A Knowledge-Based System for Capturing Human-Computer Interaction Events: CHIME

by

Alber N. Badre
Paulo Santos

GIT-GVU-91-21
September 1991

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

A Knowledge-Based System for Capturing Human–Computer Interaction Events: CHIME

Observations and Issues

Albert N. Badre
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
phone: +1 – 404 – 894-2598
email: badre@cc.gatech.edu

Paulo J. Santos
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
phone: +1 – 404 – 853-9393
email: pas@cc.gatech.edu

ABSTRACT

The ability to capture and analyze human-computer interaction (HCI) data is an important part of the iterative interface development process. In this paper, we claim that in order to perform accurate and extensive analysis of HCI events, an automated technique has to be used. We describe steps towards identifying desirable features of such a technique and present a list of those features. We present the key characteristics and architecture of a knowledge-based monitoring system, CHIME. We claim that the more knowledge a monitor has about the semantics of an application the more useful the data it collects. Finally, we report on the results of an exploratory study to assess the adequacy of key CHIME assumptions and architecture.

KEYWORDS

User Interfaces, Human–Computer Interaction, Human–Computer Interaction Monitoring

INTRODUCTION

The scenario

Two of the steps in the iterative design (Figure 1) of quality human–computer interfaces [Gould 85] are the collection and analysis of interface utilization data. Several techniques have been used to collect such data. Historically, data gathering has been accomplished either manually, by automated means, or a combination of the two. Examples of manual or combination data collection techniques are session recording by expert annotators or videotaping of sessions. These techniques require tend to be costly in terms of effort and manpower. In addition, the accuracy and completeness of the data collected is limited by the abilities of the human observer.

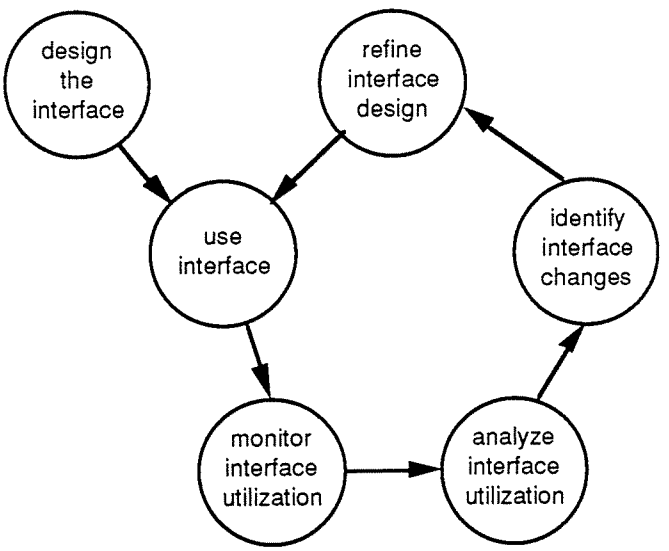

Figure 1 – Iterative Interface Design Model

To overcome these problems, a number of computer-based techniques have emerged. The initial and most basic of these techniques consisted of embedding trace commands in the program being executed. This technique, one of the first

used extensively in debugging, has the capability of recording very detailed information and of identifying the relevant context. Its unpopularity is due to the necessity of introducing modifications to the program each time a different monitoring function is required.

Then came automatic transcribing [Good 85]. Using this technique, computers gather data about themselves. All the interaction with the user can be captured, generally at a low level of abstraction (keystroke or mouse click level, or at the command level). This technique provides a remarkable advancement over the embedded trace techniques, especially since it does not require changes to the application or interface code each time a different monitoring function is needed. However, a bare automatic transcription still has some serious drawbacks. First, the amounts of raw data generated are generally extremely very large, 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. And, as all the techniques that are based exclusively on information gathered by the computer, 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.

Finally, the UIMS explosion brought along some UIMS-based transcription facilities, such as in [Olsen 88]. This technique differs from the automatic transcription described above in the fact that a User Interface Management System is in control of all the interaction. In particular, the UIMS has some knowledge of the model of the interface, and can thus build and record higher level interaction units by combining lower level ones. However, the UIMS approach, although extremely valuable in user interface design and support, did not solve all the problems associated with interaction monitoring. First, this technique requires the program to be running under the control of a UIMS runtime system, increasing the size and complexity of every interface application generated with that UIMS. Secondly, the UIMS-based technique can not be used with interfaces built outside of the world of that UIMS. And finally, like all other computer-based techniques, it totally disregards actions and conditions not directly affecting the computer.

