CHIME: A Knowledge-Based Computer-Human Interaction Monitoring Engine

by

Albert N. Badre
Paulo Santos

GIT-GVU-91-06
July 1991

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

Albert N. Badre
Paulo J. Santos

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

E-mail: badre@cc.gatech.edu
pas@cc.gatech.edu

Abstract

One of the prerequisites to improving the quality of a user interface is the ability to capture and analyze human-computer interaction data. We claim that in order to perform an accurate and extensive analysis of human-computer interaction events, an automatic mechanism has to be used which attends to both the syntax and semantics of interaction. In this paper, we describe such a data capturing tool, and a model of interaction which it uses. We address issues of interference with user work, semantics of the interface, data collection, and integration of the monitor in a UIMS. We claim that the more knowledge a monitor has about the semantics of an application the more useful the data it collects.

Keywords
User Interfaces, Interactive Techniques, Automated Monitoring, Human-Computer Interaction

Introduction

One of the steps in the iterative design of quality human-computer interfaces is the collection and analysis of interface utilization data. Tools and techniques are required to perform this phase of the interface development cycle. Among the techniques that have been used to capture human–computer interaction information are user feedback [Coleman 85], annotation by experienced transcribers [Neal 83], videotaping of interaction sessions [Mackay 88], automated transcription [Good 85] and UIMS-supported transcription [Olsen 1988].

All of these techniques present some drawbacks. User feedback, obtained through questionnaires [Root 83], interviews [Lief 85], analysis of customer support activity or protocol analysis [Rubinstein 84], actually transfers the responsibility of usability evaluation to someone – the user – that is not necessarily a usability expert. This technique also does not allow for objective evaluation of an interface based on performance studies, but rather it limits its evaluation to user satisfaction and detection of errors.

Human annotators, no matter how experienced and proficient they are, can not accurately record all the information about the interaction. Many analysis techniques use time-based metrics [Card 80, Barber 83]. It is unlikely that human annotators can accurately record detailed timing information. Furthermore, they can not record every action the user takes, simply because such recording will often require more time than it takes to perform the actions being recorded. Finally, annotators often intrude into the user’s space in field situations where it is desirable to capture longitudinal data unobtrusively.

Videotaping of sessions has some advantages over human annotators, but it still presents some serious drawbacks. Although it allows for detailed timestamping and transcribing by playing the recorded session in slow motion, the transcription task may be very time consuming and expensive, especially for long term usage. Video cameras are also intrusive, and just their presence may intimidate the user, thus affecting the quality of the field data collected.

Automated transcription overcomes some of the problems associated with the preceding methods, by having the computers gather data about themselves. However, a bare
automatic transcription has three major disadvantages. First, the amounts of raw data generated can be very large and the evaluator can be overwhelmed, requiring the use of elaborate analysis tools [Siochi 89 and 91]. Secondly, when the recorded information is based exclusively on system level actions such as key presses, the semantics of the interaction can not be inferred and recorded. A third disadvantage is that this technique can not record aspects of the interaction that are not communicated to the computer, such as user satisfaction or environment conditions and perturbations.

UIMS-supported interaction data collection minimizes the problems associated with automated transcribing, mainly by allowing for the incorporation of knowledge about the interface into the transcribing process. The information recorded may be of a higher level, and carry some semantic information specific to the interface. However, three problems still daunt the UIMS approach. First, this approach requires the program to be running under the control of a UIMS runtime system, and that runtime system would have to be augmented with recording facilities, thus increasing the size and complexity of every interface application generated. Secondly, the UIMS approach can not be used with existing interfaces, built without the help of the UIMS. And finally, like in the automated transcription case, it totally disregards actions and conditions not directly affecting the computer.

To overcome the difficulties with the traditional methods of human–computer interaction data capture, we have developed a new technique and supporting tools. Our technique is based on an underlying model to overcome the two problems of the amounts of data collected and the abstraction levels. The result is a knowledge-based monitoring system, whose central tool is CHIME, the Computer–Human Interaction Monitoring Engine.

The technique we developed provides an increased functionality over the UIMS-based approach. It is also possible to integrate it into existing or future User Interface Management Systems. We have addressed this issue of integration in UIMSs, in order to provide them with the capabilities for true and full iterative design.

**Overview of the Monitoring Process**

The iterative design model for the development of user interfaces (Figure 1) is strongly supported in the HCI research and development community [Gould 85].

