Design and implementation of a user-oriented speech recognition interface: the synergy of technology and human factors

Sietse H. Kloosterman

The design and implementation of a user-oriented speech recognition interface are described. The interface enables the use of speech recognition in so-called interactive voice response systems which can be accessed via a telephone connection. In the design of the interface a synergy of technology and human factors is achieved. This synergy is very important for making speech interfaces a natural and acceptable form of human–machine interaction. Important concepts such as interfaces, human factors and speech recognition are discussed. Additionally, an indication is given as to how the synergy of human factors and technology can be realised by a sketch of the interface’s implementation. An explanation is also provided of how the interface might be integrated in different applications fruitfully.

Keywords: human–computer interaction, interface, speech recognition, interactive voice response system

Natural interaction
It is a well-known trend in computer science to adapt computers to suit their users and not vice versa. The exploitation of more human senses in human–machine interaction than just sight can be proved to be more efficient and natural.

One way of enhancing human–machine interaction is to supply an additional response channel by means of speech recognition. Simply formulated, speech recognition can be regarded as the automated ‘understanding’ of spoken words. In the third section we describe speech recognition in more detail.

Speech recognition allows interfaces to be built that permit users to interact with computers using natural language. By doing so, it widens the bandwidth of the communication channel that exists between user and computer. However, speech recognition is not a panacea. Some input/output tasks (e.g., numerical data entry) are such that using SR could decrease the interface
performance. Furthermore, natural language can be ambiguous. It is of great importance, therefore, not to implement speech recognition in each and every application, but to use it carefully. For instance, in critical applications such as nuclear plant control, key-entered input would be more appropriate.

It is also unwise to conclude that 'plugging in' a speech recognition device offers the best way of communication, even though the situation seems suitable because using speech recognition draws on aspects in the domains of linguistics, cognitive psychology, etc. It is not enough just to present the user with the opportunity to use a speech recognition program without taking human factors into account. Not supporting the user in an appropriate way during the interaction process will diminish the benefits of speech recognition. If, on the other hand, human factors are taken into account, a natural communication channel can be created.

Application area
Interfaces that provide speech recognition can be built to support direct communication between a computer and a user — who is directly connected to the computer with a microphone headset — as well as to support remote communication. In the latter case, the interface must be equipped with some communication device, too.

The interface described in this paper supports remote communication in a voice response system (VRS). Using a telephone set, a user (caller) connects to a computer system. The user enters his or her input via speech recognition (or the keys on the keypad) and the computer produces output using playback of digitized speech. When, in addition to playback features, a true dialogue emerges, the voice response system is called an interactive voice response system (IVRS). Such systems can be put to use in many different applications, e.g. tele-shopping, automated attendance, voice mail, fax response and credit card authorization. In order to support speech recognition facilities in a telephone-based VRS, a telephone handling device is needed. Such a device should be able to perform all telephonic interactions. In normal life, people automatically make telephone calls and are unaware of the actions needed to complete such a task. These actions (such as taking a phone offhook, putting it onhook, detection of loop current drop and number dialling) can be performed by a telephone handling device. See Figure 1 for a sample VRS configuration.

Features
In the field of speech recognition numerous applications have been built for functions such as data entry, command and control, and database access. These speech recognition functions are employed in industry, hospitals and laboratories, offices, and in military and public applications. The interface described in this paper differs from other speech interfaces in several ways:

- the interface is separated from the application;
- it can easily be added to existing telephone-based IVRSs;
- it combines human factors and technology;
- the interface is generic in the sense that it is not designed for a specific IVRS (see below).
The interface offers the functionality of question-and-answer dialogues with respect to menu selection, yes-no questions, and digit entry. Any voice response system that has an appropriate architecture can, in principle, be extended with the speech recognition interface to comprise one of the functions mentioned before. So, the interface can truly be called a generic one. As a matter of fact, the interface has been used in the construction of an information retrieval system and in a prototype of a 'putting through' application.

**Overview**

The remainder of this paper is structured as follows. First we describe some basic concepts, then in the next section we describe the interface concept. Speech recognition is discussed further in the third section, and then in the fourth section we address human factors. Design considerations of the interface are discussed in the fifth section. In the section that follows, some implementation aspects are addressed. Then, in the seventh section we illustrate how applications can be built using the interface. After that, the interface and its implementation are reviewed in the eighth section. Finally, some conclusions are drawn.