The problem

Automated HCI monitoring and analysis techniques are beneficial mainly because of their accuracy and the potential for extensive automation. However, it is clear to us that current automated techniques have serious drawbacks and limitations. We also realize that some of the potential of manual techniques is lost by relying exclusively on automation. There is a large body of important HCI events that can not possibly be captured by the computer. Examples of such events are user inactivity ("why isn't the user typing?"), user off-line activities {user documentation look-ups}, environmental conditions (light and noise levels), psychological aspects ("the user has just received a promotion"), and demographic data (gender, age, etc.).

The current unsatisfactory state-of-the-art of HCI monitoring suggests that there is room for improvement. Thus, we proceeded to investigate a new technique, knowledge-based monitoring, that would eliminate or reduce the problems of current techniques, while at the same time, incorporating their good features. In knowledge based-monitoring, the monitoring task is performed essentially by the computer, but the computer is given a specification of the interaction to enable it to monitor higher-level interaction units.

We have developed a prototype knowledge-based monitoring engine which we called CHIME [Badre 91]: knowledge-based Computer-Human Interaction Monitoring Engine. A description of CHIME will follow in the next section.

This paper presents some studies that we performed to test the adequacy of CHIME for the HCI monitoring task. We also identified changes and improvements needed in CHIME.

BRIEF DESCRIPTION OF CHIME

Characteristics of CHIME

In conceptualizing CHIME, to assess interface quality, the strategy was to incorporate the following key desirable characteristics of automatic knowledge-based monitoring.

The first and most important characteristic is "automatic" monitoring. This is the only way to achieve high accuracy of the data collected. It is also important to automate the monitoring process to reduce manpower costs.

A second important characteristic 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. This enables the collection of only the necessary data for some analysis task. It is essential to keep the amount of collected data down to a bare minimum, while capturing all the data relevant for the analysis.

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]. Chunk size is associated with expertise level, and may be detected by automatic time monitoring of pauses between responses or actions. Based on these and other studies of chunking, CHIME incorporates the capability to identify action chunks based on an inter-action time measure.

Another characteristics of CHIME is its ability for local or remote monitoring of human–computer interaction. Results of the monitoring process can be stored for later analysis, or they may be immediately analyzed and the results displayed on the user's screen or on another screen.

CHIME is also independent of any specific UIMS. 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 UIMS 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 instantiatable 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.

CHIME has knowledge about the interaction. The CHIME interaction model supports interaction specification of the two interacting agents: the human user and the machine. These two components of the CHIME interaction model are related by a third component, the communication component. In an ideal interface design, the human and machine components of the model are identical, and the communication is a simple one-to-one mapping. However, we are far from achieving interfaces with such powerful designs. Meanwhile, our goal is to design the interface (thus indirectly the system component of the model) in order to reduce the complexity of the communication component.

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 characteristic of CHIME is that it minimizes the effects on the user's work. It achieves this by running a minimal CHIME kernel in the background of the user environment.

CHIME is independent of specific applications, interface managers, or UIMSs. 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.

CHIME acquires its knowledge from the specification of the user and machine components made using the interaction model. 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.

