ABSTRACT
In this paper, we describe the development of a new, non-haptic user interface for IBM-compatible PCs. The results of development itself have been demonstrated at a computer graphics conference [3]. The advanced user interface consists of a combination of spoken commands and head movements. It translates spatial and symbolic input into the traditional mouse, keyboard, and system events.

Keywords
New Input Devices, Speech, Gestures, User Interface Design

1 INTRODUCTION
This paper will show some of the details about how we work on realising an idea and how we have achieved our results in this specific design case. We do not claim that our methods are new or better than others, we simply want to describe this case honestly.

The starting point of our efforts was the observation that a significant number of users are physically not able to use today's WIMP-style (Windows, Icons, Menus, and Pointers) Graphical User Interfaces (GUIs). The reasons, as far as input modalities are concerned, include hands-busy situations (e.g., a mechanic at work), paralysis, or bad neural control of body movements.

Our goal was to overcome these difficulties in the practical use of existing software applications, and demonstrate results for some exemplary applications. We have chosen two very different examples: A commercial shopping catalogue on CD-ROM and a common painting program. These programs should be navigated in an effective manner without touching the traditional input devices. In our example this was solved by simulating mouse and keyboard events by speech input and head pointing gestures. The earliest version was demonstrated in August 1996 [3].

For more information about the use of speech and gestures in user interfaces, the reader is referred to the references [5, 6, 7, 8, 9, 11, 12].

Permission to make digital/hard copy of part or all this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

DIS '97 Amsterdam, The Netherlands
© 1997 ACM 0-89791-863-0/97/0008...$3.50

2. THE SUBTASKS OF THE DESIGN-PROCESS
The general task of integrating new input modalities into an existing system for a specific purpose can be divided into several subtasks:

- Finding the right input modalities that match the purpose at a minimum of overall costs.
- Defining concepts for technical integration that match the technical resources and programming skills of the developers team.
- Making a plan for implementing functionality and for testing it with users.
- Programming and testing in correspondence to the above concepts.

Unfortunately, these tasks are generally not independent from one another. What makes it worse: Our knowledge of technological possibilities, human factors, and even about our own resources – in terms of skills and person-power – is naturally incomplete, especially at the beginning stage of any project. In our example, if we choose an input modality that requires urgently the implementation of a hardware interrupt handling routine, we may have to live with the need for assembler language programming. This decision does not only require special programming skills, but also makes the development more hardware-dependant.

The following subchapters deal with the decisions taken within the subtasks and shall make transparent some of their interdependencies.

2.1 Choosing Input Modalities
Many ways of generating alternative (i.e., non-haptic) input to machines have been discussed since the beginning of the computer age. Among them are all kinds of signals emitted by humans: spatial, electrical, visual, acoustical, chemical, and thermal signals. Let us summarise them under the term "biosignals".

As stated above, we want to realise an interface at a minimum of costs. Two kinds of costs have to be taken into consideration: End users cognitive and monetary costs, and the costs of development for the interface. The questions are simple:
a) Which combination of biosignals delivers the optimal interface?
out that this demand was not always easy to fulfil

platform-specific object-libraries. All programs were meant
documented API-functions or messages, as well as any
common electromagnetic tracking device \[2\].

only one of the additional input channels, which may be
conceivable for, let's say, a painting program.

What remains after a closer inspection is mainly speech,
spatial signals, and visually analysed gestures and viewing
direction. To fulfil the requirement of cost minimisation we
have decided to start with speech recognition, serving as
symbolic input, and a spatial signal that should be mapped
to cursor movement. For both modalities there already devices and development tools on the market for PC
peripherals \[1, 2\].

2.2 Concepts of Technical Integration

The decisions about the methods of integration are natu-
really ruled by what the target system, the developer tools,
and the hardware-interfaces to peripherals deliver. But a
sum of restrictions is of course not yet a concept: A good
concept takes into consideration such things as

- modularity in programming,
- redundancy in competence, and what we would call
- "preference of main stream methods and components".

All of these serve for the long-run feasibility of a larger
development.

These thoughts in mind we have decided to build our inter-
face with the help of a development system for single-word
speech recognition \[1\] by the market leader, and the most
common electromagnetic tracking device \[2\].

Both modalities are processed by individual programs. This
concept makes it easier to exchange parts of the whole
interface with new devices. In addition, it is possible to run
only one of the additional input channels, which may be
sufficient in some application cases. The same arguments
apply to the internal modules of programs: Modules should
be reusable, and what is specific to a certain device or
application should be kept apart from other things. It turned
out that this demand was not always easy to fulfil
(especially when members of the team have their own mind
with programming...).

For software development it was decided to use the C lan-
guage (as far as possible), and to avoid the use of any un-
documented API-functions or messages, as well as any
platform-specific object-libraries. All programs were meant
to be run on the system's application-level, provided that
the results were acceptable in terms of speed and stability.
To explain this to non-programmers: The application level
is the "highest" level, meaning that those programs have
the lowest priority in the time sharing process of the
machine. Hardware related or time-critical processes, like
keyboard or mouse event routines, usually run on a lower
system level.

2.3 Plans for Implementation

Because users are not machines, and there seems to be no
way of simulating their behaviour adequately, it is very
important to have early feedback in the development of
interactive systems. This gave us a sort of guideline for
implementation: The first step has been to have the com-
plete input pipeline working, even if any comfort would be
missing. The second step has been an iterative loop of
adding and changing features and filters, testing them, and
deciding whether to keep them or not.

Most of the testing has been done by team members on
their own, but it was also helpful to have external test-users
from time to time. This is because programmers become
blind in some sense to the real needs of users, i.e. they get
used to some curious details they have produced.

