Haptic Output in Multimodal User Interfaces

Stefan Münch and **Rüdiger Dillmann**
University of Karlsruhe, Department of Computer Science
Institute for Real-Time Computer Systems and Robotics
Kaiserstraße 12, D-76128 Karlsruhe, Germany
phone: +49 721 608 3547 fax: +49 721 606740
email: {smuench,dillmann}@ira.uka.de www: http://wwwipr.ira.uka.de/

ABSTRACT
This paper presents an intelligent adaptive system for the integration of haptic output in graphical user interfaces. The system observes the user's actions, extracts meaningful features, and generates a user and application specific model. When the model is sufficiently detailled, it is used to predict the widget which is most likely to be used next by the user. Upon entering this widget, two magnets in a specialized mouse are activated to stop the movement, so target acquisition becomes easier and more comfortable. Besides the intelligent control system, we will present several methods to generate haptic cues which might be integrated in multimodal user interfaces in the future.

KEYWORDS: Haptic output; User modelling; Adaptive interfaces; Intelligent feedback; Multimodality

INTRODUCTION

Basically, computers are only tools which should support the human user in any kind of task—preparing the tax return, designing a new car, analyzing financial data etc. But most of these tasks require a certain amount of human-computer interaction (HCI). In this process, information to the human is mainly conveyed via the visual channel, whereas inputs are made with a keyboard and a pointing device (motor channel). This situation is in contrast with real-world communication between humans, and it leaves certain capabilities of both, man and machine, unused: "In an earlier work (Buxton 1986), I speculated on what conclusions a future anthropologist would draw about our physical make-up, based on the tools (namely computers) used by our society. The objective was to point out that these tools reflect a very distorted view of our physiology and the motor/sensory skills. ... Nevertheless, things have changed very little in the intervening years." [7, p. 1] In the following, we will analyze the reasons and propose a solution for the integration of haptic output in user interfaces.

IUI 97, Orlando Florida USA
© 1997 ACM 0-89791-839-8/96/01 ..$3.50

Haptics in HCI?

With the development and dissemination of multimedia and virtual reality (VR) systems, things begin to change. Speech input and output are commercially available and animations are accompanied by sound. But what about *haptic output*, i. e. feelable cues? Does today's HCI make use of the human's sense of touch and kinesthesia? According to Bevan, no: "Whereas synthetic visual and audio images are omnipresent nowadays, opportunities to reach out and feel non-existent objects possessing texture, shape and inertia are, to put it mildly, not." [2]
In our opinion, the *haptic modality* is clearly missing in HCI today. Not because it is not useful, but simply because it is much more complex than graphical and acoustical output (see next section). Therefore, we have developed a system for the integration of haptic output into an application's user interface (UI). It runs in the background and is independent of the application, but it only operates on UIs written in Tcl/Tk. It is a prototype at the moment, but it already works well with the combination of a simple device and a complex controller.

The ForceMouse

For the interaction process, i. e. the control of the cursor and the haptic output, we have built a specialized 'multimodal mouse' with a moveable pin in its left button and two electromagnets in its base. This socalled FORCEMOUSE (Figure 1) is based on an idea by Akamatsu & Sato ([1], see below), but instead of one we have implemented two electromagnets. Thus, the movement of the mouse can not only be rendered more difficult but can be stopped completely.

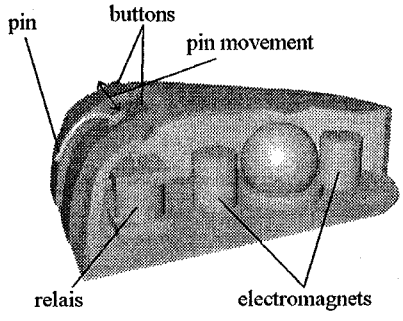

Figure 1: The FORCEMOUSE with its moveable pin and two electromagnets in its base

The Basic Idea

Two basic ideas have been the starting point of the system's development. First, we wanted to give the user an impression of the interface's structure and texture during mouse movements—this is realized with the pin. Second, the magnets are used to stop the mouse when the cursor is above an interaction object if the user wants to use it, see Figure 2. This kind of support can not be achieved by pure muscle memory, because it is independent of the position of an application and its widgets on the screen.

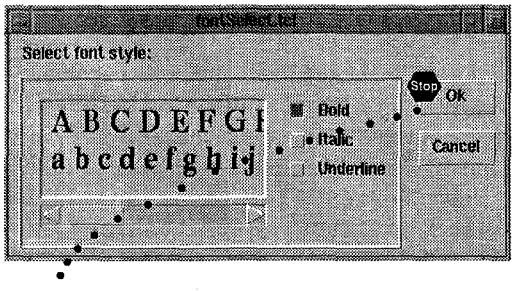

Figure 2: Kinesthetic feedback with prediction of the next interaction object

Especially for this 'positioning support', an *intelligent control* of the haptic output is needed. The system has to observe the user's interactions and to calculate the probability of a widget to be used. Based on this calculation, *the widget which is most likely to be used next is predicted*, and if the probability exceeds a threshold, the mouse movement is stopped. Thus, the selection of this object becomes significantly easier. Without the prediction mechanism, the mouse would be stopped above any widget—which is clearly not desired and can not be accepted, see Figure 3.