**Interfaces**

**Definition**

Formally, an *interface* can be defined as a link between complete systems — hardware or software — so that the whole functions as an integrated system, or it can be defined as a specification of communication between two programs. Hence, an interface is a medium (wire, software, intelligent hardware) providing a channel through which different entities can communicate. In some
situations, an interface just transfers an electrical signal through a wire or coaxial cable, while in other situations an interface translates information into a different format, amplifies information, or performs another action.

In the case of the interface described in this paper it provides a connection between an application program and a user via a speech recognition device. By doing so it achieves a widening of the information channel between the two interacting participants. When a user calls the IVRS, an interface task is initiated. The interface then starts processing, communicates with the user and, at the end, will present the result of the previously initiated task to the application.

Several viewpoints
In general, the interface concept has several dimensions. Basically, an interface can either be described from a physical or from a cognitive point of view. From a physical point of view it establishes the physical link between user and computer. In the case of IVRSs it comprises telephones, connections, telephone exchanges, speech recognition devices, telephone handling devices and computers. In Figure 1 such a physical configuration is sketched.

The physical part of the interface supplies a service supporting the cognitive part. From this cognitive point of view, the interface allows for a high level communication between user and computer, as illustrated in Figure 2. The software implementation of the interface takes care of this cognitive part. To be precise about the cognitive interface, it can be divided into a semantic and a syntactic part. Whereas the semantic part is concerned with task completion and related concepts (what to do), the syntactic part is concerned with how a specific dialogue comes about (how to do it).

In this paper aspects of both angles (physical and cognitive) are discussed.

Speech recognition

Speech recognition defined
Speech recognition can be considered as the process of transforming an analogue speech signal into discrete representations which may be assigned proper meanings, and which, when comprehended, may be used to effect responsive behaviour. The ultimate goal is to understand the input enough to be able to select and produce an appropriate response. This transformation from analogue speech signals into discrete representations is done by a speech recognizer which is attached to a host computer. Such a transformation is necessary because computers can only manipulate binary data.

When a user (i.e. a caller) is prompted by the application to say a word, the recognizer identifies the utterance, assigns a proper (predefined) meaning to it and presents an indication of the recognized word to the application. The application in turn is responsible for the appropriate processing of the supplied word.

Classification
Basically, speech recognizers can be classified into four classes: a recognizer
recognizes isolated words or continuous speech and it can be speaker-dependent or speaker-independent. *Isolated word recognizers* require a short pause before and after each utterance (a word or a phrase), which is interpreted as one entity. When, on the other hand, *continuous speech* is involved, no pause is needed and several words or phrases (each considered to be different by the recognizer) may be run together. So the user can speak whole sentences in a natural way, but they do need a silence at the beginning and at the end.

Dedicated isolated word (or discrete) recognizers are available and can be used in real, commercial applications. This is not the case with continuous speech recognizers. Some prototypes of this kind are available, but there are still problems to be solved before these recognizers will be capable of recognizing continuous speech smoothly. For instance, word boundaries are difficult to detect and semantic analysis is needed to interpret a sentence correctly.

A *speaker-dependent recognizer* is only capable of identifying one person's voice. This is in contrast to a *speaker-independent* system which, in principle, is able to recognize any person's voice. Speaker-dependent recognition can be achieved with simpler systems than speaker-independent recognition.

As we shall explain later, we use a speaker-independent isolated word recognizer for our interface.

**Tokens, templates and vocabularies**

The speech recognition process can be divided into two steps:

- generating a representation for the utterance to be identified,
- comparing the generated representation with all known utterances.

The generation of a representation for an unknown utterance results in a so-called *token*. The representation of a known utterance on the other hand is called a *template*. All templates form a set, named the *vocabulary*. The process of recognition comes down to comparing a token with all the templates in the vocabulary. The template that matches best with the token is considered to be the pronounced word.

In order to be able to recognize words, these words must be taught first. The process of teaching is called training. When a speaker-dependent vocabulary has to be learned, it is sufficient to train the recognizer with a couple of samples from just one speaker. As consequence, the system cannot recognize another's voice with much confidence. For a speaker-independent recognizer, more people have to be involved in the training process to construct a vocabulary the words of which can be identified without regard to the actual speaker.

