).
- Graph of relative command frequencies as they vary over time [Kay and Thomas 1995] (Figure 15).

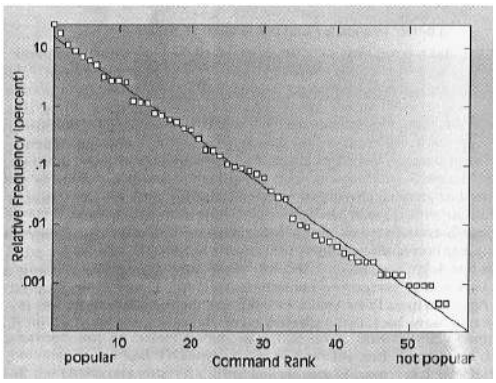


Fig. 14. Relative command frequencies ordered by "rank".

The following are examples of static 2D effects superimposed on an abstract coordinate space representing the interface:

- Location of mouse clicks [Guzdial et al. 1994; Chang and Dillon 1997].
- Mouse travel patterns between clicks [Buxton et al. 1983; Chang and Dillon 1997].

The following are examples of static and dynamic 2D and 3D effects superimposed on top of a graphical representation of the interface:

- Static highlighting to indicate location and density of mouse clicks [Guzdial et al. 1994].
- Dynamic highlighting of mouse click activity as it varies over time [Guzdial et al. 1994].
- 3D representation of mouse click location and density [Guzdial et al. 1994] (Figure 16).

Sequence detection: The same technique illustrated in Figure 12 can be used to visualize the results of selecting subsequences of UI events based on sequence detection techniques.

EDEM provides a dynamic visualization of the occurrence of UI events by highlighting nodes in a hierarchical representation of the user interface being monitored. A similar visualization is provided to indicate the occurrence of abstract events, defined in terms of abstract patterns of

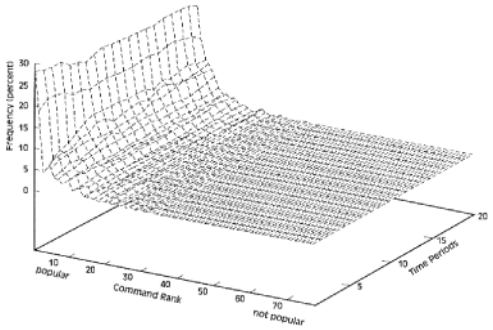


Fig. 15. Relative command frequencies over time.

events, by highlighting entries in a list of agents responsible for detecting those patterns. These visualizations help investigators inspect the dynamic behavior of events, thereby supporting the process of event pattern specification [Hilbert and Redmiles 1998a].

Sequence comparison: As described above, MacSHAPA provides facilities for aligning two sequences of events as optimally as possible and presenting the results visually as cells in adjacent spreadsheet columns [Sanderson and Fisher 1994]. Aligned events appear in the same row and missing cells indicate events in one sequence that could not be aligned with events in the other (Figure 17).

UsAGE provides a similar visualization for comparing sequences based on drawing a connected graph of nodes [Ueling and Wolf 1995]. 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 novice performed the actions. 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 uses a connected graph visualization to illustrate the results of his Markov-based analysis [Guzdial 1993]. The result is a process model with nodes representing process steps and arcs indicating the

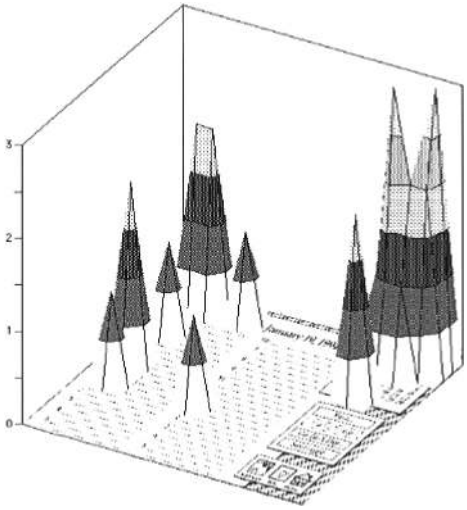


Fig. 16. A 3D representation of mouse click density superimposed over a graphical representation of the interface.

observed probabilities of transitions between process steps (Figure 18).

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 visual representations of the interface can be particularly problematic.