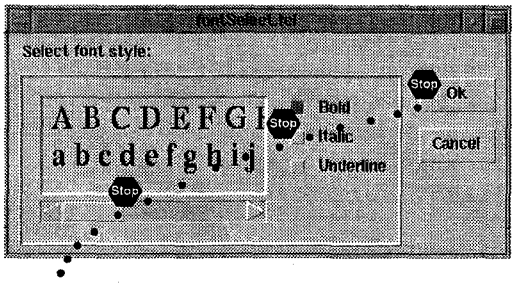

Figure 3: Kinesthetic feedback without prediction of the next interaction object

In the following section, we will give a brief introduction in haptic perception and a review of related work. In addition, we will discuss methods for the generation of haptic output and formulate some requirements which have influenced the system design. In the third section, the user analysis approach and its mathematical background as well as some aspects of the implementation will be described. Finally, we will present preliminary results, draw conclusions, and present our ideas for work in the future.

HAPTIC OUTPUT AT THE USER INTERFACE

Although the idea to integrate tactile output and force feedback in an application program is not new, there is yet no standard software available. In this section, we will examine the reasons for this shortcoming and propose two approaches to solve the integration problem.

Haptic Perception

Humans interact with their environment by using several modalities via several communication channels. Regarding perception, the visual channel (i.e. the sense of sight) is the most important one, but other modalities like hearing, smelling or feeling are important as well. Here, we will concentrate on *haptic perception*, a term which comprises cutaneous as well as proprioceptive and kinesthetic sensations. For a comprehensive discussion, see [13] or any good book on human physiology.

Tactile sensors can be found in the skin which is the largest human organ with a surface of more than 3,000 in^2. With these sensors, humans are able to recognize the overall shape of an object as well as the microscopic properties of its surface. With the finger tips, one can detect whether an object is rough or smooth, stiff or elastic. All of these properties can only be sensed in direct contact with an object, and no other modality serves this purpose.

Although "the field of touch is still very much in its formative stage" [13], some figures shall underline the power of the sense of touch. The skin has about 500,000 receptors of six different types which can process about 200,000 bits/sec. Vibrations can be sensed up to 10 KHz, and can be discriminated up to 320 Hz. For these sensations, an amplitude of 0.1 μm is sufficient. Pressure to the skin can be detected down to 10^{-5} N.

In addition, force feedback can be used in HCI to address the procioceptive and kinesthetic senses. Force feedback is wellknown in teleoperation for more than 50 years (e. g. [4]), and it plays an increasingly important role in VR applications [16, 6]. It can be used whenever object interactions or manipulations with contact relations are needed to reduce forces or to simulate objects more realistically.

There are some important aspects which distinguish the haptic modality significantly from hearing and seeing:

- Haptic interaction has a *spatial* (like visual) *and temporal structure* (like acoustic).
- Haptic interaction needs *direct contact* to objects.
- Haptic object properties may be sensed *actively or passively* by the human.
- The haptic modality is not a unique one but is composed of several *different sensations* which are conveyed by several *different channels.*

The last two aspects led to a classification of five different modes, ranging from *tactile perception* (passive, cutaneous information only) to *active haptic perception* with cutaneous information and afferent kinesthesis and efference copy [13].

Review of Related Work

If "the field of touch is still very much in its formative stage" then our knowledge on how to make effective use of it in HCI is still in its infancy. With force feedback in teleoperation and telerobotics it's a different matter, but regarding the creation of haptic sensations in standard applications the field has only recently started to emerge. Even worse, the few techniques and methods which exist are usually designed for a small range of applications only, and the integration of haptic cues in standard software is usually not addressed. Before we will present our findings, we will briefly review three other approaches.

A publication by two Japanese researchers deals with a multimodal mouse with tactile and force feedback [1]. Akamatsu & Sato have equipped a standard mouse with a movable pin in its left button and an electromagnet in the bottom. The pin can be raised or lowered to indicate a certain state or to create a kind of vibration, and the magnet can be used to stop the mouse if moved on a ferromagnetic pad. Experiments with target acquisition tasks revealed that the error rate increased slightly with haptic feedback, but the execution time could be reduced by 7.9% and 10.5%, resp. A result much more interesting is the fact that the effective target size has been increased significantly, which means that the target acquisition task is much easier with haptic feedback.