In some communication sessions, not all words need to be under consideration at the same time. To reduce the chance that a spoken word is matched with a vocabulary entry which turns out to be a mismatch, it is possible to exclude temporarily a number of words from the vocabulary. To this goal, a vocabulary is split into various *subvocabularies*, which may overlap. By selecting the correct subvocabulary at a certain moment, one can reduce the set of possible recognition candidates and thus reduce the chance of picking the wrong word from the vocabulary, provided that users only say one of the words in the selected subvocabulary.
Human factors

Perspective
Human factors engineering is also known as ergonomics. This is an applied science that incorporates theories, methods and results from other sciences. It gathers, develops and consolidates scientific knowledge necessary for the design of machines so as to improve the working or use conditions and the environment, that is, to increase reliability, performance and satisfaction, while at the same time decreasing stress and discomfort.

In the 1950s, ergonomics was characterized by major influences of psychology and medicine. Later, it was affected by physiology, cognitive psychology and then by linguistics. The latter development originated from the fact that human-computer communication was beginning to use means of interaction based on natural language. The use of speech recognition is derived from this trend to integrate natural language aspects into computer interfaces.

Human factors in the domain of speech recognition address several aspects. These aspects concern recognition accuracy, vocabulary size, response time, dialogue character, communication constraints, user experience, feedback, error correction, type of executed task etc. In the following subsections we shall discuss several aspects of input and output, the use of protocols and aspects regarding the way of interaction. This division of human factors should be put in the context of the physical, syntactical and semantical viewpoints respectively, of the interface concept described earlier.

Input and output
Input aspects are heavily related to vocabularies. Users want to express themselves as naturally as possible, using as many words as they want. Therefore, vocabulary size is of great importance. Too few words in a vocabulary makes a dialogue very unnatural. On the other hand, too many words may endanger the vocabulary performance. Not only is training a tiresome and lengthy job, it may be difficult to ensure that the templates are distinctive enough to result in reliable recognition.

Apart from vocabulary size, accuracy is another important aspect. Using a badly-trained or ill-designed vocabulary results in many recognition rejections. The correction of these errors makes the dialogue unnecessarily complex and disturbs the user's train of thought.

In a later section some 'rules of thumb' are given with regard to constructing a usable vocabulary.

As to the contents of a vocabulary, user input can be divided into data entry input and sequence control input. During interaction the user should know the control commands and the moments they can be used. Ideally, both types of input should be accepted at any time.

As stated in the introduction, the output of an IVRS consists of speech messages, but there are also systems which produce other types of output, such as fax messages. The voice messages are composed of pieces of prerecorded digitized speech. As this limits the flexibility of interaction, other ways of speech generation are developed. One alternative is text-to-speech (TTS) conversion,
but no entirely satisfactory TTS converter is available at this moment, so we use the conventional way of speech generation, that is: using recorded digitized messages.

The time between the moment a word is spoken into a VRS and the perception of the related response is called the *response time*. This response time typically depends on vocabulary size and the complexity of the application. Of course, the response time must have an acceptable maximum value, otherwise the interface is not transparent for the user. In the worst case, a user may repeat his or her input which can lead to undesirable misrecognition. For the speech recognizer used in this paper, the dependency on vocabulary size is linear:

\[ \text{resp\_time} = 11.88 + 2.09 \times \text{voc\_entries ms} \]

Thus, if 100 vocabulary entries is typical for an application, the recognizer response time will be 0.22 s, which we consider to be acceptable.

Output messages must provide the user with appropriate feedback. Feedback should be followed by an opportunity for the users to correct their input. Both feedback and error correction must be presented in a consistent way.

**Protocol**

In order to synchronize the user and the speech recognition system, rules are needed. These rules specify when a user is allowed to say something, which action the system should take in reaction to the user's input etc. A set of rules describing action and reaction is called a *protocol*. Often a protocol must be known beforehand by both parties, otherwise efficient communication is not possible. This, however, would not work in a speech recognition dialogue; explaining to the user in advance what to say and when to say it will confuse the user. It is almost impossible to keep all rules mind, because at each point in communication a specific and different rule will apply. Hence, the interface must rely on *implicit* rules.

Although using an implicit protocol, the system has to make it very clear to users when they are allowed to say something and what exactly they can say. This timing can be done using beeps (high tones) and buzzes (low tones). A beep signals the user to make a choice. In the case of recognizing several words at once, the first beep can signify that input is expected, the second one confirms the first input and asks for the second one, and so on. A buzzer can signal to the user and that the previous input was not correct and another input is awaited.

Although some users do not like beeps and buzzers, these sounds have several advantages. They identify the voice response system as mechanical rather than human. When a user hears a beep, he or she knows that input is expected and, thus, beeps 'provoke' speech. Furthermore, beeps are quicker than spoken messages.