Most UIMS research and development has been centered around providing tools for an easier, more productive and more consistent development and execution of user interfaces. An improvement in the quality of user interfaces has generally been achieved by UIMSs by imposing interface design restrictions on the designer. A major caveat in this approach is that consistency, robustness and reliability are achieved through a limitation on the degrees of freedom in interface design.

However, quality in user interfaces through support of iterative design is not incorporated into most UIMSs. MIKE [Olsen 86] is a notable exception. Other UIMSs have the capabilities to support the full iterative cycle, but the full implementation has not been achieved.

![Figure 1 - Iterative Interface Design Model](image-url)
In conceptualizing CHIME, to assess interface quality, the strategy was to incorporate the following key desirable characteristics of automatic knowledge-based monitoring.

First, automatic monitoring improves the accuracy of the data collected. Every user action, at every level, can be monitored and recorded in detail. In CHIME, timestamps can be attached to every action. Screen coordinates of user actions, subtle mouse movements or trajectories, hesitations, and similar very low level and detailed actions can be detected and recorded. Collecting this data with accuracy would be extremely hard or even impossible using exclusively more traditional recording techniques such as human observation or videotaping.

A second important feature is the ability of a monitor to distinguish human-computer interaction events that are relevant for exploring the validity of some hypothesized conclusion from events that are irrelevant for this purpose and thus need not be collected. CHIME incorporates this feature. When the monitor is given a description of the tasks and subtasks involved in an interaction, and also the tasks relevant for some analytical study, it can derive the list of elementary tasks that need to be gathered and those that need to be recorded for the purpose of the intended analysis. For example, let us suppose that we are interested in analyzing the way in which users perform the tasks of Cut/Copy-Paste. Let us also suppose that those tasks can be performed by selecting items from a menu of using keyboard accelerators. If a monitor is notified by the analyst that one is only interested in these three commands, it should be able to derive the relevant elementary events from the description of the interface. In this case, the monitor would only capture and parse the relevant events.

A third characteristic incorporated in CHIME is the ability of the monitor to capture data related to the user's level of expertise. The ways in which users execute actions on the computer are dependent on their level of expertise and familiarity with the interface and the model of the software at hand. In particular, the way in which they perform chunking of concepts is clearly dependent on these factors [Badre 82, Smelcer 86, Cooper 87]. Based on these and other studies on chunking, we hypothesize that action chunk size is associated with expertise level, and may be detected by automatic time monitoring of pauses between actions. Preliminary experimentation has confirmed this hypothesis.

A fourth desirable feature of automated monitoring is the capability to perform remote, independent monitoring. The MIKE [Olsen 86] User Interface Management System contains a Metric Collector to perform data collection, which is then passed on to an evaluator for analysis. However, it is only possible to collect data on applications built with and running under MIKE. While a UMIS with a data capture tool makes the UMIS stronger, the data collection phase in iterative design does not need to be tightly coupled with a specific UIMS. In the case of CHIME, while the monitor is built in a UIMS environment, it can be decoupled from the UIMS to operate independently. The underlying principle here is building a generic data capture tool that can be used for data collection in a wide range of applications. This requires that the monitor has knowledge about the semantics of the interface and its interaction units. Once this knowledge is available, it would be possible to collect data on existing applications that were not built using any specific UIMS; it will also not be bound to any specific metaphor, as is the case with the UIMS approach. Furthermore, the existence of a generic monitor instantiable for a wide range of situations provides a significant economy of scale by distributing the development effort throughout a wider range of opportunities for its utilization.

An interaction model was developed for the purpose of defining how to make available to CHIME knowledge about the interface to be monitored. What follows is a description of the model.

A Model for Interaction

The human-computer interaction environment may be viewed as consisting of the user, the application, and the interaction between them. The computer side is represented by the interface, whose purpose is to give the user information on the relevant states of the application and to accept commands and data from the user. On the user side, there is a set of goals to be accomplished; some information that may be helpful or necessary to reach those goals, and some strategic knowledge of a combination of tasks that will lead to the goal.

One important aspect of the interaction model is the acknowledgment that both the application (or, more appropriately, its interface) and the user exist, and are separate entities. The interface has its features and behavior specifications regardless of the user who is interacting with it, and the users have their sets of goals, models of the problem, and some information, regardless of whether or not they are using a specific interface. This simple statement led us to create a model that consists of three subsystems: the interface, the user, and the communication between them. Both the interface model and the user model can be defined separately, and the communication model establishes the connection between the two.