In Vienna, Austria, a research team has developed a FEELMOUSE to 'translate' the texture of a graphical user interface (GUI) into haptic cues [12]. A standard mouse has been equipped with an electromagnet at the left button which serves two purposes: first, it can actively move the mouse button up and down, thereby giving information about the interface's texture. Second, the resistance to press the button can be varied, so that crucial operations can be made more explicit to the user.

One major problem with force feedback in computer mice is its *isotropic* nature: the mouse movement can be made more difficult or even impossible, but the effect is independent of the direction. Even worse, the mouse can not be moved actively in either direction. This problem has been addressed by Engel et al. who have developed a trackball with socalled 'contextual' force feedback [9]; a context-sensitive feedback of the application's expectations with respect to the user's input. In other words, the user's interactions are interpreted by the program in order to give a more detailed and more useful kind of feedback which supports the interaction.

Engel et al. have equipped a trackball with two servomotors which control the x- and y-axis independently. For first experiments, users had to guide a 'ball' through a one-track maze—a task which clearly favors the kind of feedback supported by their device. Not surprisingly, the number of errors as well as the execution time decreased remarkably (by a factor of ≈ 4 and 1.5, resp.) In a second trial, a less specific target acquisition task was used. Here, targets of different size which appeared at a random position on screen had to be hit. The force feedback was realized by an S-shaped profile around the object's center. The primary goal of this investigation was to find out whether Fitts' law holds for devices with haptic feedback as well (yes, it does), but unfortunately the figures for execution time and error rate show only minor improvement when force feedback is used. Instead, another benefit has been experienced: "... the effort needed to master the trackball is minimal compared with that for the conventional trackball without force feedback." [9]

The findings of Akamatsu & Sato or Engel et al. are in full accordance with results from teleoperation experiments with force feedback: the major benefit is not reduced task completion time, but lower error rates, more intuitive interaction, and—in real contact situations—reduced forces. Sometimes, there is even the effect of 'intelligence amplification': "The most valuable result from using GROPE-III for drug docking is probably the radically improved situation awareness that serious users report. Chemists say they have a new understanding of the details of the receptor site and its force fields, ..." [5].

Generation of Haptic Cues

For at least two reasons, haptic output should be integrated into the UI: first, the haptic modality complements the visual channel. Interactions will be more comfortable and intuitive if the interface's structure is presented both, graphically and haptically. Second, output to the hand and fingers will reduce cognitive load. Whereas most visual cues have to be 'translated' and interpreted by the user before an action can start, at least some of the haptic cues can be processed on a much lower level, thus reducing the perception-action-loop to a minimum.

Unfortunately, at least two problems have to be solved: first, techniques and methods to generate haptic cues are needed. Some of them will be discussed below. Second, if the cues are available, we need a way to integrate them. A solution for this problem which does not affect the already existing application or its UI will be presented in the following section.

The first method we will describe models the pseudo 3D relief of the GUI with haptic cues. Therefore, this method can be called WYSIWYF—*What You See Is What You Feel*, i.e. the 'third dimension' which represents the structure or texture of the GUI is mappped to the position of a moveable pin or button at the input device. Today, pseudo 3D reliefs are presented graphically with colors and shadows to enhance the vividness of the interface, thus supporting the idea of direct manipulation.

An example of a 'haptic profile' derived from the GUI's features is depicted in Figure 4. The realization is quite simple because the widgets' 'level' can be used directly for haptic output with an appropriate device. One of the advantages of a haptic model is the reduced perception-action-loop, i.e. if a stimulus like a short vibration caused by a raising pin is directly presented to the finger which performs the action, the reaction time will be shortened. It is also practicable to model *all* widgets, because in contrast to force feedback the haptic output does not conflict with the user's input as long as the movement is not influenced significantly.

The second method also models the relief, but with a different intention. Instead of simply mapping the relief, the widgets

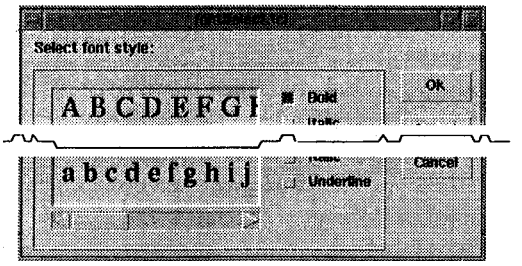

Figure 4: Graphic and haptic modelling of a pseudo 3D relief

are associated with potential fields to support the navigation process, see Figure 5. This method is more attractive because it can actively support interactions, but it is far more complex than the first one. The calculation of potential fields needs a lot of computing power, and to yield good results, dynamic calculations are needed. Moreover, the advantage of potential fields—to provide *directed* force vectors (a kind of gravitation)—can only be exploited with a device which gives directed output.