Apart from timing, a user would like to know what his choices for response are. It is very tiresome to listen to an introduction when one is already familiar with the system. Therefore, a user is considered to be an 'expert' until proven otherwise. If it turns out that a user is a novice to the system, tailor-made messages are played to help the user make his or her choice quickly and easily.

*Kloosterman*
Way of interacting
It is desirable for human-machine interaction to be fluent; interaction should follow the user through the dialogue. At the same time, the user must be guided through the interaction in an active fashion. This means that it is very important not to react to the user's input in a fixed way. A user will become frustrated knowing he or she is making a mistake, but not knowing which mistake and how to correct it. A fixed reaction does not take a history of the dialogue into account and relies too heavily on the interaction context. In traditional terminal interfaces, the context of a dialogue is clear and visible on a screen. In telephonic interactions, however, the context resides in one's short-term memory, so standard communication is not appropriate for telephone-based voice response systems. An inflexible dialogue is likely to diverge, that is, to fail. To accomplish a converging (i.e. correctly terminating) dialogue, rather than a diverging one, appropriate feedback is needed, together with error recovery facilities. However, it is necessary to incorporate some degree of consistency in the interface, so that it acts in a predictable way.

The purpose of an interaction process is to discover what a user want. Users can express what they like by using speech; their voices are the medium to express what they want. So, a speech recognition interface should concentrate on what a user wants (be user-oriented) and not too much on what he or she says (speech-oriented).

Disregarding this issue can lead, for instance, to the following situation: a user answers the question "Do you want help?" with "Yeah", and this answer results in a recognition rejection. To verify the user's response, a question such as "Did you say yes?" is asked. This question will invite the user to answer with a "No", which is the correct answer to the question in itself — the user actually said "Yeah", not "Yes". But it is the wrong answer with regard to what the user means; he or she wants to express that he or she answered the question, "Did you say yes?" positively. To prevent situations like this, the system should question the user with, "Did you say you wanted help?", allowing the user to restate the answer to the original question.

While it is important for the speech recognizer in itself to focus on what a user says, it is to the benefit of the dialogue for the interface to focus on what a user wants. User-oriented interactions concentrate on the goals that the users want to achieve and stress forward reasoning, which stimulates the users to think about their will, and not backward reasoning, which stimulates users to think about their words.

User modelling
In order to build interfaces that are easy to use and to learn, it is important to know what users know about the interfaces they are using. If one knows how a user interacts with an interface and how he or she perceives it, one is able to predict the user's behaviour in a particular situation and one can design interfaces which anticipate the observed behaviour. This knowledge can be put in the form of the representation a user has of the operation of an interface. Such knowledge is often referred to as a 'mental model'. Mental models can be made
part of an interface so that the latter can adapt itself to the observed behaviour of users.

Although it is difficult to construct and use correct and adequate mental models, we experimented with the integration of a network model in the designed interface. A network model assumes that the user perceives a computer system as a machine which, at each moment, is in a certain state — a so-called state machine. The system, i.e. the interface, shifts to another state if it is triggered by some action. State transitions occur if either the user or the interface performs an action. Examples of such actions are the pronunciation of a word by the user, or a request made by the interface for input on a time-out signal. Using this model, the interface can, for instance, switch from an ‘expert mode’ to a ‘novice mode’ when a user makes too many mistakes in a row. We discuss the implementation of state machines in a later section.

**Design considerations**

**Synergy of human factors and technology**

In the previous sections we discussed some important aspects of human factors and speech recognition technology. We believe that by combining the described technology and human factors, the result is more than the sum of its parts. The synergy of human factors and speech recognition technology produces an interface that enables a natural and acceptable form of human-machine communication. We think that this synergy is inevitable for such an interface.

Linking human factors with technology requires an alternative approach to interface implementation. Here we highlight the three concepts we used to realize the synergy of human factors and technology.

**State machines**

To use a network model we implemented state machines in C-code. State machines represent not only the mental model the interface is equipped with, but they also realize the interface’s interaction; in each state an output message is generated, or user input is expected.

Furthermore, state machines provide a good means to tackle the complexity of the implementation of the interface. State machines are described later.

**Multitasking**

We already stated that response time should have an acceptable maximum value. To ensure this in a multi-user environment on a computer platform supported by a single-tasking operating system, we have implemented a multitasking feature. In this way, several users can share one service without being delayed.