4.8 Integrated Support

4.8.1 Purpose. Environments that facilitate flexible composition of various

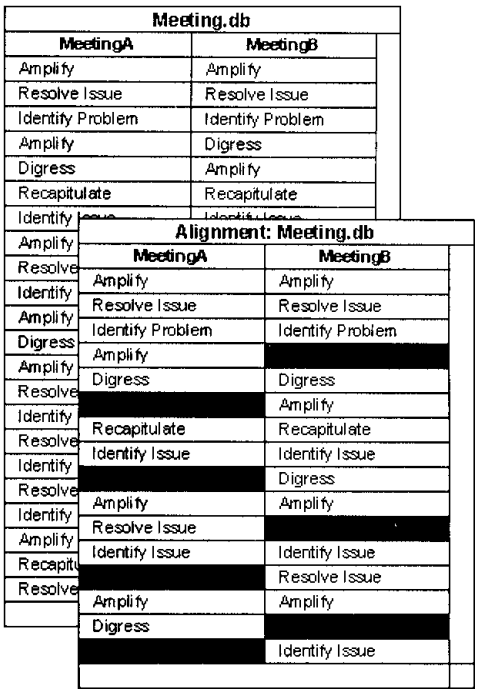


Fig. 17. Results of an automatic alignment of two separate event streams. Horizontal alignment indicates correspondence. Black spaces indicate where alignment was not possible.

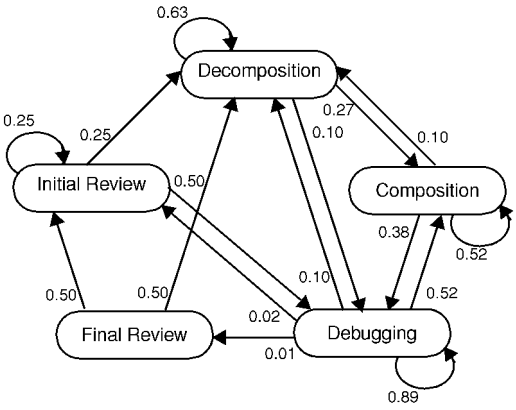


Fig. 18. A process model characterizing user behavior with nodes representing process steps and arcs indicating observed probabilities of transitions between process steps.

transformation, analysis, and visualization capabilities provide integrated support. Some environments also provide built-in support for managing domain-specific artifacts such as evaluations, subjects, tasks, data and results of analysis.

4.8.2 Examples. MacSHAPA is perhaps the most comprehensive environment designed to support all manner of exploratory sequential data analysis (ESDA) [Sanderson et al. 1994]. Features include: data import and export; video and coded observation synch and search capabilities; facilities for performing selection, 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 built-in visualizations and reports.

DRUM 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 [Macleod et al. 1993].

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

Finally, ErgoLight Operation Recording Suite (EORS) and Usability Validation Suite (EUVS) [ErgoLight Usability Software 1998] offer a number of facilities for managing usability evaluations, both local and remote, as well as facilities for merging data from multiple users.

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

Column Label	Key to Column Values
Event Capture	(Yes) = events captured automatically; (Instr) = application must be hand-instrumented; (Sim) = events captured by command line simulation.
Synch/Search	(Obs) = events synchronized with coded observations; (Vid) = events synchronized with video.
Use of Context	(UI) = the UI can be queried for contextual information; (App) = the application can be queried. (User) = the user provides contextual info.
Select/Recode through Sequence Detect	(Built-in) = built-in selection/abstraction/counts & stats/detection; (User) = user selects/abstracts events; (Model) = abstract model used to select/abstract/detect; (Script) = scripts used; (Program) = programs used; (Database) = database query & manipulation language used.
Sequence Compare	(Concrete) = source sequence compared against concrete target sequence; (Model) = source sequence compared against abstract model.
Sequence Character	(Manual) = abstract model constructed manually using statistical/grammatical techniques; (Auto) = abstract model generated automatically.
Visualize	(Built-in) = built-in visualizations; (Graphs) = use of standard graphing facilities; (Database) = use of database graphing facilities.
Data Manage	(Built-in) = built-in management of domain-specific artifacts.

Fig. 19. A key to interpreting the values listed in columns of Table III.

can significantly reduce the burden of data management and integration.

4.8.4 Limitations. Most of the environments above possess 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. As a result, it lacks support for event capture and focuses primarily on analysis techniques that, when applied to UI events, require extensive human involvement and interpretation. MacSHAPA provides many of the basic building blocks required for an "ideal" environment for capturing and analyzing UI events, however, selection, abstraction, and recoding cannot be easily automated. Furthermore, because the powerful features of MacSHAPA cannot be used during event collection, contextual information that might be useful in selection and abstraction is not available.

While providing features for managing and analyzing UI events, coded observations, video data, and evaluation artifacts, DRUM does not provide features for selecting, abstracting, and recoding data.

