Alternative methods for controlling the user interface in a browser for technical documentation

Cecilia Svensson

2003-02-26
Alternative methods for controlling the user interface in a browser for technical information

Examensarbete utfört i Medieteknik vid Linköpings Tekniska Högskola, Campus Norrköping

Cecilia Svensson

Handledare: Jan-Olov Benitez
Examinator: Mark Ollila

Norrköping den 26 februari 2003
<table>
<thead>
<tr>
<th>Language</th>
<th>Report category</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Svenska/Swedish</td>
<td>Licentiatavhandling</td>
<td>ISRN ITN-MT-EX--03/005--SE</td>
</tr>
<tr>
<td>Engelska/English</td>
<td>Examensarbete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-uppsats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D-uppsats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Övrig rapport</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Alternative methods for controlling the user interface in a browser for technical information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Författare</td>
<td>Cecilia Svensson</td>
</tr>
</tbody>
</table>

Sammanfattning
Abstract

When searching for better and more practical interfaces between users and their computers, additional or alternative modes of communication between the two parties would be of great use. This thesis handles the possibilities of using eye and head movements as well as voice input as these alternative modes of communication.

One part of this project is devoted to find possible interaction techniques when navigating in a computer interface with movements of the eye or the head. The result of this part is four different controls of an interface, adapted to suit this kind of navigation, combined together in a demo application.

Another part of the project is devoted to the development of an application, with voice control as primary input method. The application developed is a simplified version of the application ActiViewer, developed by AerotechTelub Information & Media AB.

Nyckelord
Keyword

Eye tracking, head tracking, user controls, speech recognition, speech synthesis, grammar, command & control.
Abstract

When searching for better and more practical interfaces between users and their computers, additional or alternative modes of communication between the two parties would be of great use. This thesis discusses the possibilities of using eye and head movements as well as voice input as these alternative modes of communication.

One part of this project is devoted to find possible interaction techniques when navigating in a computer interface with movements of the eye or the head. The result of this part is four different controls of an interface, adapted to suit this kind of navigation, combined together in a demo application.

Another part of the project is devoted to the development of an application, with voice control as primary input method. The application developed is a simplified version of the application ActiViewer™, developed by AerotechTelub Information & Media AB.
Acknowledgements

The author wishes to thank the staff at Information & Media AB for moral support, help and encouragement, and in particular my supervisor Jan-Olov Benitez for good cooperation and many ideas in all situations and my co-supervisor Gunnar Carlson for tips and proof-reading. Thank you Svante Ericsson, Ulf Jansson and Fredrik Göransson for the initiatives taken that made this project possible.

A word of gratitude also goes to my examiner Mark Ollila at Linköping Institute of Technology for his trust in this project.

Cecilia Svensson
Växjö, January 20, 2003
10.2.2 Memory ..............................................................................................21
10.2.3 Sound card .........................................................................................22
10.2.4 Microphone........................................................................................22
10.3 Software requirements ............................................................................22

11 Limitations of Speech Technology ............................................................23

12 Microsoft Speech API ..............................................................................25

13 Application Design Considerations ..........................................................27
  13.1 Combination of devices...........................................................................27
  13.2 Using speech interfaces in an effective way.........................................28
  13.3 Adding speech to applications..............................................................29
    13.3.1 Speech as an Add-on ........................................................................29
    13.3.2 Designed for speech.........................................................................30
    13.3.3 Speech Required ..............................................................................30

14 Grammars ..................................................................................................31
  14.1 Functions of Grammars...........................................................................31
  14.2 Extensible Markup Language ................................................................32

15 The application ...........................................................................................35
  15.1 Navigation List .......................................................................................36
    15.1.1 Read Items ....................................................................................37
    15.1.2 Open ............................................................................................37
    15.1.3 Read and display ............................................................................38
  15.2 Tab Control .............................................................................................39
  15.3 Text and Image Area .............................................................................40
  15.4 Resize button .........................................................................................41

16 Process of work ..........................................................................................42

17 Continuation of the project ......................................................................49

18 Voice navigation: Discussion and conclusion ..........................................51

19 Conclusion of master thesis .....................................................................52

20 Bibliography .............................................................................................54
1 Introduction

1.1 Purpose

This Master’s thesis is the author’s final thesis at the Master of Science Programme in Media Technology and Engineering, Linköping University, Sweden. The project is developed for AerotechTelub Information & Media AB, where all work has been carried out. The task was to find and implement alternative navigation methods for a computer interface, with focus on navigation with head or eye movements and voice navigation.