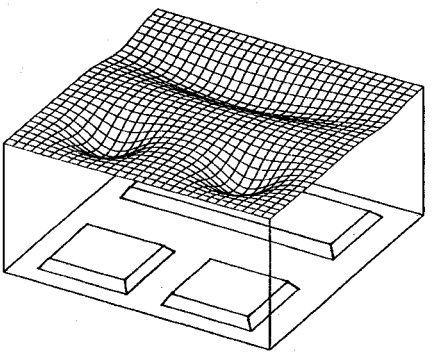

Figure 5: Haptic modelling with potential fields

Fortunately, there is a solution to avoid these drawbacks: if a simplified version is used which does not produce directed output vectors continuously but supports the navigation by *stopping the cursor and mouse movement* at the desired position only, the complexity of both the calculation and the device can be reduced significantly. Although the support during navigation will be reduced, too, there is still some benefit in the interaction process which is worth the effort (see next section).

Hard- and Software Requirements

For this simplified kind of haptic output, isotropic forces are sufficient, but for a more sophisticated output a more complex device will be needed. For 2D interactions, two degrees of freedom are adequate, so three types of device with different characteristics might be modified to provide haptic feedback. A *joystick* can easily be equipped with motors to provide directed force feedback, but it has limited input capabilities only. A better input device is the trackpoint which can be found in many notebooks now, but here the feedback is much harder to realize due to its tiny measurements.

The *mouse* is the most popular and universal 2D input device, and as described above, it can be equipped with haptic output generators. The best solution would be to combine the adjustable mouse button of the FEELMOUSE with the magnets implemented in the FORCEMOUSE. The major drawback of the mouse is that it is not possible to provide *directed* force feedback, although this problem can partially be solved by an intelligent, low-level mouse controller which stops the movements in a specified direction. Of course, this method is not comparable to *active force feedback.*

With a *trackball,* the advantages of both joystick and mouse can be combined. On the one hand, a trackball is a comfortable and universal input device. On the other hand, by adding motors the movement can be influenced in x- and y-direction, and even active output of forces is possible. Because the user's fingers usually rest on one or more buttons, tactile output mechanisms might be added as well.

Regarding software requirements, the results of our own as well as other experiments revealed the drawbacks of simple haptic feedback. A naive approach is clearly not usable with a complex UI (force feedback is "not only a source of information, but may act as a disturbance input as well". [10], see Figure 3), therefore several requirements have been formulated which can only be met by an intelligent control method:

Reaction time: To present the user a haptic feedback which is consistent and coherent with the visual cues, the delay between entering a widget and launching the feedback has to be minimized. (According to [8], the delay between visual and haptic cues must be less than 50 ms: "The cycle time τ_P of the perceptual processor is ... taken to be $\tau_P = 100[50 \sim 200]msec$." and "..., sensory stimuli that occur within one perceptual processor cycle may combine into a single coherent percept.")

Prediction: One method to reduce the delay is to determine the next widget to be used in advance. Consequently, a prediction method which processes the user's interactions in real-time is needed.

Adaptability: Every application has a specialized layout and uses different widgets. In addition, every user interacts with an application in his or her own way. Therefore, we not only need a model of the user's behavior with respect to a specific GUI, but the model must also adapt itself, thus optimizing the system's performance and error rate over time.

Independency: The generated model will be application as well as user specific, but the system must generate it for whatever application and user. In this sense, the system should be completely independent and it should not be necessary to modify the application program.

Versatility: The system architecture should be modular and flexible in order to support the integration of other modalities and the adaptation to different platforms. Moreover, the generated model should be usable for other purposes as well, e. g. for recording macros or generalizing action patterns.

AN INTELLIGENT SYSTEM FOR HAPTIC OUTPUT

The requirements above lead to the development of a system which predicts the next user interaction regarding the usage of widgets in the GUI based on previously recorded data. The goal of this system can be stated as follows:

> *The primary task is to **predict** the next user action in order to give the haptic feedback **selectively** and to **adapt** this capability over time.*

The basic idea of our approach is to create a user and application specific model based on empirical data by 'observing' and analyzing the user's interactions. Both, the creation of the model as well as the on-line analysis is performed by a multi-agent system based on stochastical classification methods. With an adequate amount of input data, the model constitutes the basis for the prediction of future actions.

User Modelling

Several different prediction methods exist from which we have realized two: the first approach analyzes the features of the cursor's trajectory, whereas the second one is a dialog-based approach in which the semantics of the UI and the dependencies of widgets are reconstructed. To yield best results, both approaches which complement each other have been combined.