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

5. DISCUSSION

5.1 Summary of the State of the Art

Synch and search techniques are among the most mature technologies for exploiting UI event data in usability evaluations. Tools supporting these techniques are becoming increasingly common in usability labs. However, these techniques can be costly in terms of equipment, human observers, and data storage and analysis requirements. Furthermore, synch and search techniques generally exploit UI events as no more than convenient indices into video recordings. In some cases, events may be used as the basis for computing simple counts and summary statistics using spreadsheets or statistical packages. However, such analyses typically require investigators to perform selection and abstraction by hand.

The other, arguably more sophisticated, analysis techniques such as sequence detection, comparison, and characterization continue to remain denizens of the research lab for the most part. Those that are most compelling tend to require the most human intervention, interpretation, and effort (e.g., exploratory sequential data analysis techniques and the Markov- and Grammar-based sequence characterization techniques). Those that are most

automated tend to be least compelling and most unrealistic in their assumptions (e.g., ADAM, UsAGE, and EMA). One of the main problems limiting the success of automated approaches may be their lack of focus on transformation, which appears to be a necessary prerequisite for meaningful analysis (for reasons articulated in Section 3 and discussed further below).

Nevertheless, few investigators have attempted to address the problem of transformation realistically. Of the twenty-five plus approaches surveyed here, only a handful provide mechanisms that allow investigators to perform transformations at all (Microsoft, SunSoft, Apple, Chen, CHIME, EDEM, Hawk, and MacSHAPA). Of those, fewer still allow models to be constructed and reused in an automated fashion (CHIME, EDEM, and Hawk). Of those, fewer still allow transformation to be performed in context so that important contextual information can be used in selection and abstraction (EDEM).

5.2 Some Anticipated Challenges

There is very little data published regarding the relative utility of the surveyed approaches in supporting usability evaluations. As a result, we have focused on the technical capabilities of the surveyed approaches in order to classify, compare, and evaluate them. To take this analytical evaluation a step further: our understanding of the nature of UI events (based on extensive Java, Windows, and X Windows programming experience) leads us to conclude that more work will likely be needed in the area of transforming the "raw" data generated by such event-based systems in preparation for other types of analysis in order to increase the likelihood of useful results. This is because most other types of analysis (including simple counts and summary statistics as well as sequence analysis techniques) are sensitive to lexical-level differences in event streams that can be removed via transformation, as illustrated in Section 3.2.

There are a number of ways that investigators have successfully side-stepped the transformation problem. For instance, building data collection directly into a user interface management system or requiring applications to report events themselves can help ameliorate some of the issues. However, both of these approaches have important limitations.

User interface management systems (UIMSs) typically model the relationships between application features and UI events explicitly, so reasonable data collection and analysis mechanisms can be built directly in, as in the case of the MIKE UIMS [Olsen and Halversen 1988], KRI/AG [Lowgren and Nordqvist 1992], and UsAGE [Ueling and Wolf 1995]. Because UIMSs have dynamic access to most aspects of the user interface, contextual information useful in interpreting the significance of events is also available. However, many developers do not use UIMSs, thus, a more general technique that does not presuppose the use of a UIMS is needed.

Techniques that allow applications to report events directly via an event-reporting API provide a useful service, particularly in cases where events of interest cannot be inferred from UI events. This allows important application-specific events to be reported by applications themselves and provides a more general solution than a UIMS-based approach. However, this places an increased burden on application developers to capture and transform events of interest, for example, as in Kay and Thomas [1995] and Balbo [1996]. This can be costly, particularly if there is no centralized command dispatch loop, or similar mechanism, that can be tapped as a source of application events. This also complicates software evolution since data collection code is typically intermingled with application code. Furthermore, there is much usability-related information not typically processed by applications that can be easily captured by tapping into the UI event stream, for instance, shifts in input focus, mouse movements, and the specific user interface

actions used to invoke application features. As a result, an event-reporting API is just part of a more comprehensive solution.

Thus, we conclude that more work is needed in the area of transformation and data collection to ensure that useful information can be captured in the first place, before automated analysis techniques, such as those surveyed above, can be expected to yield meaningful results (where "meaningful" means the results can be related, without undue hardship, to aspects of the user interface and application being studied as well as users' actions at higher levels of abstraction than simple key presses and mouse clicks). A reasonable approach would assume no more than a typical event-based user interface system, such as provided by the Macintosh Operating System, Microsoft Windows, X Window System, or Java Abstract Window Toolkit, and developers would not be required to adopt a particular UIMS nor call an API to report every potentially interesting event.

5.3 Related Work and Future Directions

There are a number of related techniques that have been explored, both in academia and industry, that have the potential of providing useful insights into how to more effectively exploit UI events as a source of usability information.