CHIME architecture

Figure 2 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

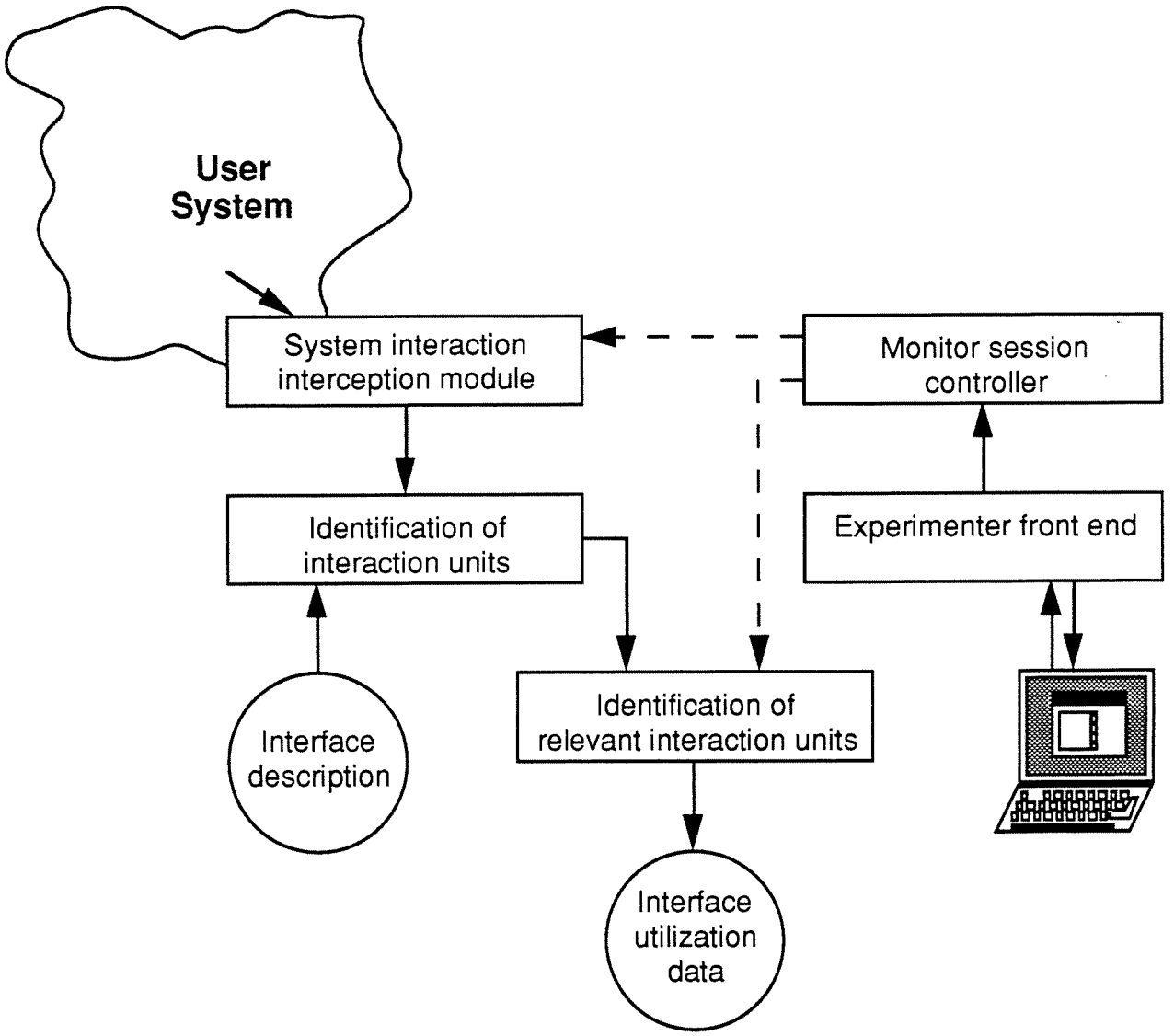

Figure 2 – CHIME monitor architecture

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.

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.

APPROACH AND RESULTS

Methodology

In order to investigate the adequacy of the CHIME assumptions and architecture, we developed a series of field studies. The initial objective of these studies was to identify features of human-computer interaction that should be considered in monitoring. The ultimate goal would be to incorporate them into a working prototype of CHIME.

We prototyped human-computer interaction monitors for two platforms — the X Windows System and the Apple Macintosh. With these tools, we monitored users in their normal work on those platforms and recorded the

interaction. We also performed manual transcription of human-computer interaction, using both expert and novice annotators.

These studies were undertaken as empirical field observations rather than controlled experiment in order to minimize turnaround time as well as to emulate the field as closely as possible. Nevertheless, two quasi-experiments were designed and executed. In the first study, users were asked to perform some text editing and information search and navigation tasks on a computer. Annotators, both novices and experts at the tasks, were asked to record every action made by that user that they felt was relevant for the study of the interaction, recording as much detail as possible. The users were made aware of the fact that their actions were being monitored by annotators.

The second study was designed to extract hints on how users represent and translate a computer task into actions on the computer, and how those representations change with practice. In this quasi-experiment, users (subjects) were asked to perform repetitive tasks on a computer. Each session consisted of three periods of fifteen minutes each. In each period, the users would have to solve repetitively a task using an interface. Two tasks were available, and two interfaces were available for each of those tasks.