1.2 Objectives

1.2.1 Objectives of navigation with head or eye movements

When the period devoted to the head and eye navigation of the project is finished, an interface and interaction techniques for navigation with the head and the eyes is designed and implemented in a demo application. Depending on how time consuming the development is the technique is also implemented in ActiViewer™, an application developed by Information & Media, see the next section. The interface contains implemented ActiveX controls, which substitutes the ordinary controls; buttons, checkboxes etc. These developed controls are combined in the demo application, with the purpose of testing the navigation in the interface and to show how it works.

At the end of the period there is a decision whether the developed solution is an appropriate solution. The decision will be built on testing and evaluation of the equipment from SmartEye.

1.2.2 Objectives of voice navigation

When the period devoted to the voice navigation is finished, there is an alternative version of ActiViewer™, in which the user is able to navigate with voice commands. ActiViewer™ is an xml-browser developed by Information & Media AB, used to display large amounts of structured information.
Chapter 1. Introduction

Figure 1.1 The original application ActiViewer 1.0

The developed application is capable of reading texts for the user, and the user can control this reading by voice commands, such as stop, continue, repeat etc. The navigation in the application is handled with voice commands or traditional input devices, such as the mouse and the keyboard. A well-formed grammar is created, with all the commands specified as an xml file.

The main part of the application interface have been implemented by my supervisor Jan Olov Benitez, and the voice control interface have been implemented by the author.

A primary objective is to make the application as general as possible. The contents shall be easy to replace, without having to do any code changes in the application.

The intention is to control the application with as natural speech as possible. Helpful error messages are given when the user gives an incorrect command, and the commands to the application are constructed to be as natural and intuitive as possible.

Depending on how time consuming the development is, a search function is to be implemented. This will facilitate for the user who wants to go to a special part in the application, without having to navigate through the whole content.
1.3 Thesis outline

The thesis consists of two major parts, one part which considers the head and eye navigation, one part for the voice navigation. The first part comprises chapter 2 to chapter 7, and the second part chapter 8 to chapter 18. At the end of the thesis the conclusions drawn of the complete project is handled.
Chapter 2. Navigating with head or eye movements: Introduction

2 Navigating with head or eye movements: Introduction

When searching for better and more practical interfaces between users and their computers, an additional or alternative mode of communication between the two parties would be of great use. To use the movements of the user’s head or eyes could serve as a good source of additional input. While technology for measuring a user’s visual line of sight and head position and reporting it in real time has been improving, what is needed is good interaction techniques that adds eye and head movements into the user-computer dialogue in a convenient and natural way.

This part of the thesis handles the first part of the project, to find interaction techniques and an interface to use when navigating with head movements or eye movements in an application interface.

To design good interaction techniques and an intuitive interface, the qualities and drawbacks of the input methods using head and eye movements must be considered. The interface design issues of this kind of input device are discussed in chapter 3. The resulting interface solutions and interaction techniques are handled in chapter 4.

To realise this alternative way of navigating in an interface, some special equipment had to be used. In chapter 5 further details about this equipment is given, which is used for head and eye navigation. The work process is presented in chronological order in chapter 6. In the last chapter of this part of the thesis, chapter 7, conclusions from this part of the project are drawn and problems discussed.
3 Interface Design Considerations

3.1 Eye tracking

When designing an interface for eye movement navigation, the simplest solution would be to substitute an eye tracker directly for a mouse. This means that a screen cursor follows the visual line of sight, that is, where the user is looking on the screen. But compared to mouse input, eye input has some advantages and disadvantages, which must be considered when designing eye movement-based interaction techniques.

First, eye movement input is faster than other current input media. Before the user uses the screen cursor, he or she usually looks at the destination to which he wants to move. With the screen cursor following the eye movements of the user, he or she can perform action before any other input device is used.

Second, the eye navigation is easy to use. No training or particular coordination is required for the users to make their eyes look at an object. The eye movement navigation method is a natural way of using an interface.

Despite these qualities of navigating with eye movements, there are quite some drawbacks with using the eye as a computer input device. The first is the eye itself, the jerky way it moves and the fact that it is almost never still. During a fixation with the eye at an object, the user thinks he or she is looking steadily at the object, and is not aware of the small, jittery motions performed by the eye. This will cause the screen cursor to rove without the user understanding why it acts this way.