A number of researchers and practitioners have addressed related issues in capturing and evaluating event data in the realm of software testing and debugging:

- Work in distributed event monitoring, e.g., GEM [Mansouri-Samani and Sloman 1994], and model-based testing and debugging, e.g., EBBA [Bates 1995] and TSL [Rosenblum 1991], have addressed a number of problems in the specification and detection of composite events and the use of context in interpreting the significance of events. The event specification notations, infrastructure, and experience that have come out of this work might provide useful insights that can be applied to the problem of capturing and analyzing UI event data.
- Automated user interface testing techniques, e.g., WinRunner™ [Mercury Interactive 1998] and JavaStar™ [Sun Microsystems 1998], are faced with the problem of robustly identifying user interface components in the face of user interface change, and evaluating events against specifications of expected UI behavior in test scripts. The same problem is faced in maintaining the relationships between UI components and higher-level specifications of application features and abstract events of interest in usability evaluations based on UI events.
- Monitoring of application programmatic interfaces (APIs), e.g., Hewlett Packard's Application Response-time Measurement API [Hewlett Packard 1998], addresses the problem of monitoring API usage to help software developers evaluate the fit between the design of an API and how it is actually used. Insights gained in this area may generalize to the problem of monitoring UI usage to evaluate the fit between the design of a UI and how it is actually used.
- Internet-based application monitoring systems, e.g., AppScope™ [Aqueduct Software 1998] and Talkback™ [Full Circle Software 1998], have begun to address issues of collecting application failure data on a potentially large and ongoing basis over the Internet. The techniques developed to make this practical for application failure monitoring could be applicable in the domain of large-scale, ongoing collection of user interaction data over the Internet.

A number of researchers have addressed problems in the area of mapping between lower level events and higher level events of interest:

- Work in the area of event histories, e.g., Kosbie and Myers [1994], and undo mechanisms has addressed issues involved in grouping lower level UI events into more meaningful units from the point of view of users' tasks. Insights

gained from this work, and the actual event representations used to support undo mechanisms, might be exploited to capture events at higher levels of abstraction than are typically available at the window system level.

- Work in the area of user modeling [User Modeling 1998] is faced with the problem of inferring users' tasks and goals based on user background, interaction history, and current context in order to enhance human-computer interaction. The techniques developed in this area, which range from rule-based to statistically-oriented machine-learning techniques, might eventually be harnessed to infer higher level events from lower level events in support of usability evaluations based on UI events.
- Work in the area of programming by demonstration [Cypher 1993] and plan recognition and assisted completion [Cypher 1991] also addresses problems involved in inferring user intent based on lower level interactions. This work has shown that such inference is feasible in at least some structured and limited domains, and programming by demonstration appears to be a desirable method for specifying expected or unexpected patterns of events for sequence detection and comparison purposes.
- Layered protocol models of interaction, e.g., Nielsen [1986] and Taylor [1988a; 1988b], allow human-computer interactions to be modeled at multiple levels of abstraction. Such techniques might be useful in specifying how higher level events are to be inferred based on lower level events. Command language grammars (CLGs) [Moran 1981] and task-action grammars (TAGs) [Payne and Green 1986] are other potentially useful modeling techniques for specifying relationships between human-computer interactions and users' tasks and goals.

Work in the area of automated discovery and validation of patterns in large corpora of event data might also provide valuable insights:

- Data mining techniques for discovering association rules, sequential patterns, and time-series similarities in large data sets [Agrawal et al. 1996] may be applicable in uncovering patterns relevant to investigators interested in evaluating usage and usability based on UI events.
- The process discovery techniques investigated by Cook and Wolf [1996] provide insights into problems involved in automatically generating models to characterize the sequential structure of event traces
- The process validation techniques investigated by Cook and Wolf [1997] provide insights into problems involved in comparing traces of events against models of expected behavior.

Finally, there are numerous domains in which event monitoring has been used as a means of identifying and, in some cases, diagnosing and repairing breakdowns in the operation of complex systems. For example:

- Network and enterprise management tools for automating network and application administration, e.g., TIBCO Hawk™ [TIBCO 1998].
- Product condition monitoring, e.g., high-end photocopiers or medical devices that report data back to equipment manufacturers to allow performance, failures, and maintenance issues to be tracked remotely [Lee 1996].