Both tasks consisted of algebraic manipulation, using basic operations at the elementary school level (addition, subtraction and division). To complete the task, the users would have to combine a series of operators and operands to reach a goal. Every task presented to the users was solvable with one to four applications of operators to operands — although it could also be solved, using non-optimal strategies, using longer combinations. In both tasks, every error was recoverable, and therefore no user action could result in an unsolvable situation. Task 1 consisted of sequentially adding or subtracting a set of 8 numbers from a target number, with the purpose of bringing the target number to zero. No pair operator/operand was invalid. The two operations were the simplest. Is is important to realize that there was not a simple algorithm that would always

solve the problem. Task 2 had the same goal, but allowable operators were subtraction and division. Operations that would result in non-integers or in negative numbers were not allowed. In this task, one of the 8 numbers used as operands was always the number 1. Therefore, this task had an algorithm that would always solve the problem, although not necessarily the by optimum number of operations: sequentially subtracting the largest number smaller or equal to the target number.

Both interfaces were mouse driven. Figure 3 shows an example of interface A being used for task 1. To solve that problem using an optimal strategy, the user could select the sequence **- 34 - 34 - 20 - 5**. After 15 minutes of using on pair task/interface, one of the variables (task or interface, i.e., syntax or semantics), were changed. After another 15 minutes, the other variable was again changed. Users were told to solve as many problems as possible in each of the 15-minute sessions. After the first and last trial of each 15-minute session, the users were asked to verbalize their mental process.