The Interface Component

The interface model conceptually separates the actions that may occur in two layers of interaction units (IUs): system and syntactic. System layer IUs constitute the low level events captured by the monitor, which generally correspond to what the monitor obtains directly from the device drivers or the window system. The set of syntactic layer IUs forms a language that consists of combinations of system layer
IUs, and describes the interaction at the level that might be found in an application's reference manual: the user command level.

Any IU can have arguments. For example, at the system level we may say that a MouseDown IU has arguments timestamp, x, y, and button; at the syntactic level, a MenuItemSelection IU could have arguments timestamp, duration, and mechanism.

System layer IUs are simple translations of the system events. Our model describes the system IUs by specifying a mapping between system events and System IUs. This mapping defines how system events are translated into system IUs, and how their arguments are derived from the system information. The main purpose of this layer is to create an abstract unit at the lexical level that allows for a greater portability of the monitor. In fact, all the processing done by the monitor (and any other tools that may follow it down the path of the development cycle) is made on device- and system-independent IUs. The early conversion of system information to IUs, along with a simple way to specify that conversion, leads to greater portability and reusability of the remainder of the monitor and other applications. In fact, the monitor captures system events and immediately converts each of them into a system IU. The remaining processing is done on IUs, which are machine independent, thus providing for a highly portable monitor. Below is an example of a simple system IU specification.

The model comprises a language for specifying IUs. The specification of a system IU includes three main aspects: the declaration of the IU itself, and of its arguments; the specification of a mechanism to detect the occurrence of that IU; and a way to construct that IU from the system information. The declaration of the IU consists of giving it a unique identifier; optionally, arguments may be attached to each IU, and in this case these would be defined in the IU declaration. To identify an IU we have created the concept of a test relation: tests are tuples of a test relation that define the mapping between system events and system IUs. Finally, constructors are processes that perform the conversion of system information into IU argument format.

We wish to illustrate the system interaction model with an example. Figure 2 presents an example of a very simple IU specification: a MouseDown IU. This specific example presents the declaration of that IU and the definition of four arguments. Then it describes how to connect the system event (in this case, the X11 Window System event) with the system IU. The first step ("EVENT event : XEvent") is necessary only once regardless of the number of IUs defined in the interface specification. It indicates that system events are of the data type XEvent and declares a dummy identifier event of that type. This identifier is to be used in tests, constructors, and as the variable holding the system event. The test relation in this example is defined with only one tuple, relating the X Event type ButtonPress with the system IU MouseDown. Thus, whenever an X11 ButtonPress event occurs, a MouseDown UI is generated. Finally, we present four constructors for the four arguments of the MouseDown IU. Each constructor consists of a short piece of C code to perform the conversion.

```c
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "IUX11.h"

IU MouseDown
ARGUMENT timestamp : time
ARGUMENT x, y : int
ARGUMENT button : int

EVENT event : XEvent;

TEST MouseDown TEST
event->type == ButtonPress

TEST

CONSTRUCTOR MouseDown
timestamp = X11timeToUltime (event->xbutton.time);
x = event->xbutton.x;
y = event->xbutton.y;
button = event->xbutton.button;
```

Figure 2 – Specification of a simple interaction unit

The system IU specification can be read by a cross-compiler, that reads the IU specification and generates a file of the C function source code that performs the identification and construction of syntax IUs. Figure 3 shows the output of the cross-compiler, when given as input the IU specification of Figure 2. This function can then be compiled separately and linked with the CHIME engine to generate an executable code of a monitor for the X Window System.

```c
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "IUX11.h"

#define FALSE 0
#define TRUE (!FALSE)

int IUToken (event, iu)
IUtoken *event;
IUtoken *iu;
{
    if (event->type == ButtonPress) {
        IUSetName (iu, "MouseDown");
        IUSetArg (iu, "timestamp", X11timeToUltime (event->xbutton.time));
        IUSetArg (iu, "x", event->xbutton.x);
        IUSetArg (iu, "y", event->xbutton.y);
        IUSetArg (iu, "button", event->xbutton.button);
        return TRUE;
    }
    return FALSE;
}
```

Figure 3 – C code generated from the example specification

Syntactic layer IUs are made of compositions of system IUs. They are specified using a context-free grammar and pre-conditions, that have to be met for a grammar rule to
apply. Syntactic IUs may be hierarchically defined. The
monitor will parse the stream of system IUs to build the
syntactic IUs. Every matching syntactic IU is a candidate
for recording by the monitor, depending on the monitoring
parameters set for the session. After a sequence of tokens
for an IU is identified, the arguments of that IU are filled
out from the information contained in the tokens of its
system IUs. The system conditions may be changed and
these will become pre-conditions for future IU matching.