The prediction mechanism

The basic problem of predicting the user's actions can be described as follows: Given a set of widgets (potential targets) and the user's observable behavior (a *trajectory*), find the widget which is most likely to be used next. This is a typical classification problem where a given input sample—the user's interactions—has to be classified in one of n classes—the potential targets.

A sequence of events $e_i = (x_i, y_i, s_i, t_i)$ which set up a relation between the cursor position (x_i, y_i) and the button state s_i on the one side and a time t_i on the other side is called a *trajectory*. Because the cursor is not moved continuously, it can be divided in *segments* which are usually related to a positioning operation. When the system has sufficient information describing the widgets $\omega_1, \omega_2, \ldots, \omega_n$ of an application, the goal of the prediction mechanism is to analyze a given trajectory $(e_0, e_1, e_2, \ldots, e_T)$ in order to predict the widget $\hat{\omega}$ which is most likely to be the next target:

$$P(\hat{\omega}|e_0, e_1, \ldots, e_T) = \max_{i=1}^{n} P(\omega_i|e_0, e_1, \ldots, e_T)$$

Hence, the first step of the analysis is to extract meaningful features from the GUI as well as from the trajectory. The extraction of the widgets' parameters is comparatively simple: find all widgets which might be used as interaction objects (i. e. remove all 'passive' widgets like frames and labels) and get their size, position etc. (this process is described in more detail below, see *The Spy*). The trajectory analysis is much more complicated, because it covers many features of which only some are relevant in a given context. Features which are usually interesting and meaningful are, e. g., the cursor's velocity, acceleration, the trajectory's curvature, the direction of the enter event etc., see Figure 6. The main difficulty is to extract and construct an appropriate combination of features including adequate information without increasing the complexity too much.

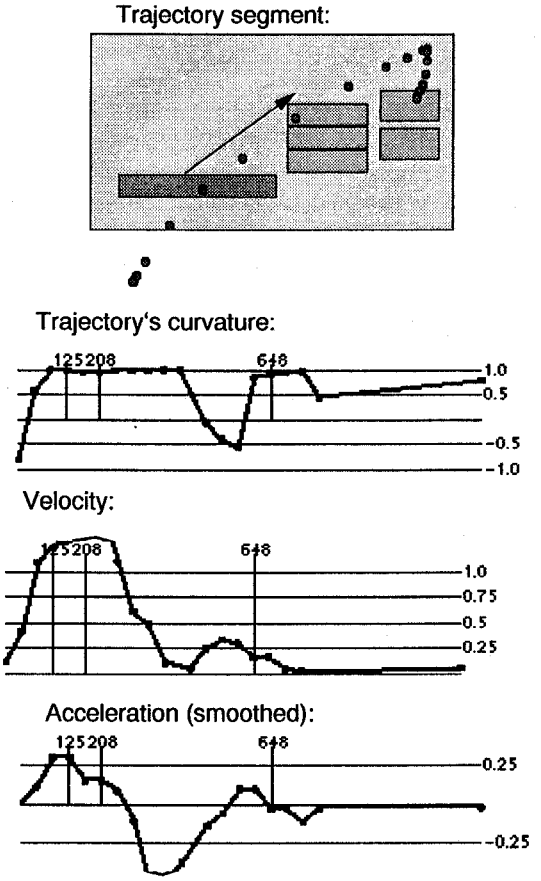

Figure 6: Example of a positioning operation (arrow added by hand). The vertical lines in the diagrams mark the time a widget has been entered by the cursor.

Trajectory-based prediction

The implemented procedure is based on *stochastic classification*. The principle idea is to observe the user's interactions for some time, to collect the features extracted from the trajectories of these interactions, and to create a model of the user's behavior. For each *Enter*-Event, i. e. whenever the cursor enters a widget, the characteristic features of the last part of the trajectory are recorded in two separate models: one model for the usage of widgets (pos), one if the widget has only been traversed ($\neg pos$).

After several interactions, the models will become more and more powerful and can be used to predict the next action by analyzing the current mouse movement and calculating an *a priori*-probability:

$$P_j(pos|\vec{v}_i) = \frac{P_j(\vec{v}_i, pos)}{P(\vec{v}_i)} \approx \frac{f_j(\vec{v}_i, pos)}{f_j(\vec{v}_i, pos) + f_j(\vec{v}_i, \neg pos)} \quad (1)$$

Here, $\vec{v}_i$ is a *feature vector* and f_j a *frequency* denoting how often widget ω_j has been used when the feature vector $\vec{v}_i$ has been observed. In other words, positive (ω_j used) and negative (ω_j not used) examples are simply *counted*.

Although the complete feature space including *smoothed acceleration*, *velocity*, *trajectory curvature*, *stopping distance*, and *overall distance* covers 180 distinct cells, the prediction usually starts to work after a small number of interactions because some widgets are used much more frequently than other ones. This is a great advantage compared to an analysis based on neural networks, which usually need a large amount of training data before reliable predictions can be generated. By storing the collected data (i. e. the models) in a file, it can be used in subsequent sessions for the prediction and it will be refined and adapt with every new user interaction.