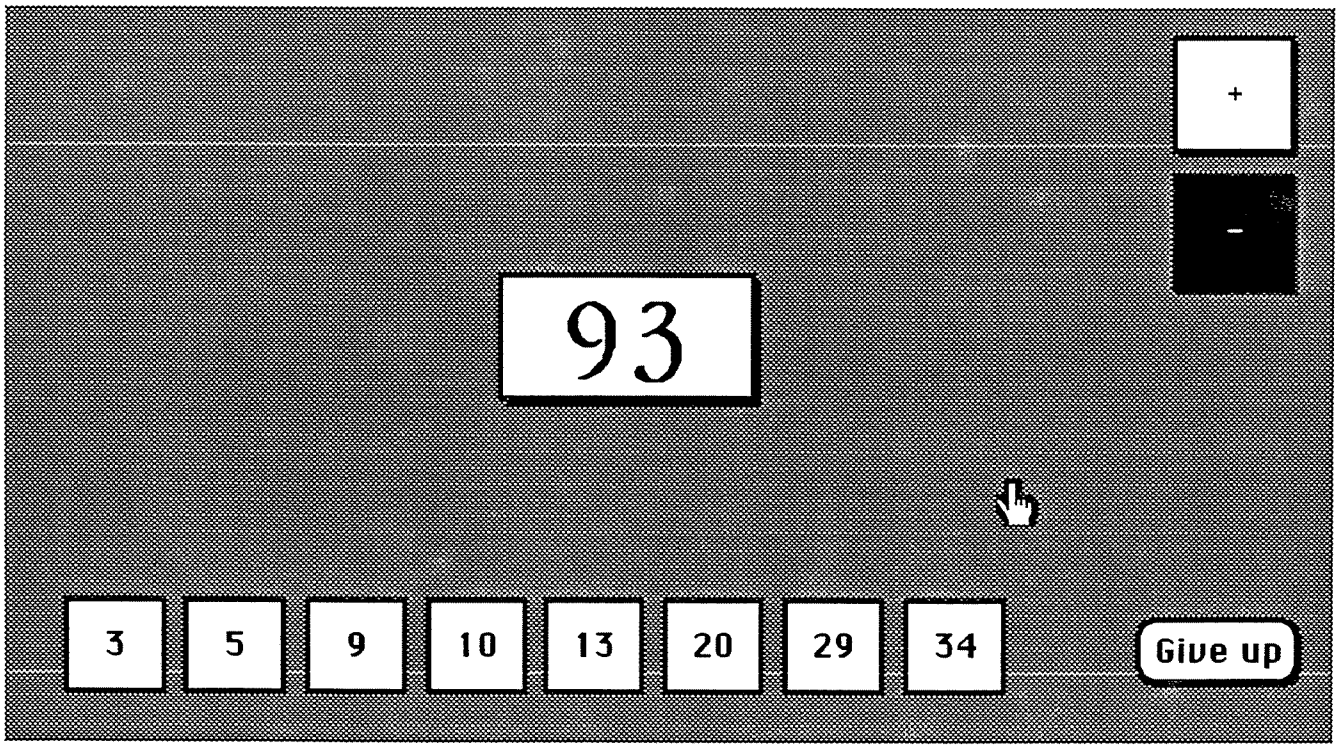

Figure 3 – Example of a screen using interface A to solve task 2 in quasi-experiment 2

Results of the studies

The first study allowed us to identify what type of information annotators collect during HCI sessions. We also observed how novice and expert annotators observe and record different aspects of the interaction.

Novice annotators tended to record mainly two types of actions: high-level domain specific actions (such as "Deleted ZIP code") and physical actions (such as "When he needed to use both hands on the keyboard, he put his pen in his mouth" or "He scratched his forehead when he was trying to remember something").

Expert annotators recorded a lot more information directly related to the interaction with the computer (e. g., "User cut and pasted a segment" or "Selects text (click – hold down – move"). They do that at a high level, though, and details such as typing errors, timing information, etc., were seldom or never captured by the expert annotators.

In the second study, we investigated how human models and computer interaction models relate. we analyzed whether practice with a computer task helps users deal with complexity. It was expected that experienced users deal with complex situations faster and better than novice users. The results show that:

- Well-practiced users have larger action chunks than do beginners. This was detected by analyzing action time intervals from the timestamps recorded for each action. The pauses between actions for experienced users were shorter and less frequent than for beginners.

- Both novice and experienced users use non-optimal strategies to solve problems. However, as users get more practice, they are able to improve their strategies. This seems to happen in bursts: users will try a modification to their strategies, and if the modification works with better results, they start using it again. If it doesn't work as desired, they revert to the old strategy.

The above results led to the identification of several features that are desirable for effective human-computer interaction monitoring. What follows is a list of those features along with brief justifications. We have already incorporated some of those features into CHIME.

DISCUSSION AND IMPLICATIONS

The first study suggests that information gathered by human annotators is extremely diverse in nature. It also shows that not all of this information can be captured automatically (such as "user puts the pen in his mouth") by the computer. This suggests that an integrated monitoring tool should consider external factors as well as data gathered by the automatic monitoring process. This result will lead us to incorporate external factors into the knowledge model of CHIME.

The second study reveals the possibility of automatically evaluating the user level. By identifying chunks, their sizes and timing between them, it should be possible to infer the user's level of expertise. The CHIME interaction model provides a basis for a description of the chunks. The capability for automatic chunk identification is obtainable from this model.

The second study also revealed that users change their strategies as they get more practice with using the system. In order to change their strategies, they continually experiment with new ones. One important feature of the monitoring process is to detect the new strategies being tried out by the user, and to give the user instantaneous feedback on their performance. CHIME incorporates this facility by including graphical display of feedback and data on the user's screen, as well as on a remote screen.

Interaction Features

1. It should be possible to monitor and record interaction at several levels of abstraction. This is to accommodate the increasing complexity of the interfaces. For example, a monitoring tool should be able to record lexical interaction units such as mouse and keyboard actions, or higher level units such as commands, tasks, errors, etc.

2. It should be possible to capture the user's model of interaction. Communication occurs between a user and a machine. The machine's model can be known *a priori* (hardware, application and interface metaphors, etc.), but the user's model of the interaction and the user goals need to be captured during the interaction session. This is indispensable in order to evaluate how well the interface is helping the user in communicating with the application. We realize that this task can not be fully automated. However, automation can be used to gather some information about the user. For example, as some of the data collected during our studies suggests, user expertise levels can be identified by analyzing the timing in the chunks of interaction [Badre 82, Smelcer 86]. Novice users chunk fewer items of interaction together, and more experienced users build larger chunks and execute them with shorter pauses between chunks.

3. It should be possible to relate user and application models. This is the obvious feature needed to evaluate the interaction: how the user and application models relate and how the interface helps in the translation process.

4. It should be possible to identify the relevance of different human–computer interaction events. Some events are relevant for exploring the validity of some hypothesized conclusion, others are totally irrelevant.