The User Component

The user model is composed of User Knowledge Units, or
UKUs, that model the user knowledge of the system, the
problem domain, and the problem instance. One
particularly important class of UKUs are the User Goals, or
UGs. User Goals are defined as results the user wishes to
obtain by executing tasks with the application. UGs can be
defined at several levels. Two users may have different UGs
for the same system, even if they execute the same tasks.
For example, novice and expert users chunk their
information and goals in different ways, reflecting their
expertise. We have collected some preliminary
experimental data providing evidence for this claim. In
particular, early analysis shows us that novice users are
more concerned with low level actions and environmental
conditions, and expert users chunk the actions in higher
levels of abstraction. The way in which UGs and UKUs are
combined is addressed elsewhere in work on skill acquisition
and novice-expert differences [Badre 82].

The Communication Component

The Communication model establishes the relations
between sequences of IUs and UKUs. These relations can
be identified in several ways: during the analysis phase of
the iterative design cycle; by interviewing the users; by
empirical studies; and by matching cognitive models with
the application model. The arguments of the syntactic IUs
belonging to each sequence that is mapped onto a UKU are
passed along and converted by the relation in the
Communication model.

Characteristics of CHIME

An application interface is generated using either a UIMS
design tool, traditional software development methods, or
both. If it is generated using a UIMS design tool, it might
be possible to automatically generate an interface
specification document, which would be then read by the
monitor. If, however, no such document can be generated
automatically, some reverse engineering of the interface
would have to be performed. Therefore, CHIME is designed
to accept complex and detailed interface descriptions, such as
those generated by a UIMS, or simple and incomplete ones,
such as those that result from a possibly crude reverse
engineering process.

Another requirement for CHIME is that it minimizes the
effects on the user's work. In order to minimize interference
with user work, the monitor runs in the background of the
user environment. It can even be executed on a different
machine. In general, a monitor should not be integrated
into the application that the user is executing. Furthermore, integrating the monitor functionality into the
user application may be impossible or unfeasible.

The CHIME architecture makes several assumptions about
the operating system underlying the monitor. It should
support concurrency, or at least some form of voluntary
yield of control from the running process (like on a
Macintosh running System 6.x or earlier). It should also
have some mechanism for interception and replication or
reporting of interaction.

CHIME acquires its knowledge from the specification of the
interface component made using the interaction model
presented in the previous section. This model gives
CHIME a description of the interaction units, and allows
the analyst to control the monitoring activity using a
mental framework at a high level of abstraction.

As long as a description of the relevant interaction units is
available to CHIME, it will be able to perform its task.
Therefore, CHIME can be used to collect usage data from an
application that is already existing, and whose interface was
not built using any specific UIMS; it can also be supported
by any future UIMS or improvements on current UIMS,
which could provide an interface specification suitable for
use by the monitor as one of the forms of output of their
interface design tools.

The interaction model allows for specification of both types
of interaction units: interface management interaction units
and application specific interaction units. The monitor is
made aware of, not just the syntactic units of the interface
(such as buttons and keywords), but also of the semantics of
such units. This allows capturing context-dependent data.

CHIME architecture

Figure 4 depicts CHIME's environment architecture.

The system interaction interception module (SIIM)
communicates with the interface management system (the
operating system, the window system, or the UIMS
runtime system, whichever is being used to control the
interaction in the user environment). This is the only
module of the monitor that is environment-dependent, and
has to be custom-written for each system on which CHIME
is to run. It senses user actions and application responses,
converts them into an internal environment-independent
representation, and passes them to the module that identifies
interaction units. The SIIM performs its task with minimal disturbance of the interface.