Moving the eyes is often done subconsciously. Unlike a mouse it is relatively difficult to control eye position precisely all the time. The eyes continually dart from spot to spot without one’s knowing of it. Having the screen cursor following the users line of sight, which is moving constantly and very quickly from spot to spot, continually scanning the screen, would be very disturbing for the user. The user already knows where he or she is looking, thus it can be very annoying to have the screen cursor to tell the same thing.
Further, if there is any calibration error, the cursor will be slightly offset from where the user is actually looking, causing the user’s eye to be drawn to the cursor, which will displace the cursor further.

In comparison to a mouse, eye tracking lacks the functions that the mouse buttons have. Using blinks or eye closings is not appropriate as this is a subconscious act. Another solution is required, and is explained in chapter 4, Interface Solutions.

3.2 Head tracking

The advantages of using eye tracking are not good enough compared to the drawbacks.

Another solution that has been considered is the use of head movements instead of eye movements. The user’s head movements are more controlled and also easier to track and follow. The screen cursor will follow the movements more naturally, and will not irritate the user. This way of navigating has the benefits of mouse use and the benefits of the eye tracking.

These qualities of head tracking together with the fact that the SmartEye software for head tracking is much more precise than for eye tracking, made us decide to abandon the eye tracking and only consider head tracking in the development of the interface.
When navigating in a computer interface with head movements, solutions to perform actions corresponding to mouse clicks and keyboard presses are needed. The navigation interface must provide all the functions that the user normally has access to, for example click, double-click and right-click. One solution is to develop alternative controls, adapted to the new navigation method, and replace the ordinary controls, such as buttons, checkboxes etc in the interface. It is also important that the controls together make up a good and intuitive interface and that the user understands how to navigate in it.

To extend the application ActiViewer™ with head tracking as an alternative input device the application interface had to be studied. Four different controls are the most interesting for the interface of ActiViewer™, buttons, checkboxes, a special made list box and a button that resizes the four different sub windows in the application. To make an interface suited for head navigation alternative versions of the controls mentioned was decided to be developed, and combine them in an interface. The resize control is created as a discrete application because of implementation complexity. The other three alternative controls are created as ActiveX components for Visual Basic, and they can be used when implementing applications.

4.1 Interaction techniques

An interaction technique is a way of using a physical input device to perform a task in a human computer dialogue. The techniques implemented in this project are application specific, adapted to the specific controls developed. Below, a description is made of the head movement-based controls and interaction techniques implemented and the motivations for them.

As mentioned before, the interface studied contains buttons, checkboxes and list boxes. To click a button or a checkbox, or to open up a scroll-list two possible solutions were considered – using dwell time and using pop-up menus.

With the dwell time approach a click is performed when the screen cursor passes over the desired object to be clicked and stays there during the dwell time. With pop-up menus the menu shows when the cursor passes over the desired object, but nothing is performed until an item in the menu is selected. The pop-up menu approach was found to be more convenient as the user has an option to click or not.
The dwell time approach could be perceived as a stressful feature, as it will perform a click whenever the user passes the cursor over one of the controls for a short dwell time. Using a long dwell time, to ensure that simply navigating around on the screen does not perform a click, could solve this problem, but it attenuates the speed advantage using head movement as input and also reduces the responsiveness of the interface.

The pop-up menu displayed below the control, contains one or several icons showing the type of action that can be performed, in this case it is only an ordinary click with the left mouse button. When the mouse moves over this icon the click is performed. If the user chooses not to click, he can just continue to navigate in the interface and the pop-up menu disappears.

![Figure 4.1 Pop-up menu of the button](image)

Though, the dwell time approach was not discarded completely, it is used to open the list box and to select items in it. To open up the list box and select items the hover function of the mouse is used. This means that the list opens when the cursor passes over the list box heading. The body of the list appears on the screen and the user can look at the items shown in the list. The same technique is used to select an item in the list, and it can be seen as a form of dwell time as the mouse hover function takes about 50 ms to make the selection.

There is a thought behind the solution of combining the pop-up menu approach and the dwell-time approach. If the result of selecting the wrong item can be undone in a simple manner, that is, if another selection could be done on top of the first, without causing an adverse effect, the dwell time approach can be used instead of the pop-up menu approach. In the list box, this is the case; an item is deselected easily by selecting another item in the list. A button click is more difficult to undo, so in that case the pop-up menu approach is used.

The list box is minimized when something else in the application retrieves focus, even if no item in the list has been selected. A solution where the list disappears when the screen cursor moves out from the list box area was considered, but was discarded rather quickly as it caused trouble in combination with the scroll function of the list.