**Vocabulary building**

The vocabulary quality is very important for speech recognition interfaces; it addresses many human factors. Therefore, we give some heuristics for constructing a satisfactory vocabulary in a later section.
Programmer's view
To encourage application programmers to use a speech recognition interface, its implementation must be easy to use for them. This implies that it must be simple to customize the interface to specific needs. Therefore, we require that the system has just a few parameters that have to be set. The most crucial parameter is the one describing the dialogue for each telephone line between user and computer, specifying which questions are presented to the user and which responses are possible. The dialogue is permitted to have a hierarchical structure. In addition to a dialogue, a vocabulary must be supplied by the programmer.

Although a speech recognition interface provides application programmers with a flexible and powerful tool, there is a trade-off for the programmer between flexibility and responsibility. A flexible software package has open ends and the programmer has to 'tie up' these loose ends. So, to let the interface work properly, it is the programmer's task to supply the appropriate parameters.

We indicate later how the interface can be integrated by application programmers in different applications.

Portability
It is not feasible to implement an interface without using specific software or hardware features. Nevertheless, the dependence of specific features must be minimal in order to make the interface portable with respect to hardware and software.

To make the interface applicable for already existing software packages, we coded it in the programming language C, not using any specific dialect but true ANSI C. This ensures software compatibility, so that the interface's implementation can easily be ported to some specific C dialect.

At the same time hardware compatibility is required. Hardware compatibility is only possible to some extent; using a particular device requires a specific way of talking to the hardware. So, the use of a specific speech recognizer or a specific telephone handling device imposes restrictions on the ideal of independence. To reduce hardware dependence, however, device-dependent code has been written in a separate module, so it can easily be adapted without having to modify the complete implementation.

Implementation aspects

Hardware and software
The interface establishes a connection between a user and a program which is running on a computer. Often this will be a personal computer using the MS-DOS operating system. Personal computers are cheap, fast and are used in almost every business organization. Above all, in our case, existing voice applications were developed on a PC-platform. Since it is feasible to implement a SR interface on this platform, there are no special needs with regard to the system.
Nowadays various speech recognizers are commercially available. Keeping the application area of the interface in mind, it is obvious that we need a speaker-independent recognizer. The level of desired quality and flexibility of such recognizers is only to be found in discrete recognizers. Consequently, we used a speaker-independent, discrete recognizer.

As with recognizers, various types of telephone handling devices exist. They differ, amongst other aspects, in the quantity of telephone connections that can be handled at the same time. We preferred a THD with a modular structure, which makes it possible to compose any desired configuration. When handling multiple telephone lines simultaneously, it is important to realize a fair application. The notion of fairness we mean here is that no user connected to the IVRS should get stuck in a dialogue, waiting for services occupied by other users. As a matter of fact, we want each user to be unaware of the fact that he or she is sharing the same service with other users. So, the THD must be capable of supporting a version of multitasking. To establish this, the selected THD uses non-blocked I/O-functions (see the section on multitasking for more detail).

We used the programming language C to code the interface functions and we tried to limit the use of non-ANSI constructions. The working environment consisted of conventional programming tools, such as source editors, compilers and debuggers.

In addition to the programming tools, we needed software drivers to talk to the hardware i.e. the recognizer and telephone handler. Both drivers come with the hardware.

Structure
We can describe the implementation of the interface from both a hardware and a software point of view. The hardware structure consists of a telephone line connected to a SRD and a THD in parallel, which in turn are connected to a host computer. A typical hardware structure is illustrated in Figure 1.

From a software point of view, we can distinguish a modular structure. There is a module for each supported function of the interface. Furthermore, device-dependent code and auxiliary functions are placed in separate modules. The set of modules can be linked to an application engine, i.e. an already existing module which controls the various functions of a VRS. This engine ensures progress for all ‘hooked’ applications for all available telephone lines. One could say that the application engine provides the ‘heartbeat’ of the implementation.

Multitasking
In general, a telephone-based IVRS is connected to several telephone lines (typically 16 to 24). To provide all callers with the same service, all connections should be handled simultaneously, that is in a non-blocking way. Non-blocking means that no user may be blocked by another user. Otherwise, caller 20 for example has to wait until caller 19 has entered a credit card number, caller 19 has to wait until caller 18 has decided which sports results he or she would like to hear and so on. So, it is desirable to have multitasking features to realize a fair application. But as the VRS is often installed on a personal computer using the
operating system MS-DOS — or a different operating system capable of serving just one application at a time — a problem arises.