6. CONCLUSIONS

We have surveyed a number of computer-aided techniques for extracting usability-related information from UI events. Our 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

Table IV. Index into the References Based on the Categories Established by the Comparison Framework

Category	Approaches
Synchronization and Searching	Playback [Neal & Simmons 1983), Apple [Weiler 1993], SunSoft [Weiler 1993], Microsoft [Hoiem & Sullivan 1994], I-Observe [Badre et al. 1995]
Transformation	Incident Monitoring [Chen 1990], User-Identified CIs [Hartson et al. 1996], CHIME [Badre & Santos 1991], EDEM [Hilbert & Redmiles 1997]
Counts and Summary Statistics	UIMS [Buxton et al. 1983], MIKE [Olsen & Halversen 1988], KRI/AG [Lowgren & Nordqvist 1992], Long-Term Monitoring [Kay & Thomas 1995], AUS [Chang & Dillon 1997], EORS & EUVS [ErgoLight Usability Software 1998]. *Related:* AppScope [Aqueduct Software 1998], Talkback [Full Circle Software 1998]
Sequence Detection	LSA [Sackett 1978], Fisher's Cycles [Fisher 1988], TOP/G [Hoppe 1988], MRP [Siochi & Hix 1991], Expectation Agents [Girgensohn et al. 1994], EDEM [Hilbert & Redmiles 1997], USINE [Lecerof & Paterno 1998]. *Related:* TSL [Rosenblum 1991], Amadeus [Selby et al. 1991], YEAST [Krishnamurthy & Rosenblum 1995], EBBA [Bates 1995], GEM [Mansouri-Samani & Sloman 1997]
Sequence Comparison	ADAM [Finlay & Harrison 1990], USAGE [Ueling & Wolf 1995], EMA [Balbo 1996], USINE [Lecerof & Paterno 1998], EUVS [ErgoLight Usability Software 1998]. *Related:* Process Validation [Cook & Wolf 1997]
Sequence Characterization	Markov-based [Guzdial 1993], Grammar-based [Olson et al. 1994]. *Related:* Process Discovery [Cook & Wolf 1995]
Integrated Support	MacSHAPA [Sanderson et al. 1994], DRUM [Macleod & Rengger 1993]. Hawk [Guzdial 1993], EORS & EUVS [ErgoLight Usability Software 1998].

techniques; and finally, techniques that provide integrated evaluation support.

Very few of the surveyed approaches support transformation, which we argue is a critical subprocess in the overall process of extracting meaningful usability-related information from UI events.

Our current research involves exploring techniques and infrastructure for performing transformation and analysis automatically and in context in order to greatly reduce the amount of data that must ultimately 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, we believe that automated techniques, such as those surveyed here, will be useful in capturing indicators of the "big picture" regarding application use in the field. However, we believe that such techniques may be less suited to identifying subtle, nuanced usability issues. Fortunately, these strengths and weaknesses nicely complement the strengths and weaknesses inherent in current usability testing practice, in which subtle usability issues are identified through careful human observation, but in which there is little sense of the "big picture" of how applications are used on a large scale.

A usability professional from a large software development organization recently reported to us that the usability team is often approached by design and development team members with questions such as "how often do users do X?" or "how often does Y happen?" This is obviously useful information for developers wishing to assess the impact of suspected problems or to focus development effort for the next version. However, it is not information that can be reliably collected in the usability lab. We believe that automated usage data collection techniques will eventually complement traditional usability evaluation practice, not only by supporting developers as described above, but also in helping assess the impact of, and focusing the efforts of, usability evaluations.

REFERENCES

ABBOTT, A. 1990. A primer on sequence methods. *Organ. Sci. 4.*

AGRAWAL, R., ARNING, A., BOLLINGER, T., MEHTA, M., SHAFER, J., AND SRIKANT, R. 1996. The Quest data mining system. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*.

AHO, A. V., KERNIGHAN, B. W., AND WEINBERGER, P. J. 1988. *The AWK Programming Language*. Addison-Wesley, Reading, MA.

ALLISON, P. D. AND LIKER, J. K., 1982. Analyzing sequential categorical data on dyadic interaction: A comment on Gottman. *Psychological Bulletin 2*.

AQUEDUCT SOFTWARE. 1998. AppScope Web Pages. URL: http://www.aqueduct.com/.

BADRE, A. N. AND SANTOS, P. J. 1991a. CHIME: A Knowledge-Based Computer-Human Interaction Monitoring Engine. Tech Rept. GIT-GVU-91-06.

BADRE, A. N. AND SANTOS, P. J. 1991b. A Knowledge-Based System for Capturing Human-Computer Interaction Events: CHIME. Tech. Rept. GIT-GVU-91-21.

BADRE, A. N., GUZDIAL, M., HUDSON, S. E., AND SANTOS, P. J. 1995. A user interface evaluation environment using synchronized video, visualizations, and event trace data. *J. of Software Qual. 4*.

BAECKER, R. M, GRUDIN, J., BUXTON, W. A. S., AND GREENBERG, S., Eds. 1995. *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann, San Mateo, CA.

BALBO, S. 1996. EMA: Automatic Analysis Mechanism for the Ergonomic Evaluation of User Interfaces. CSIRO Tech. rep.

BATES, P. C. 1995. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Trans. on Comput. Syst. 13*, 1.