The fourth control, the resize button, is implemented as a specially made drag and drop function with a dwell time to pick the button up and to put it down. When the user places the screen cursor over the button for a given dwell time (400 ms.), the default cursor transforms to the four-headed sizing cursor and the button follows
the cursor until it is held still for 400 ms. Then the default cursor shows and the button is released.
All of the controls in the interface also work together with the keyboard and the mouse, if the user desires to use these input devices instead of, or in combination with, the head movements.
5 Equipment for head tracking

To track the movements of the head, we used a product from the company Smart Eye\(^1\), called Smart Eye Mouse Ground Pro. Together with a web camera, placed in front of the user, the application makes the mouse pointer follow the head movements of the user. When the application is started, the user gets an image on the screen from the web camera. A mask appears, and the user places the mask at the position of the eyes and mouth, see figure 5.1. The user can adjust how much the cursor will respond to the head movements in the horizontal and vertical direction. There is also a possibility to set the camera properties to get the optimal conditions for the tracking.

![SmartEye Mouse Ground Pro](image)

**Figure 5.1** SmartEye Mouse Ground Pro

Smart Eye are careful about giving information about the techniques of their solution, so the process of the Smart Eye equipment is only described briefly below.

---

\(^1\) www.smarteye.se
A digital camera is used to collect the input information. If an analogue camera is used the signal is digitalized (1). Then real time image processing is performed on a hardware platform, for example a PC. The required processes and algorithms are implemented in the Smart Eye software platform (2). The result is extracted from the system. It could be the head position or the gaze vector of the person in front of the camera (3).

**Figure 5.2 The SmartEye process**
6 Process of work

The first step in the developing process was to work out an appropriate model for an interface, where the user navigates with head movements. As mentioned earlier in the thesis, the final demo application is a set of controls, where pop-up menus and dwell time are used in combination to perform action in the interface.

Four different icons was designed, for the pop-up menus, in Photoshop. The icons describe the four different mouse actions; single click, right click, double clicks and drag-and-drop. But as the interface of ActiViewer contains controls only using an ordinary single click, the only icon used in the demo application is the single click icon, see Figure 4.1.

The first two controls implemented were the alternative button and checkbox. They were created as ordinary controls but with different behaviour to suit the actual interface. When the alternative list box with its scroll function was developed, the three different solutions were merged into one application. This solution generated a lot of code, with no structure at all, which made us reconsider and the new decision was to implement three different ActiveX components instead. The benefit of this solution is not only the code issue, but it also makes it possible for other developers to use the controls when creating similar applications, the controls will be reusable.

The resize button was implemented as an own application, partly because of the complexity of the code, partly because it is not one of the basic features in ActiViewer, and therefore it is not necessary to have it in the demonstration application together with the other controls.

When the ActiveX controls were implemented a demonstration application was developed, with the controls in combination with each other. When the controls were functioning together, it was detected that some smaller features in the controls had to be modified, to make them work satisfactory together in the interface. So, a step back was taken, the controls were modified, and the demo application development then continued. The development continued like this until the design and function of the implemented interface was satisfactory.

The next step was to deploy the application. An installer was created that installs the application on another computer and sets up shortcuts and file associations. The installer also checks if the necessary .NET Framework is installed on the computer, and if it is not, the installer is supposed to install the framework. Unfortunately, the
installation of the framework is not working correctly yet. Time will be devoted to solve this problem when other, more important tasks are solved, and if it cannot be solved, it is fairly easy to download the framework before installing the demo application.

The equipment for the head and eye tracking from SmartEye, SmartEyeLite was not delivered until after the implementation of the demo application. Because of this the interface could not be tested before it was finished, and the decision whether the solution was appropriate or not had to be taken after the development.

To navigate in the developed interface, a function where the screen cursor follows the tracked head movements was needed. This function was not included in the SmartEyeLite software, so an application where an image follows the head movements, was developed. The image was later supposed to be replaced by the screen cursor. The result of the implemented application was not good enough, the image did not follow the movements of the head as correctly as wanted and the image was roving quite a lot when the head was held still.

SmartEye was contacted about this, and advice of how the lighting in the room could disturb the tracking of the head movements was given. The presence of strip light behind the user makes the image captured by the camera to flicker, which results in a roving picture. They also informed that SmartEye has developed an application like the one implemented at the moment, where the screen cursor follows the head movements. They recommended the use of their application Mouse Ground Pro instead of implementing an application like this, as it probably would not be as exact and well functioning as theirs.