Recording every event and later analyzing their relevance is generally not a viable option due to the extremely large amounts of interaction information that would be recorded unnecessarily. 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.

Data collection accuracy features

5• Every user action, at every level, should be monitorable and recordable in detail. Analysis tools may analyze interaction at different levels, and therefore interaction needs to be recorded at each level. The monitoring technique should ensure that record have the capability of recording details of the higher level interaction units, gathered from the lower level units and summarized. For example, a menu selection (as on a Macintosh) could be performed by pressing the mouse on the menu title in the title bar, moving the mouse to the appropriate item, and then releasing it. Or it could be selected by a keyboard equivalent command. In this case, a monitoring tool should be able to record not only that the menu item was selected, but which mechanism was used to select it (key or mouse) and other lower level information such as timing, mouse position, etc.

6• Timestamps should be attached to every action. For atomic actions, the event time should be recordable. For compound actions (such as the menu item selection using an item), timestamps should be recorded for the beginning and end of the action, and possibly for relevant intermediate steps.

7• Detailed low level interaction should be recordable. Details such as screen coordinates of user actions, subtle mouse movements or trajectories, etc., can be important in analyzing 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, and is an important measure that allows for fine tuning of an interface for optimum performance.

CONCLUSION

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.

CHIME implements most of the desirable characteristics of knowledge-based monitoring. Continuation of the work in CHIME will lead to the incorporation of yet some more features.

The knowledge-based monitoring technique that CHIME uses provides an increased functionality over the UIMS-based approach. It is also possible to integrate it into existing or future User Interface Management Systems.

PROBLEMS AND FUTURE WORK

Automated transcription techniques 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 analyzers to operate on all the information known about a specific interface.

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 are in the process of implementing a CHIME prototype to include this feature, with or without the "smart" device drivers.

REFERENCES

[Badre 82] Badre, A.N., "Selecting and Presenting Information Structures for Visual Presentation", IEEE Transactions on Systems, Man, And Cybernetics, 12, 4 (July 1982).

[Badre 91] Badre, A.N., and Santos, P.J., "CHIME: A Knowledge-Based Computer-Human Interaction Monitoring Engine", Technical report GIT-GVU-91-06, Georgia Institute of Technology, Atlanta (1991).

[Bailey 83] Bailey, J. E., and Pearson, S. W., "Development of a Tool for Measuring and Analyzing Computer User Satisfaction", Management Science, 29, 5 (May 1983), pp. 530–545.

[Good 85] Good, M., "The Use of Logging Data in the Design of a New Text Editor", Proceedings of the CHI'85 Conference on Human Factors in Computing Systems (1985), pp. 93–97.

[Gould 85] Gould, J. D., Boies, and Lewis, C. H., "Designing for usability: Key Principles and Waht Designers Think", Communications of the ACM, 28–3 (1985), pp. 300–311.

[Jones 90] Jones, W., Williams, P., Robertson, G., Joloboff, V., Conner, M., "In Search of the Ideal Operating System for User Interfacing", Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (Snowbird, Utah, USA, October 3–5, 1990), ACM Press (1990), pages 31–35.

[Olsen 86] Olsen, D. R., "MIKE: The Menu Interaction Kontrol Environment", ACM Transactions on Graphics, 5, 4 (October 86).

[Olsen 88] Olsen, D. R., and Halversen, B. W., "Interface Usage Measurements in a User Interface Management System", In Proceedings of ACM SIGGRAPH Symposium on User Interface Software (Banff, Alberta, Canada, Oct. 17–19), ACM Press, 1988, pp. 102–108.

[Siochi 89] Siochi, A. C., "Computer-based user interface evaluation by analysis of repeating usage patterns in transcripts of user sessions", (doctoral dissertation) Virginia Polytechnic Institute & State University, (1989).

[Siochi 91] Siochi, A. C., and Hix, D., "A study of computer-supported user interface evaluation using maximal repeating parttern analysis", Proceedings of the CHI'91 Conference on Human Factors in Computing Systems (May 1991), pp. 301–305.

[Smelcer 86] Smelcer, J. B., "Expertise in Data Modeling or What is inside the Head of the Expert Modeler", Proceedings of the CHI'86 Conference on Human Factors in Computing Systems (1986).