Of course, this model can not be perfect and the user's behavior may change, so a threshold θ_j which determines whether to give haptic feedback or not when widget ω_j is approached and feature vector $\vec{v}_i$ is observed is adapted whenever a wrong decision occurs: θ_j is increased if a positioning action has been predicted although the widget has not been used, and it is decreased if it is used but not predicted.

Dialog-based prediction

The second method to predict the next user action has been adapted from speech recognition. Instead of using Equation 1, socalled *trigram probabilities* are used to approximate the *a priori*-probability:

$$P(W) \approx \prod_j P(\omega_j|\omega_{j-2},\omega_{j-1})$$

The underlying idea is to model the *semantics* of the man-machine-dialog. Typically, one user performs a sequence of actions in the same order, e. g. when changing the font attributes first change the size and then the style. In addition, often some actions restrict followup actions to a subset of all possible actions. In these cases, the usage of some widgets is much more likely than the usage of other widgets. The trigram probabilities are an appropriate method to model these interdependencies.

At any time t_j, the widgets which have been used last are $\omega_1,\omega_2,\ldots,\omega_{j-1}$. Thus, $P(\omega_j|\omega_{j-2},\omega_{j-1})$ determines the probability that widget ω_j will be the next target to be used if ω_{j-2} and ω_{j-1} were the last two widgets used. This probability, which is completely independent of the probability calculated in the trajectory-based approach, can be modelled as follows:

$$P(\omega_j|\omega_{j-2},\omega_{j-1}) = \frac{f(\omega_{j-2},\omega_{j-1},\omega_j)}{f(\omega_{j-2},\omega_{j-1})} \qquad (2)$$

Here, $f(\omega_{j-2},\omega_{j-1},\omega_j)$ denotes the frequency of the occurrence of the *positioning sequence* $(\omega_{j-2},\omega_{j-1},\omega_j)$.

Combining both approaches

By introducing a weight λ, the overall probability P_{all} to use a specific widget ω_j can be calculated from a combination of both, the trajectory-based and the dialog-based prediction in Equations 1 and 2:

$$P_{all} = \lambda P_j(pos|\vec{v}_i) + (1-\lambda)P(\omega_j|\omega_{j-2},\omega_{j-1}) \qquad (3)$$

Interestingly, the trajectory- and the dialog-based model complement each other with their different prediction methods: the first one seems to predict rather 'optimistically', whereas the second one shows a more 'pessimistic' behavior. Depending on the user's preferences, the factor λ can be adjusted to favor one of the models.

System Architecture

The most flexible implementation of a software system can be achieved by realizing its single parts as independent modules which exchange data—a *multi-agent system*, see Figure 7. By using PVM (Parallel Virtual Machine [11]) as the underlying software package for interprocess communication, such a system is fairly easy to realize and might be extended with other components with minimal effort.

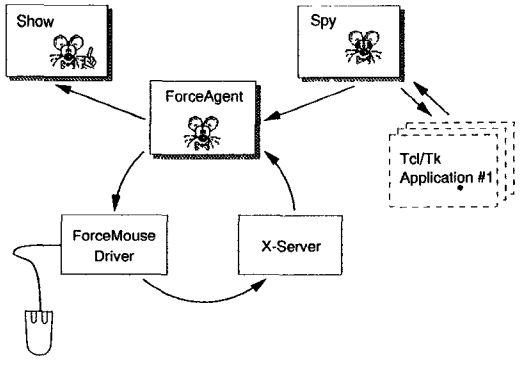

Figure 7: The multi-agent architecture

The central component of the system is the FORCEAGENT. It manages the statistical model, analyzes the user's behavior, predicts the next widget, and launches the feedback by sending a command to the FORCEMOUSE driver. This low-level module provides several commands for different feedback modes and has been integrated in the X Window System to control the standard cursor (not described here).

In order to increase the system's performance and to reduce the FORCEAGENT's load, the observation tool SPY has been developed. This module creates a simplified model of the GUI which contains all necessary data for the FORCEAGENT. Finally, the visualizer tool SHOW has been implemented. It can be used to display the internal models and to record and play back movement trajectories, but it mainly serves as a debugging tool during system development (not described here).

The Spy

The SPY has been developed to 'spy' on other applications. Its main task is to create a *model* of the GUI. Hence, it extracts all relevant features of a Tcl/Tk application, e. g. the uppermost picture of Figure 6 is a model of the application shown in Figures 2 and 3. Tcl is a simple scripting language, and Tk a toolkit for the X Window System providing its own widget set [15]. In principle, the SPY can be used to observe applications using other widget sets based on X, too, but this has not been realized yet.