In order to solve this problem, each channel is described by states (see the next section for a discussion on states and state machines). Only when some event occurs, does the application have to decide which state the channel will be in next. Events are actions which occur at an unexpected moment in time, e.g. an event is generated if someone calls the system or input is entered. Thus, if a specific task for a telephone line has been completed, an event is generated to inform the application about the termination of the task. The word task here refers to an activity which requires some time to complete, for instance playing back a voice message or waiting for input. During the time this operation is taking place, the channel is busy, but should not block other channels from being served. Since only one task at a time can be carried out for each channel, other channels can be served concurrently.

In this way, despite using a single-tasking operating system, a — more or less 'pseudo' — multitasking application can be built. Here multitasking refers to the processing of the different channels; of course, the main application is, as far as the operating system is concerned, the only process running. To buffer the events generated by all connections and to provide a first-come-first-served service, all events are placed in an event-queue. Each event is associated with a particular channel. By inspecting the events in the same order as they arrived, one can easily implement an equal priority scheduler. As a matter of fact, one could interpret the event mechanism as a 'smart' polling instrument: only those channels that need to be polled are actually polled. In fact, the application engine realizes this mechanism. This is shown in Figure 3. Whenever an event is presented, the state of the corresponding channel is completed and a new state is determined. After initiating this new state, the event-checking cycle is executed again. The following pseudo-code implements the engine.

```plaintext
Initialization();
while (system_is_running)
{
    HandleKeyboard();
    HandleApplicationDevices();
    if (event present)
        HandleEvent (channel, event);
    CheckSystemStatus();
}
```

To ensure that hardware devices are making progress, the function HandleApplicationDevices() is called in the main loop. It checks the status of, for example, speech recognizers, and generates events, if necessary. In general, the function HandleApplicationDevices() can be described as follows:

```plaintext
HandleApplicationDevices()
{
```
The HandleEvent( ) function takes care of the completion and initialization of states, and serves as a gateway to the connected applications. In the next section we describe the completion and initialization part of this function.

The function CheckSystemStatus( ) determines the status of the overall application and deals with a request for system shutdown, for instance.

So when might a caller be delayed by other callers in the system? Suppose the system serves $n$ telephone lines simultaneously, and handling an event takes an average of $t$ seconds. Say that the average duration of a multitasking function is $d$, then an arbitrary user is delayed if $n \times t \geq d$. These three parameters are very difficult to measure; they differ for each application and are dependent on...
hardware and software. Using the worst case values of $t \approx 0.05 \text{s}$, and $d \approx 5 \text{s}$, one can estimate the number of telephone lines that can be served without delays. Filling in the equation yields $n \geq 100$, which is much greater than the number of connections that can be realized on a PC-platform. This corresponds with the observations of real operational systems, which show that there are no delays.

**State machines**
A state machine is an elegant way to describe the desired behaviour of an application. For each channel, the set of possible states is defined, ranging from an idle state, establishing a call, the dialogue of the call to the termination of a call. When the states are identified, all possible transitions from one state to another state must be tagged; transitions are caused by events (mentioned above). States and transitions together form a state machine that implements the desired interface function.

In fact, each state in a state machine can be divided into two parts: an initialization part and a completion part (see Figure 4). In the former, a multitasking function such as receiving input or playing a prompt is initiated, while in the latter the next state is identified. Chaining all initialization–completion parts together gives the path through the program during an execution of the program. Both completion part and initialization part are executed during the HandleEvent() function mentioned above. They form the kernel of each application. The pseudo-code for both functions is as follows.

```c
CompleteApplicationState (channel, event)
{
  switch (current_state)
  {
    
    case state_i:
      switch (event)
      {
        case event_i:
          perform corresponding actions;
          determine next state;
          
        case event_j:
          perform corresponding actions;
          determine next state;
          
        default:
          perform default actions;
          determine next state;
      }
    
  }
}

InitializeApplicationState (channel)
```
{ 
    switch (state) 
    { 
        . . .
        case statei: 
            start multitasking action; 
        . . .
    } 
}

State machines are clear and universal, but nevertheless become cluttered when used for complex systems. To maintain the benefits of state machines and to preserve clarity — thus reducing the risk of making mistakes — one can design several state machines, according to their functions. In the implementation of the interface, there are state machines for each recognition function class (see next section). Different state machines can be combined to form one complex function, by stacking all sub state machines. Each time another state machine is started, the current context is saved and the new state machine is pushed on a stack structure. The state machine on top of the stack is the state machine which is actually being executed. When a state machine has been completed, it pops itself from the stack and restores the saved context. Because of this context-saving, state machines can be used in a nested way.