After installation of Smart Eye Mouse Ground Pro, the evaluation of the head tracking and the interface of the demonstration application was started. See the next chapter for the results of the evaluation.
7 Navigating with head or eye movements: Discussion and conclusion

When moving the screen cursor around the screen with the application Mouse Ground Pro, the cursor follows very well. But in combination with the interface of the demo application, it gets more complicated. The details in the interface seem far to small, and proper navigation demands a lot of patience of the user. The problem here lies not in the interface, as the icons and controls are of standard size. To implement them bigger would not be appropriate; the interface would seem clumsy and unpractical. What is needed is a head tracker that is much more sensitive for small movements, which will make it possible to navigate properly in detailed areas. Unfortunately, this feature will raise the problem of moving the cursor longer distances; the user will have to make large head movements to, for example, move the cursor from one side of the screen to another.

Another drawback of the Mouse Ground Pro application is that the screen cursor still roves a bit, which makes the resize button application impossible to use. This is because the timers implemented in the button require the mouse pointer to be exactly still to trigger an action.

Another feature that is needed is some kind of “renewal of the grip taken”. When using a mouse to navigate we often take a new grip, we do not move it in one single movement. Using the head to move the screen cursor feels very stiff, as we cannot take this new grip. The navigation is uncomfortable and the user will respond negatively to this kind of navigation.

These considerations made us reconsider and the decision was made not to continue developing the head navigation in ActiViewer™. At a later time in the project we will consider this decision again, and maybe we will continue on the head tracking solution.

However, for a user that is not able to use the mouse or keyboard, because of a handicap of the user or that the user has “full hands”, the eye tracking method could be a good alternative.
The result of this part of the project is two demo applications and a decision. The demo applications contain implemented ActiveX controls functioning in combination. These applications are used together with the program Smart Eye Mouse Ground Pro, to make navigation with the head possible. The demo applications can easily be installed on remote computers with the created installers if the .NET Framework is installed. The decision that was taken, as mentioned earlier, was to not continue the development of head tracking in ActiViewer™.
8 Voice navigation: Introduction

Speech recognition applications are conversations. But instead of conversations between people, they are conversations between people and machines. A speech application will prompt the user to speak with the computer to get a task done, such as booking a flight or, like in this project, navigating in a large amount of information. As the user speaks, speech recognition software searches against a predefined set of words and phrases for the best match to perform the task.

In this part of the thesis the techniques behind speech recognition and speech synthesis will first be described in chapter 9. The system requirements and limitations to use the technology are handled in chapter 10 and 11. In chapter 12 the Microsoft Speech API is presented, and what should be considered when designing a speech-enabled application is declared in chapter 13. How a grammar works will be explained in chapter 14, how the final application functions in chapter 15, and the process of work will be described in chapter 16. At the end of this part discussion and drawn conclusions of this part of the project are made.
In the mid 1990s, personal computers started to become powerful enough to make them understand speech and to speak back to the user. In 2002, the technology has become more affordable and accessible, for both business and home users, but is still a long way from delivering natural conversations with computers that sound like humans.

Speech technology delivers some useful features in real applications today, for example many companies have started adding speech recognition in their services, like flight booking systems or stock selling systems. Home users can use the speech technology in mainstream applications, like dictating a Microsoft Word document or a PowerPoint presentation. It is also possible to use commands and control menus by speaking. For many users, dictation is far quicker and easier than using a keyboard. Certain applications speaks back to the user, for example Microsoft Excel from Office XP, that reads back text as the user enters it into cells.

The two underlying technologies behind these possibilities are speech recognition (SR) and text-to-speech synthesis (TTS).

### 9.1 Speech Recognition

Speech recognition, or speech-to-text, involves capturing and digitising the sound waves, converting them into basic language units called phonemes. Words are constructed from these phonemes, which are contextually analysed to ensure correct spelling for words that sound alike (such as write and right). The figure below illustrates and explains the process of speech recognition.
Speech recognition engines, also referred to as recognisers, are the software drivers that convert the acoustic signal to a digital signal and deliver recognized speech as text to the application. Most speech recognisers support continuous speech, meaning that the user can speak naturally into a microphone at the speed of a normal conversation. Discrete speech recognisers require the user to pause after each word, and are currently being replaced by continuous speech engines.

Continuous speech recognition engines support two sorts of speech recognition; dictation, in which the user enters data by reading directly to the computer, and command & control, in which the user initiates actions by speaking commands.