Besides its many features, Tk provides two mechanisms which are especially useful: event bindings and the send command. An *event binding* allows to associate a script or a program with an event, e. g. the entering of a widget. Whenever the event occurs, the script will be processed. With the send command, this can be done in 'foreign' applications as well: "With send, any Tk application can invoke arbitrary Tcl scripts in any other Tk application on the display; ..." [15]. Although of great power, send is processed *synchronously*. Because the complete widget hierarchy of an application has to be processed in order to collect data for *all* widgets, the delay is not acceptable if this command is executed in the FORCEAGENT which has to fulfill real-time requirements.

Therefore, the SPY checks the display periodically. If a new application is detected, it uses the send command for each widget in order to get the widget class and the X-ID, which are both sent to the FORCEAGENT via sockets by using PVM in an *asynchronous* mode. With this information, the FORCEAGENT can observe the widget's lifecycle with minimal effort. In addition, the SPY installs new event bindings in all applications which send information whenever a new widget is created, an existing one is destroyed or the state of a widget changes. Thus, every application has to be processed only once, whereupon configuration changes are reported automatically.

The ForceAgent

The FORCEAGENT decides *when* to apply *which* kind of haptic feedback. The decision is based on the application model, the trajectory-based model, the dialog-based model, and the preferences of the user.

Managing widgets Relevant widget attributes like size, position, status etc. are received from the SPY and managed internally to enhance the performance.

Managing user inputs All user input is received by the X server and sent to the respective clients as *events*. Although there is no common structure for all events, the attributes *x/y-position*, *time stamp*, and *window of occurrence* (X-ID) are always included.

In a first preprocessing step, the FORCEAGENT determines the *relevant events* from the sequence of all events. Subsequently, the event data are converted into a special data structure *trajectory* for further processing. This structure has been developed and implemented in order to support direct input and access in $O(1)$. In addition, it allows easy filtering, processing, and segmentation of the trajectory.

Managing interaction models All trajectory and dialog specific data sampled during the interaction process are recorded in *interaction models* which are stored in order to be used in subsequent sessions. Thus, an overall user and application specific model is created which adapts to the user's needs over time.

The decision process The FORCEAGENT's main task is to support the user during positioning tasks by giving haptic feedback when needed. The decision *when* to trigger is determined by the overall probability P_{all} calculated by the prediction mechanism, see Equation 3. The decision *which* kind of feedback (F_{mode}) to use is determined by the user—several different modes and combinations can be selected via the FORCEAGENT's user interface. The final decision rule can be written as follows:

$$\textbf{\textit{when}}\ (P_{all} >= \theta_j)\ \textbf{\textit{if}}\ (F_{mode})\ \textbf{\textit{then}} \rightarrow \text{haptic output}$$

RESULTS

The system has not been evaluated in detail so far, but nevertheless some preliminary results will be presented. The main purpose of the FORCEAGENT is the prediction of the next interaction object ω_i which might be wrong in two senses:

1. ω_i *is predicted* but the user did not intend to use it.
2. ω_i *is not predicted* although the user selects it.

The haptic feedback is only an *additional modality* which should support the user during positioning (or target acquisition) tasks. Therefore, the second type of error is a minor problem only, but not critical. In contrast, the first type of error has to be minimized because every time the mouse is stopped when the user does not want it to stop, the user is actively hindered to complete the desired task (see Figure 3).

Although an evaluation of the system's performance is difficult due to the fact that many different factors will influence the result, first tests with simple applications and a short training phase have shown a mean error rate between 5–25%, with most errors belonging to type 2. With the default parameter settings, the error rate for type 1 errors was less than 5%. More important, the error rate drops down when the system is used for a longer period.

CONCLUSIONS AND FUTURE WORK

We have presented a way of supporting direct manipulations with a mouse by introducing haptic output. A sophisticated mechanism predicts the next target of the current cursor movement and stops the mouse at the desired position. Thus, positioning tasks become faster and safer, especially when target regions are very small. In order to give the best support, the system generates a user and application specific model and improves its capabilities over time by adapting the models and thresholds.

The multi-agent system realized to support the prediction of user interactions creates two independent models: First, a trajectory-based model reflecting the user's way to move the cursor and to interact with a specific application is generated. Second, part of the application's semantic is modelled by analyzing the order in which the widgets are used (dialog-based approach). Both models do not fully cover the user's behavior but are complementing each other, so that the combination leads to good results in the prediction of the next interaction object.