The identifier of interaction units takes as input the interaction atoms captured by both, the system interaction interception module and the interface description generated by the UIMS or by a reverse engineering process. It is basically a parser that identifies higher-level interaction units based on the lower-level interaction atoms. It then passes these interaction units to the identifier of relevant interaction units.

The identifier of relevant interaction units is controlled by the monitor session controller, which is in turn controlled by the experimenter. It determines the relevant units for recording in the current experiment session, based on the information received from the monitor session controller. The units that are found to be relevant for the session are then formatted giving the output of the monitoring activity. The output can then be stored for later analysis, or fed directly into an analyzer for real-time analysis of the interaction.

To control the monitoring session, the experimenter uses the experimenter’s console. This console is in most cases remote from the user workstation, and will provide the experimenter with the monitor control panel. Through the control panel, the experimenter can select the interaction units to be monitored and recorded, and those that are to be discarded. That information is passed on to the monitor session controller, which in turn sends control information
to the system interaction interception module and to the identifier of relevant interaction units.

On the experimenter console, the experimenter can also be running some analysis tools during the data collection process. These would enable the experimenter to have some early real-time results of the interaction.

Current implementation of CHIME

The current version of CHIME performs a subset of the full set of features. The features implemented are:
- capturing of system layer interaction units.
- recording of captured units.
- capability to configure types of actions to capture and relevant screen areas.

The prototype was written for operation in the X11 Windows environment. It intercepts X events that occur in the relevant screen areas, and senses pointer movement. It then checks if the detected actions are the types of actions that are to be recorded, and if so, records them. Arguments of interaction units, such as time stamp, pointer position, and status of buttons or keys, are also recorded.

By taking advantage of the X11 networking abilities, the monitor can be executed on the same workstation that the user is running on, or on a remote workstation. Execution on a remote workstation minimizes burden on the user workstation, thus minimizing interference with user work. Network traffic is inevitably increased, but this is kept to a minimum by the usage of a filtering mechanism on the user workstation to inhibit reporting of actions that will be discarded by the monitor. We have not observed any perceivable degradation in system performance due to the background presence of CHIME performing its monitoring activity.

Integration of CHIME in a UIMS

CHIME does not rely on the existence of any user-interface management system (UIMS) to support its execution. Clearly the design tool of a UIMS would be a useful tool to automatically generate the description of the interface, but that is not strictly necessary in our architecture.

The integration into a UIMS would involve a translation tool, specific to each UIMS, to translate the interface description generated with the UIMS into the interaction model. Due to the openness of our model, this translation tool should be quite straightforward to build for most UIMSs. This translation tool should produce the interface component specification documents in the CHIME model. Both system UIs and syntactic UIs have to be generated. The definition of the system UIs should include their identifying names, their constructors (specific for the UIMS runtime system) and all the test relations between the UIMS representation and the CHIME UI representation. The definition of the syntactic UIs should be translated from the UIMS representation into the CHIME representation based exclusively on CHIME system UIs. For this translation to be possible, it is necessary that the UIMS possess a model to support knowledge about the interaction [Foley 88].

Problems and Future Work

Automated transcription techniques, and CHIME in particular, are not sufficient for gathering all the usability data that is relevant for analysis. User satisfaction assessment techniques [Bailey 83, Chin 88] should still be used to complement the information gathered by CHIME. We intend to integrate techniques to consider non-computer-related events and conditions into the model. This will allow or analysts to operate on all the information known about a specific interface.

We also have plans to continue developing the model, in particular to address modality in applications, errors and error recovery, and to investigate its adaptability to a range of cognitive models.

In [Jones 90], George Robertson points out that techniques need to be developed that allow device drivers to be defined at a higher level of abstraction, then compiled into an appropriate form. We stand by this point, and plan to move ahead of the device driver implementors by revamping the initial CHIME implementation to include this feature, with or without the "smart" device drivers.

Summary and Conclusions

Knowledge-based monitoring is a technique that incorporates several of the advantages of the other techniques, minimizing undesirable drawbacks. It has the potential to match or surpass every other computer-based technique. It also has the potential for integration with non-computer-collected data. Most importantly, it can be used with or without a UIMS, thus opening up a wide range of opportunities for accurate interface evaluations.

Acknowledgements

The authors wish to thank the Center for Information Management Research for partially funding this research under an NSF grant. We also wish to recognize the support of JNICT for the research work of Mr. Santos, under grant number BD/733/90-IA.
References