BELLOTTI, V. 1990. A framework for assessing applicability of HCI techniques. In *Proceedings of INTERACT '90*.

BUXTON, W., LAMB, M., SHEMAN, D., AND SMITH, K. 1983. Towards a comprehensive user interface management system. In *Proceedings of SIGGRAPH '83*.

CHANG, E. AND DILLON, T. S. Automated usability testing. In *Proceedings of INTERACT '97*.

CHEN, J. 1990. Providing intrinsic support for user interface monitoring. In *Proceedings of INTERACT '90*.

COOK, J. E. AND WOLF, A. L. 1994. Toward metrics for process validation. In *Proceedings of ICSP '94*.

COOK, J. E. AND WOLF, A. L. 1995. Automating process discovery through event-data analysis. In *Proceedings of ICSE '95*.

COOK, J. E. AND WOLF, A. L. 1997. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. Tech. Rep. CU-CS-840-97, Dept. of Computer Science, Univ. of Colorado at Boulder.

COOK, R., KAY, J., RYAN, G., AND THOMAS, R. C. 1995. A toolkit for appraising the long-term usability of a text editor. *Software Qual. J. 4*, 2.

CUOMO, D. L. 1994. Understanding the applicability of sequential data analysis techniques for analysing usability data. In *Usability Laboratories Special Issue of Behavior and Information Technology*, J. Nielsen, Ed., vol. 13, no.1 & 2.

CYPHER, A. 1991. Eager: programming repetitive tasks by example. In *Proceedings of CHI '91*.

CYPHER, A., Ed. 1993. *Watch what I do: Programming by Demonstration*. MIT Press, Cambridge MA.

DODGE, M. AND STINSON, C. 1999. *Running Microsoft Excel 2000*. Microsoft Press.

DOUBLEDAY, A., RYAN, M., SPRINGETT, M., AND SUTCLIFFE, A. 1997. A comparison of usability techniques for evaluating design. In *Proceedings of DIS '97*.

ELGIN, B. 1995. Subjective usability feedback from the field over a network. In *Proceedings of CHI '95*.

ERGOLIGHT USABILITY SOFTWARE. 1998. Operation Recording Suite (EORS) and Usability Validation Suite (EUVS) Web pages. URL: http://www.ergolight.co.il/.

FARAONE, S. V. AND DORFMAN, D. D. 1987. Lag sequential analysis: Robust statistical methods. *Psychological Bulletin 101*.

FEATHER, M. S., NARAYANASWAMY, K., COHEN, D., AND FICKAS, S. 1997. Automatic monitoring of software requirements. Research Demonstration. In *Proceedings of ICSE '97*.

FICKAS, S. AND FEATHER, M. S. 1995. Requirements monitoring in dynamic environments. In *IEEE International Symposium on Requirements Engineering*.

FINLAY, J. AND HARRISON, M. 1990. Pattern recognition and interaction models. In *Proceedings of INTERACT '90*.

FISHER, C. 1987. Advancing the study of programming with computer-aided protocol analysis. In *Empirical Studies of Programmers, 1987 Workshop*, G. Olson, E. Soloway, and S. Sheppard, Eds. Ablex, Norwood, NJ.

FISHER, C. 1991. Protocol Analyst's Workbench: Design and Evaluation of Computer-Aided Protocol Analysis. Unpublished Ph.D. thesis, Carnegie Mellon University, Dept. of Psychology, Pittsburgh, PA.

FISHER, C. AND SANDERSON, P. 1996. Exploratory sequential data analysis: Exploring continuous observational data. *Interactions 3*, 2, ACM Press.

FITTS, P. M. 1964. Perceptual motor skill learning. In *Categories of human learning*, A. W. Melton, Ed. Academic Press, New York, NY.

FULL CIRCLE SOFTWARE. 1998. Talkback Web pages. URL: http://www.fullsoft.com/.

GIRGENSOHN, A., REDMILES, D. F., AND SHIPMAN, F. M. III. 1994. 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.

GOODMAN, D. 1998. *Complete HyperCard 2.2 Handbook*. ToExcel.

GOTTMAN, J. M. AND ROY, A. K. 1990. *Sequential analysis: A guide for behavioral researchers*. Cambridge University Press, Cambridge, England.

GRUDIN, J. 1992. Utility and usability: Research issues and development contexts. *Interacting with comput. 4*, 2.

GUZDIAL, M. 1993. Deriving Software Usage Patterns from Log Files. Tech. Rept. GIT-GVU-93–41.

GUZDIAL, M, WALTON, C., KONEMANN, M., AND SOLOWAY, E. 1993. Characterizing Process Change Using Log File Data. Tech. Rep. GIT-GVU-93–44.