The system's design has two major advantages. First, it is a *plug-and-play solution* which is fully compatible with any Tcl/Tk application (versions 7.4/4.0 or later) running under X. In order to use the haptic modality, the multi-agent system runs in the background, identifies Tcl/Tk applications, and

modifies them at run-time without affecting the source code. The second advantage is its flexibility. Although designed to support haptic feedback, it might be used for other modalities as well. In addition, the system is not perfect but works well with a very cheap and simple device.

On the other hand, there is still some work to be done. First of all, the system's performance has to be evaluated under various conditions. This will be done best by letting several users try the system and ask them to judge it. Unfortunately, the current version is more a kind of prototype which shows that the approach works, but which has some drawbacks regarding the performance (e.g., the FORCEMOUSE moves not smooth enough on the iron pad and not all Tk widgets have been included in the implementation.)

Although the benefits of haptic feedback have already been demonstrated in isolated tasks, for complex systems the prove is still missing. In this context, an improved version of the system could be realized with a better device than the FORCEMOUSE—see section 'Hard- and Software Requirements'. An interesting feature not provided by the FORCEMOUSE is a voltage regulator for the electromagnets which could be used to map the probabilities of the prediction mechanism to the strength of the magnets.

It would also be interesting to develop and integrate a module for prediction based on artificial neural networks. Especially time delay neural networks (TDNNs) could perhaps lead to even better prediction results than the stochastic method used so far. Unfortunately, TDNNs—like other neural networks—need a very large amount of training data before the prediction works.

Other open questions are whether a model for one user can be used for more than one application and whether an application specific model is useful for several users. Maybe a method exist to generate a kind of 'universal' default model for all users and applications instead of starting from scratch? If this is possible, the model would still adapt to specific interaction schemes but would be working from the beginning without the need for an initial training phase.

Finally, new system versions which could support other widget sets based on X (e.g. OSF/Motif) would be attractive in order to cover a broader range of applications. The ultimate solution would be a system which is independent of any widget set but uses X functions only to create the GUI models and to observe the user's interactions.

ACKNOWLEDGEMENTS

This work has been funded by the European Union, ESPRIT BRA project No. 8579, Multimodal Integration for Advanced Multimedia Interfaces (MIAMI). It has been performed at the Institute for Real-Time Computer Systems and Robotics, Department of Computer Science, University of Karlsruhe. Parts of this work have been previously published in [14], copyright Eurographics Association. The authors would like to thank M. Stangenberg for his support during system development and the reviewers of this article for their constructive criticism and valuable comments.

REFERENCES

1 M. Akamatsu and S. Sato. A multi-modal mouse with tactile and force feedback. *Int. Journal of Human-Computer Studies*, 40:443–453, 1994.

2 M. Bevan. Force-feedback technology. *VR NEWS - Virtual Reality Worldwide*, 4(6):23–29, July 1995.

3 K. R. Boff, L. Kaufman, and J. P. Thomas, editors. *Handbook of Perception and Human Performance*, volume II. John Wiley and Sons, 1986.

4 T. L. Brooks. Telerobotic Response Requirements. In *Proceedings of the IEEE*, pages 113–120, 1990.

5 F. P. Brooks, Jr. et al. Project GROPE - Haptic Displays for Scientific Visualization. In F. Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 177–185, Dallas, TX, 1990. ACM.

6 G. Burdea and P. Coiffet. *Virtual Reality Technology*. John Wiley & Sons, Inc., 1994.

7 W. Buxton. Human Skills in Interface Design. In L. MacDonald and J. Vince, editors, *Interacting with Virtual Environments*, chapter 1, pages 1–12. John Wiley & Sons Ltd., 1994.

8 S. K. Card, T. P. Moran, and A. Newell. The Model Human Processor — An Engineering Model of Human Performance. In *[3]*, chapter 45. 1986.

9 F. L. Engel, P. Goossens, and R. Haakma. Improved Efficiency through I- and E-Feedback: A Trackball with Contextual Force Feedback. *International Journal of Human-Computer Studies*, 41(6):949–974, 1994.

10 W. R. Ferrell. Delayed force feedback. *Human Factors*, pages 449–455, October 1966.

11 A. Geist et al. *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, London, 1994.

12 W. Kerstner, G. Pigel, and M. Tscheligi. The FeelMouse: Making Computer Screens Feelable. In W. Zagler et al., editors, *Computers for Handicapped Persons. Proc. of the ICCHP'94*. Springer-Verlag, 1994.

13 J. M. Loomis and S. J. Lederman. Tactual Perception. In *[3]*, chapter 31. 1986.

14 S. Münch and M. Stangenberg. Intelligent Control for Haptic Displays. *COMPUTER GRAPHICS forum, Conference issue (Eurographics '96, August 26–30, Poitiers, France)*, 15(3):217–226, 1996.

15 J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, 1994.

16 H. Rheingold. *Virtual Reality*. Simon & Schuster (Towerstone Books), 1991.