Dictation mode allows the user to dictate memos, letters, and e-mail messages, as well as to enter data using a speech recognition dictation engine. The possibilities for what can be recognized by the engine are limited by the recogniser’s "grammar", a dictionary of words to recognize. Most recognisers that support dictation mode are speaker-dependent, meaning that the accuracy varies on the basis of the user's speaking patterns and accent. To ensure correct recognition, the application must learn the user’s way of speaking. This is done by creating a "speaker profile", which includes a detailed map of the user's speech patterns. This profile is used in the matching process during recognition.

Command and control mode is the easiest mode to use when implementing speech in an existing application. In command and control the grammar can be limited to the list of available commands, which is much more finite scope than that of continuous dictation grammar. This provides better accuracy and performance as it reduces the processing required by the application. The limited grammar also eliminates the need for the recogniser to learn the users way of speaking. To read more about grammars, see chapter 14, Grammars.

Speech recognition technology enables several possible features to be included in applications and they often result in good qualities of the application. One
Chapter 9. Speech Technology

possibility is to use hands-free computing which suits well in environments where a keyboard is impractical or impossible to use. The possibility to add speech to an application makes the computer more “human” and may make educational and entertainment applications seem more friendly and realistic. Another feature is the quality of having easier access to application controls and large lists, when the user can speak any item from a list or any command from a large set of commands without having to navigate through whole lists or cascading menus. To use voice responses to message boxes and wizards makes the use of the application more efficient and comfortable.

For the interested reader the author refers to [5], which gives a good example of the use of speech recognition in an educational and entertainment application.

9.2 Speech Synthesis

Speech synthesis, also called text-to-speech, is the process of converting text into spoken language. This process involves breaking down the words into phonemes and generating the digital audio for playback. The figure below illustrates and explains the process of speech synthesis.

![Figure 9.2 Text to speech](image)

Software drivers called synthesizers, or text-to-speech voices, generates sounds similar to those created by human voices and applies several filters to simulate throat length, mouth cavity, lip shape, and tongue position. The voices produced by synthesis technology are easy to understand, but tend to sound less human than a voice reproduced by a digital recording.

However, text-to-speech applications may be the better alternative in situations where a digital audio recording is inadequate or impractical. In general, text-to-speech is most useful for short phrases or for situations when pre-recording is not practical. Below follows a few examples of the practical use of text-to-speech.
• To read dynamic text. TTS is useful for phrases that vary too much to record and store all possible alternatives. For example, speaking the time is a good use for text-to-speech, because the effort and storage involved in pre-recording all possible times is not manageable.

• To proofread. Audible proofreading of text and numbers helps the user catch typing errors missed by visual proofreading.

• To conserve storage space. Text-to-speech is useful for phrases that would occupy too much storage space if they were pre-recorded in digital-audio format.

• To notify the user of events. Text-to-speech works well for informational messages. For example, to inform a user that a print job is complete, an application could say “Printing complete” rather than displaying a message box and requiring the user to click OK. This feature should only be used for non-critical messages, in case the user turns the computer’s sound off.

• To provide audible feedback. TTS can provide audible feedback when visual feedback is inadequate or impossible.

An interesting study has been performed about the user’s ability to understand and remember information given to him or her by computer-generated speech. The topic “Will users be influenced by the “gender” of the speech synthesis” is also studied. See further in [6] and [7].
10 System Requirements for speech applications

To run a speech application, certain hardware and software is required on the user’s computer. As not all computers have the memory, speed, microphone or speakers required to support speech, it is a good idea to design the application so that speech is optional. The following hardware and software requirements should be considered when designing an application containing speech.

10.1 Operating systems

SAPI 5.1 supports the following operating systems:

- Windows XP Professional or Home editions
- Windows.NET Server editions
- Microsoft Windows 2000
- Microsoft Windows Millennium edition
- Microsoft Windows 98
- Microsoft Windows NT Workstation or Server 4.0
- Windows 95 or earlier is not supported

10.2 Hardware requirements

10.2.1 Processor Speed

The speech recognition and text-to-speak engines typically require a Pentium II/Pentium II-equivalent or later processor at 233MHz.

10.2.2 Memory

Speech recognition for command and control requires at minimum 16 MB of RAM in addition to what the running application is requiring, but 32 MB is recommended. Speech recognition for dictation requires at minimum 25.5 MB, 128 MB is
Chapter 10. System Requirements for speech applications

recommended. Text-to-speech uses about 14.5 MB of additional RAM at minimum, but 32 MB is recommended.

10.2.3 Sound card

SAPI 5 does not support all sound cards or sound devices, even if the operating system supports them otherwise.