We did not commit to a global time schedule. Instead we
have had regular team meetings, where all suggested
improvements and next steps were collected, sorted by
priority, and than assigned to specific persons. This has
shown a motivating effect on the team and helped keeping
individual worktasks in accordance with global goals.
Another method of motivation has been to benefit from
natural external deadlines, like exhibitions and confer-
ces. Programmers naturally want to show their best stuff
to the public, so that finishing in time becomes almost a
question of personal honour.

2.4 Programming and Testing

2.4.1 Programs

The result of the iterative development process explained
above, consists of two main programs. One of them inter-
faces between the speech recognition kernel program and
the application programs (see figure 1). The second one
processes the raw signals delivered by the tracker hardware
and directs the mouse cursor to the estimated position (see
figure 2). Without going deep into the actual programming
we want to explain the basic principles and difficulties of
both in the next paragraphs.

Our speech input works with single word commands and
short phrases. User must speak command phrases without
pauses between words. The speech recognition kernel
works on a static vocabulary of such words and phrases. It
compares an incoming utterance to the pronunciation
models of the words in the active vocabulary, and returns a
list of the closest matches. Words are returned as ID-codes,
which have to be defined by the developer when generating
the vocabulary. Our program currently uses only the top
scoring word from the returned list of candidates.
On top of this sheer recognition of utterances comes, what is most of the labour when making applications become speech-aware: Linking the recognised word to the right action, and performing that action automatically, i.e. give a meaning to a word. This may sound easy, but it is not, because the graphical user interfaces of most applications are dynamic. That means, a command can have different meanings (or no meaning at all) within an application depending on the inner state of the application!

Therefore, the only solution left for us was off-line analysis of the complete user interfaces by going through all possible states of the applications and manually listing all the interactive objects found on each screen. Of course we cannot always assure completeness for this task. The following step was "hacking" of routines to estimate the state of the application at runtime, e.g. following rules like "if listbox xyz is visible, the current state is 123".

We feel that, for the future development of multimodal input, it would be very valuable to have a formal description of the user interface behaviour. Along with that, a communication protocol for enquiries about the current state and about legal user transactions would be very helpful. This could be solved in analogy to [13], who suggests a rule-based design specification notation (DSN) for the simulation of device interfaces (e.g. CD-players). In the computer simulation, the device interface is aware of its state and permanently updates a list of all possible user transactions. With such a list it would be possible to "synchronise" the state behaviour of the speech driver and the applications automatically.

By displaying such a list of possible actions on demand to the user, it would be also easier to use spoken commands. The user in this case does not have to remember all the command words, he or she just reads them aloud or calls them by an index, e.g. "take 7"

In the case of the head-tracking device such windows-related problems do not occur. Triggered by a timer the program reads raw position data from the Polhemus plug-in board and converts it to float values. In the next step a line orthogonal to the users face is calculated and intersected with the plane of the display monitor. The result is a point on the display-plane which gives a raw signal for the cursor co-ordinates. What follows is some smoothing, gain adjustment and activation switching. The last thing in the processing chain is the actual placement of the mouse-cursor (see figure 2). The automatic activation switch is based on the absolute position of the tracking probe and thereby enables automatic switching between mouse and head-based cursor control when taking off the headset or moving away from the display.

It may be interesting to mention, that the mouse events called via API-functions are not the same as the hardware-based ones; for instance it is not possible to send a "click" sequence (button_down, button_up) to a screen co-ordinate and let the system decide, which object shall receive it. You have to look up first, which object is on top at the namely position, find out its process ID, and send the "click" events directly to it.

2.4.2 Usability tests

Until now the advanced user interface has been tested by team members, employees from other departments, and visitors of the Institute. A testing phase with disabled users will follow in springtime this year. Most of the new users were surprised by the quality of the head-pointing facility.
Thanks to a recursive smoothing algorithm, it is possible to keep the cursor fixated within a 5x5 pixel area under normal conditions. Also the speech recognition input shows convincing results. The recognition rates are above 95%, so that spoken hotkey-commands, e.g. to menu functions like "open" or "save", are not just equal to mouse-clicking, but give a real enhancement in working speed.

It must also be said that each modality itself can be helpful in combination with mouse and keyboard, for those who are able to use these devices. For instance, changing the painting tool via speech spares you the need to leave the painting area, click on a specific tool icon, and return to the painting.

3 Conclusion and Further Work
We have experienced, that it is possible to integrate new input media with existing applications at moderate costs. With our flexible and modular concepts we have achieved our goal to build a non-haptic interface to common, commercial applications.

Our efforts have uncovered that nowadays operating systems and graphical user interfaces are not well prepared for the integration of speech recognition or other alternative sources of input. The root of this problem is the lack of any explicit descriptions of user interactions, application's behaviour, and its current state.

In the future our next goal is to let the user interact in a more natural style. Without any cabling he shall be able to use the computer with spoken commands along with natural pointing gestures, performed alternatively with his head or hands.

4 ACKNOWLEDGEMENTS
We thank Prof. Bertram Herzog for his help, especially with the English version of the speech interface. Farhad Hamedvaran, Jürgen Effenberger, and Robert Swaczyna did most of the programming.

5 REFERENCES
6. Martin Bichsel, Alex Pentland: Automatic Interpretation of Human Head Movements, International Joint Conference on Artificial Intelligence, IJCAI, 1993
12. U. Bröckl-Fox: Real-Time 3-D Interaction with up to 16 Degrees of Freedom from Monocular Video Image Flows, from [10], pp. 172-178