GUZDIAL, M., SANTOS, P., BADRE, A., HUDSON, S., AND GRAY, M. 1994. Analyzing and visualizing log files: A computational science of usability. In *Presented at HCI Consortium Workshop*.

HARTSON, H. R., CASTILLO, J. C., KELSO, J., AND NEALE, W. C. 1996. Remote evaluation: The network as an extension of the usability laboratory. In *Proceedings of CHI '96*.

HELANDER, M., Ed. 1998. *Handbook of human-computer interaction*. Elsevier Science Publishers B.V., North Holland.

HEWLETT PACKARD. 1998. Application Response Measurement API. URL: http://www.hp.com/openview/rpm/arm/.

HILBERT, D. M. AND REDMILES, D. F. 1998a. An approach to large-scale collection of application usage data over the Internet. In *Proceedings of ICSE '98*.

HILBERT, D. M. AND REDMILES, D. F. 1998b. Agents for collecting application usage data over the Internet. In *Proceedings of Autonomous Agents '98*.

HILBERT, D. M., ROBBINS, J. E., AND REDMILES, D. F., 1997. Supporting Ongoing User Involvement in Development via Expectation-Driven Event Monitoring. Tech Report UCI-ICS-97-19, Dept. of Information and Computer Science, Univ. of California, Irvine.

HIRSCHBERG, D. S. 1975. A linear space algorithm for computing maximal common subsequences. *Commun of the ACM 18*.

HOIEM, D. E. AND SULLIVAN, K. D. 1994. Designing and using integrated data collection and analysis tools: challenges and considerations. In *Usability Laboratories Special Issue of Behavior and Information Technology*, J. Nielsen, Ed., vol. 13, no. 1 & 2.

HOPPE, H. U. 1988. Task-oriented parsing: A diagnostic method to be used by adaptive systems. In *Proceedings of CHI '88*.

JOHN, B. E. AND KIERAS, D. E. 1996a. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Trans. on Comput.–Hum. Interaction 3*, 4.

JOHN, B. E. AND KIERAS, D. E. 1996b. Using GOMS for user interface design and evaluation: which technique? *ACM Trans. on Comput.–Hum. Interaction 3*, 4.

KAY, J. AND THOMAS, R. C. 1995. Studying long-term system use. *Commun. of the ACM 38*, 7.

KOSBIE, D. S. AND MYERS, B. A. 1994. Extending programming by demonstration with hierarchical event histories. In *Proceedings of East-West Human Computer Interaction '94*.

KRISHNAMURTHY, B. AND ROSENBLUM, D. S. 1995. Yeast: A general purpose event-action system. *IEEE Trans. on Software Eng. 21*, 10.

LECEROF, A. AND PATERNO, F. 1998. Automatic support for usability evaluation. *IEEE Trans. on Software Engin. 24*, 10.

LEE, B. 1996. Remote diagnostics and product lifecycle monitoring for high-end appliances: a new Internet-based approach utilizing intelligent software agents. In *Proceedings of the Appliance Manufacturer Conference*.

LEWIS, R. AND STONE, M., Ed. 1999. *Mac OS in a Nutshell*. O'Reilly and Associates.

LOWGREN, J. AND NORDQVIST, T. 1992. Knowledge-based evaluation as design support for graphical user interfaces. In *Proceedings of CHI '92*.

MACLEOD, M. AND RENGGER, R. 1993. The Development of DRUM: A Software Tool for Video-assisted Usability Evaluation. In *Proceedings of HCI '93*.

MANSOURI-SAMANI, M. AND SLOMAN, M. 1997. GEM: A generalized event monitoring language for distributed systems. *IEE/BCS/IOP Distributed Syst. Eng. J. 4*, 2.

MERCURY INTERACTIVE. 1998. WinRunner and XRunner Web Pages. URL: http://www.merc-int.com/.

MORAN, T. P. 1981. The command language grammar: A representation for the user interface of interactive computer systems. *Int. J. of Man–Machine Studies*, 15.

NEAL, A. S. AND SIMONS, R. M. PLAYBACK: A method for evaluating the usability of software and its documentation. In *Proceedings of CHI '83*. 1983.

NIELSEN, J. 1986. A virtual protocol model for computer-human interaction. *Int. J. of Man–Machine Studies 24*.

NIELSEN, J. 1993. *Usability engineering*. Academic Press/AP Professional, Cambridge, MA.

NIELSEN, J. AND MACK, R. L., Eds. 1994. Usability inspection methods. Wiley, New York.

NYE, A. AND O'REILLY, T. 1992. *X Toolkit Intrinsics Programming Manual for X11, Release 5*. O'Reilly and Associates.