10.2.4 Microphone

A microphone to receive the sound is required for speech recognition. In general, the microphone should be a high quality device with noise filters built in. The speech recognition rate is directly related to the quality of the input. The recognition rate will be significantly lower or perhaps even unacceptable with a poor microphone.

10.3 Software requirements

The primary software needed to use speech technology is the Microsoft Speech SDK. With this package follows a speech recognition engine and a text-to-speech engine. To develop the application a developing tool is needed, for example Microsoft Visual Studio. Microsoft Internet Explorer version 5.0 or later also has to be installed.
11 Limitations of Speech Technology

Currently, even the most complicated speech recognition engine has limitations that affect what it can recognize and how accurate the recognition will be. The following chapter describes many of the limitations found today.

As mentioned in chapter 10, System Requirements, the speech recognition rate is dependent on the quality of the microphone. A microphone with high quality is required and not every user has a microphone with quality high enough. Also the speech recognition requires a good soundcard, which is supported by SAPI 5.

In general, the user should position the microphone as close to the mouth as possible to reduce the noise coming from the user’s environment. Users in a quiet environment can position the microphone several feet away, but users in noisier environments will need a headset that positions the microphone a few centimetres from the mouth.

Another limitation is sounds generated by the user’s computer. There are some ways to make sure that the microphone doesn’t pick up the speakers. One of them is to wear a close-talk headset, which places the microphone so close to the user’s mouth that it will not pick up the sounds coming from the speakers. Another solution is to use headphones instead of speakers.

As the speech recognition engine likes to hear, it will try to recognize every word it hears. This means that when the user is having for example a phone conversation in the room while speech recognition is listening, the recogniser will hear random words. Sometimes the recogniser even “hears” a sound, like a slamming door, as words. A solution to this problem could be to allow the user to turn speech recognition on and off quickly and easily. The user should be able to do this with all of the input devices, including speech.

To make an application able to recognize words, it has to have a list of commands to listen for, a grammar. This list must contain commands that are intuitive to users and any two commands should sound as different as possible, to avoid recognition problems for the engine. As a rule of thumb, the more phonemes that are different between two commands, the more different they sound to the computer.
A way to make it easy for the user to use the available commands is to display them in the application if possible. Another way is to use word spotting, which means that the speech recogniser listens for keywords. An example is the keyword “mail”, which allows the user to both say “send mail” or “mail letter”, and the recogniser will understand both formulations. Of course, the user might say “I don’t want to send any mail” and the computer will still end up sending mail.

Speakers with accents or those speaking in non-standard dialects can expect more mis-recognitions until they train the engine to recognize their speech. Even then, the engine accuracy will not be as high as it would be for someone with the expected accent or dialect. A speech recognition engine can be designed to recognize different accents or dialects, but this requires almost as much effort as porting the engine to a new language.
12 Microsoft Speech API

The Microsoft Speech API, SAPI, is a software layer used by speech-enabled applications to communicate with Speech Recognition engines and Text-to-Speech engines (SR and TTS). SAPI includes an API - Application Programming interface and a DDI - Device Driver Interface. Applications communicate with SAPI using the API layer and speech engines communicate with SAPI using the DDI layer.

![Figure 12.1 Function of Sapi](image)

The SAPI notifies the application when a speech event occurs. A speech event might be for instance the start of a phrase, the end of a phrase or a speech recognition. Microsoft SAPI 5.1 can handle more than 30 kinds of such speech events. When an event occurs, the application will be notified and receives a structure with information about the event.

The application needs a recogniser object to access the SAPI. There are two ways to set up this object:

- **Shared resource instance.** This set-up allows resources such as recognition engines, microphones and output devices to be used by several applications on the same time.

- **Non-shared resource instance.** This set up allows only one application to control the resources.

The shared resource instance is the preferred option for most desktop applications. With this option chosen, several applications can use for example the microphone.

Initially the speech recognition uses a default voice profile, which performs good results for any voice. It is possible to configure the speech recognition system to a specific voice, which should increase the performance of the recognition even more. To get better results with the default voice profile, there is a possibility to train the recognition engine, by reading texts and teach the engine how the user speaks.
Besides the functions performed by SAPI mentioned above, SAPI controls a number of aspects of the speech system:

- Controlling the audio input, whether from a microphone, files etc., and converting audio data to a valid engine format.

- Loading grammar files, whether dynamically created or created from memory, URL or file, and resolving grammar imports and grammar editing.

- Compiling standard SAPI XML grammar format, and conversion of custom grammar formats, and parsing semantic tags in results.