Application building

Interface configuration

In principle, the use of speech recognition provides only one new feature: recognition of a spoken word. That could be a reason to implement just one function: the recognition of a single word. But to meet the caller’s way of thinking and to help the caller to say the correct word, the interface is implemented using multiple recognition functions. These functions take care of the communication between caller and the interface. In addition, as already stated earlier, the interface can be linked to various kinds of telephone-based IVRSs. In order to realize this feature, an application programmer can configure the interface by providing several parameters to three constituent functions:

- switching functions,
- recognition functions,
- information retrieval functions.

The switching functions take care of initialization and termination of the interface’s operation. Parameters must be supplied to inform the interface which vocabulary is being used, which dialogues exist, how the interface hardware is configured, which file contains the voice messages etc.

Recognition functions recognize a single word from a list, or menu, of possibilities, a number of digits, and just “yes” or “no”. Parameters indicate on
which channel the functions must be active and which dialogue must be used. In all these functions, the interface sets up an elaborate dialogue to help the user make the right choice. The interface keeps track of the user's efforts and of the unexpected situations (from the user's point of view) which arise. Doing so enables the interface to supply the user with detailed and appropriate help messages. These help messages are selected in accordance with the history of the user–machine communication session. Finally, the interface presents the recognition results to the application, or informs the application that, despite all tries, no recognition results could be deduced.

The last category contains information retrieval functions which enable the application to obtain more detailed information about recognition results or errors that occurred. After initiating a task, the functions return control immediately to the calling application and provide an error code, indicating whether the call was successful or not. If the call fails, the error code gives an indication of what caused the call to fail. A retrieval function can be used to get detailed information on the supplied error code. At the moment the function completes processing, its result is stored internally in the interface. To obtain the recognition result, the application calls a retrieving function.

Vocabulary construction

While using the interface, we found that the quality of the vocabulary plays an important role in the success of the human–machine interaction. The vocabulary affects not only response time and recognition accuracy, but also a user's expressiveness. In order to construct adequate vocabularies, we give the following rules of thumb.

Use subvocabularies

Dividing the vocabulary into several subvocabularies reduces the chance that a word not pronounced clearly confuses the recognizer (see also the section on tokens, templates and vocabularies). Particularly when information about the context of the dialogue is available, subvocabularies can be very useful.

Train thoroughly

In order to achieve speaker-independent recognition, the system must be thoroughly trained. This means that sufficient (typically, several dozen) training samples should be available. Not only is the quantity of the training samples important; but they should be taken from a representative sample of the future users.

Ensure words are distinct

The words in a vocabulary should be (phonetically) sufficiently distinct. As an illustrative example, 'bold' and 'bolt' will definitely cause problems.

Train and use in the same situation

It is important that the system has been trained on the vocabulary in the same circumstances as it will be used. If the vocabulary training is carried out in a
laboratory, but the system is then used in a noisy environment, recognition performance can be expected to decrease considerably.

Choose an appropriate size
To guarantee a natural interaction, the number of words in a vocabulary should not be too small. On the other hand, if the vocabulary has too many entries, its response time may be too slow, or it may become difficult to ensure sufficient distinction between words. What ‘small’ and ‘large’ mean with respect to vocabulary size depends on the specific situation.

Use natural words
The words in the vocabulary should not be exotic, but natural. In that way, words are easy to remember and users are likely to say the correct word instinctively.

Deal with unwanted sounds
During recognition, an unknown utterance is pin-pointed to a known word in the vocabulary. To support the possibility of recognizing words that are not in the vocabulary, a garbage template can be added to the vocabulary. Such a template must be capable of recognizing words it has never been trained with. A garbage template, however, can be trained with specific unwanted sounds (for example: breathing, coughing and laughing) and can be untrained with the words in the vocabulary. If a non-vocabulary word is spoken or true noise is encountered, the speech recognizer will select the garbage template, permitting the interface to act correspondingly. For example, the interface can play a help message on the possible input, or it can ask the user to repeat the input. This technique of coping with unwanted sounds has been proven to be very convenient.

Discussion

Design review
In our effort to establish the synergy of the human factors and technology described, we took an approach that consisted of the following steps:

- performing a literature study;
- building a throw-it-away prototype;
- constructing a more elaborate version of the interface.