OLSEN, D. R. AND HALVERSEN, B. W. 1988. Interface usage measurements in a user interface management system. In *Proceedings of UIST '88*.

OLSON, G. M., HERBSLEB, J. D., AND RUETER, H. H. 1994. Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Hum.–Comput. Interaction Special Issue on ESDA*, Vol. 9.

PAYNE, S. G. AND GREEN, T. R. G. 1986. Task-action grammars: A model of the mental representation of task languages. *Hum.-Comput. Interaction*, Vol. 2.

PENTLAND, B. T. 1994. A grammatical model of organizational routines. *Administrative Sci. Quarterly*.

PENTLAND, B. T. 1994. Grammatical models of organizational processes. *Organ. Sci*.

PETZOLD, C. 1998. Programming Windows. Microsoft Press.

POLSON, P. G., LEWIS, C., RIEMAN, J., AND WHARTON, C. 1992. Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *Int. J. Man-Mach. Studies 36*, 5, 741–773.

PREECE, J., ROGERS, Y., SHARP, H., BENYON, D., HOLLAND, S., AND CAREY, T. 1994. *Human-computer interaction*. Addison-Wesley, Wokingham, UK.

ROSENBLUM, D. S. 1991. Specifying concurrent systems with TSL. *IEEE Software, 8*, No. 3.

RUBIN, C. 1999. *Running Microsoft Word 2000*. Microsoft Press.

SACKETT, G. P. 1978. Observing Behavior, Vol. 2. University Park Press, Baltimore, MD.

SANDERSON, P. M. AND FISHER, C. 1994. Exploratory sequential data analysis: foundations. *Hum.–Comput. Interaction Special Issue on ESDA, 9*.

SANDERSON, P. M., SCOTT, J. J. P, JOHNSTON, T., MAINZER, J., WATANABE, L. M., AND JAMES, J. M. 1994. MacSHAPA and the enterprise of Exploratory Sequential Data Analysis (ESDA). *International J. of Hum.–Comput. Studies, 41*.

SANTOS, P. J. AND BADRE, A. N. 1994. 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.

SCHIELE, F. AND HOPPE, H. U. 1990. Inferring task structures from interaction protocols. In *Proceedings of INTERACT '90*.

SELBY, R. W., PORTER, A. A., SCHMIDT, D. C., AND BERNEY, J. 1991. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of ICSE '91*.

SIOCHI, A. C. AND EHRICH, R. W. 1991. Computer analysis of user interfaces based on repetition in transcripts of user sessions. *ACM Trans. on Inf. Syst.*

SIOCHI, A. C. AND HIX, D. 1991. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *Proceedings of CHI '91*.

SMILOWITZ, E. D., DARNELL, M. J., AND BENSON, A. E. 1994. Are we overlooking some usability testing methods? A comparison of lab, beta, and forum tests. In *Usability Laboratories Special Issue of Behavior and Information Technology*, J. Nielsen, Ed., Vol. 13, No. 1 & 2.

SUN MICROSYSTEMS. 1998. SunTest JavaStar Web Pages. URL: http://www.sun.com/suntest/.

SWEENY, M., MAGUIRE, M., AND SHACKEL, B. 1993. Evaluating human-computer interaction: A framework. *Int. J. of Man–Machine Stud., 38*.

TAYLOR, M. M. 1988a. Layered protocols for computer-human dialogue I: Principles. *Int. J. of Man–Machine Stud. 28*.

TAYLOR, M. M. 1988b. Layered protocols for computer-human dialogue II: Some practical issues. *Int. J. of Man–Machine Stud. 28*.

TAYLOR, R. N. AND COUTAZ, J. 1994. workshop on software engineering and human-computer interaction: Joint research issues. In *Proceedings of ICSE '94*.

TIBCO, 1998. HAWK Enterprise Monitor Web Pages. URL: http://www.tibco.com/.

UEHLING, D. L. AND WOLF, K. 1995. User Action Graphing Effort (UsAGE). In *Proceedings of CHI '95*.

USER MODELING INC. (UM Inc.), 1998. Home Page. URL: http://um.org/.

WEILER, P. 1993. Software for the usability lab: a sampling of current tools. In *Proceedings of INTERCHI '93*.

WHITEFIELD, A., WILSON, F., AND DOWELL, J. 1991. A framework for human factors evaluation. *Behav. and Inf. Technol., 10*, 1.

WOLF, A. L. AND ROSENBLUM, D. S. 1993. A study in software process data capture and analysis. In *Proceedings of the Second International Conference on Software Process*.

ZUKOWSKI, J. AND LOUKIDES, M., Ed. 1997. *Java Awt Reference*. O'Reilly and Associates.

Received October 1998; revised July 1999; accepted March 2000