The literature gave insight into existing applications and how these could be improved. Combining the uncovered topics has lead to the construction of a prototype interface. To build this prototype, we took an ad hoc approach, which resulted in a system in which the interface and the application were tightly coupled. It had a rather primitive dialogue and the vocabulary was not well constructed. The prototype, however, did support a multitasking feature. The literature study and the construction of the prototype were carried out in approximately six months.
With respect to the prototype, improvements had to be made regarding the dialogue and the mental model that was used (realized by the state machine approach) and the quality of the vocabulary. The revised version was also better structured and it was separated from the test application. Furthermore, documentation was written and the interface was put in a software production library. This third and final phase was accomplished in approximately three months.

The results of the literature study and the prototype construction, together with the revised version, resulted in the ideas and findings which we discussed in this paper. Although we investigated several existing speech recognition systems and experimented with a small group of potential users to justify our approach, we did not perform a large scale experiment to accumulate user data. Performing such an evaluation could produce valuable data for calibrating the interface design and implementation.

The final and revised version of the interface is now being integrated into a so-called ‘putting-through-on-busy’ application. Such a system offers the caller who is trying to contact a person already involved in another telephone call, the opportunity to wait or to leave a message, in the same way that a telephone operator does.

Findings
Experience so far supports the idea that it is crucial to combine human factors and technology in order to realize a natural and adequate form of human-computer interaction. Furthermore, it has been proven that the interface’s implementation is easy to integrate in current telephone-based IVRSs.

Summarizing the results of the design and implementation of the described interface, it seems that by emphasising human factors and integrating the benefits of speech recognition within telephone-based IVRSs, the interface:

- provides a wide I/O channel;
- offers a natural way of communication;
- relieves the users from coding their input;
- provides every telephone-owner access to speech recognition IVRSs;
- facilitates forward reasoning, that is, stimulates problem-solving.

In addition, the described implementation of the interface guarantees that it:

- is easy to use for application programmers;
- can easily be added to existing telephone-based IVRSs;
- is software- and hardware-independent.

We believe that the presented interface has extra value because it synergises speech recognition technology and human factors — without this synergy the interface would lose its value.

Future improvements
The results notwithstanding, some further research is required. In order to
make it possible for humans to communicate in a fully natural way with computers, it is desirable to make use of continuous speaker-independent recognizers. This requires further development of both speaker-independent continuous speech recognition technology and the way in which it can be integrated into speech recognition systems, involving research on a possibly phoneme-based approach of disambiguating utterances and assigning an appropriate meaning to them.

While designing the interface, we drew on knowledge about the ergonomics of human–machine interfaces. We suggest that application designers should take note of human factors, as well. In this way, an optimal combination of application and user interface could be constructed.

Furthermore, we found that a well-designed vocabulary on which the system is thoroughly trained is essential for the performance of the interface. It is important that vocabulary developers take care of this aspect.

As a final issue, more attention should be given to construct valid and adequate user models in order to identify users' expectations of speech recognition IVRSs and to predict and anticipate their behaviour. In particular, it is difficult to model behaviour when a user makes 'mistakes' (that is, behaves in an unexpected way). Future research will be needed to deal with this.

Conclusions

In this paper we have discussed the relevant human factors and technology aspects for a telephone-based speech recognition interactive voice response system. We stressed that establishing a synergy between human factors and speech recognition technology is crucial to ensure a natural and acceptable way of human–computer interaction. We also outlined how the synergy has been implemented and how the resulting interface can be integrated into various applications. We found that the quality of the vocabulary plays an important part in determining the value of a speech recognition system, and we provided some rules of thumb for constructing a good vocabulary. In addition, among the suggested topics for further research, we mentioned that adequate mental models should be constructed to deal with unexpected user behaviour.

We believe that realizing a synergy of human factors and speech recognition establishes a user-orientation that is vital for a speech recognition interface that will enable a natural form of human–machine interaction.

Acknowledgements

The design and implementation of the speech recognition interface has been realized in the McSPRINT project of McCOMM International b.v., (Holland). This project was supervised by Prof. W.A. Halang of the Computer Science Department of the University Groningen.

The referees provided valuable comments to enhance the quality of this paper and Henk Harkema suggested several improvements with regard to the English.
References


Clements, M.A. (1987) 'Voice recognition systems can be designed to serve a variety of purposes' Ind. Eng. 19, 44–57


Thimbleby, H. (1990) User Interface Design Addison-Wesley


Interacting with Computers vol 6 no 1 (1994)