- Ensuring that applications do not cause errors by preventing applications from calling the engine with invalid parameters.
13 Application Design Considerations

With the advent of introducing speech as an interface device, much consideration must be taken during feature design and software development stages. Each input device has its strengths and weaknesses and those characteristics need to be optimised. As an example, take the keyboard and mouse. The keyboard existed before the mouse and it is still capable of performing many of the same functions as the mouse. The mouse is more precise in certain tasks such as pointing, selecting or dragging, but it is inefficient for textual input. Lately the two devices have evolved together and the unique characteristics from each device are used.

As a new technology, speech recognition and speech synthesis has to find its role in the user interface. For some uses speech is very well suited, for others not at all. Word processors and e-mail applications can take advantage of both dictation and text to speech capabilities. Games may be better suited to use speech recognition for command and control features. Web browsers on the other hand, require additional design considerations if speech is to be used as a device. For example, web pages have fields where the user can enter information. These fields are often arranged in a visually pleasing layout, but not in a particularly systematic layout. Pages usually have a URL line, but they often also have search boxes, comment areas, forms, check boxes and links. To decide how the user assigns speech to a specific box or area can be awkward. Likewise, the ability to read information from a web page can be as awkward for the same reasons.

13.1 Combination of devices

It is possible to combine input devices, for example speech input and the mouse. As an example, a page layout or 3D-modelling application is dependent on the mouse to create for example a box and place it correctly and accurately within the design. The developers may decide to add a speech feature to access for example a dialogue box used to enter the dimensions of the box. Using command and control and the dialogue box as an example the user speaks the command “dimensions box”, the numeric dimensions of the box and then confirm the size chosen by saying “okay”. In this way the speech complements the mouse and the user performs the task placing and sizing a box without having to interrupt mouse positioning. The entire operation is completed quicker with speech and it is more comfortable as the user
does not have to move the mouse from the placing area. This combination, using command and control as well as the mouse does not require a different user interface. The user is simply accessing the application’s already existing menu items, and uses speech as a shortcut to them. The user may of course still perform the task manually. When combining different input methods, the user can concentrate more on the task to be performed because less time is spent on the mechanics of making the change itself.

13.2 Using speech interfaces in an effective way

It is very important to notice that some tasks are easier and better performed with speech, but others are not. To replace the entire interface with a voice system often fails, as the application becomes too complex and not intuitive enough. There are several things to consider when adding speech to an application or to build a speech application interface. The following text describes some of the things to consider making an application with speech convenient to use.

- Pick the appropriate level of speech for the application

For desktop applications, the keyboard is still a natural part of the computer system. To ask a user to enter information from the keyboard is not a new concept and is easily performed. Therefore it may be a good solution to keep this input method for entering text and use speech for other tasks such as command and control or navigating. In the future, it may be better to reverse the roles and use speech as the primary input method, when it is a more accepted and developed technique.

- Speech often works best in combination with other user interface methods

As mentioned before, speech in combination with other input methods is a good and convenient solution. The speech input is not supposed to compete against the other existing input devices. In action games, for example, a quick response is often required. Moving a hand from the joystick to the keyboard to give this response is often disadvantageous. When appropriate, it is better to use speech for these kinds of responses or confirmations.

- Do not add speech if not appropriate

Making a task more complex just to have speech in the application or using speech in cases where it just does not make sense, makes the user confused and the application uncomfortable to use. If the speech does not help the user or makes the application better or quicker to use, it is better to not use speech in the application.

- Use speech to simplify not to complicate

Currently, applications must break down tasks into separate steps. Entering this information is generally limited to one piece of information for each entry. We
consider a Web site, where the user can order airplane tickets. The Web site has separate boxes for each of the departure and arrival cities, date, time, airline and so on.

A natural speech approach allows the user to speak a sentence and the application to interpret the information. In the ticket-ordering example, the user could say “I would like to book a flight from Stockholm to Malmö at five p.m. on the fourth of December and come back in the morning of the fifteenth”. In this case, one sentence covers all the information.

- Consider the user’s environment

For speech recognition to work in an accurate way, the environment must be suitable. A relatively quiet environment, such as a business office, is optimal. SAPI 5.0 recognizes background noise and filters it out. Even occasional loud noises will not change the accuracy, but frequent noises will slow down the processing rate. Therefore a perfectly quiet environment gives only marginally better recognition results than a normal office environment. One problem with the use of speech in an office environment is the issue of privacy. As the user is speaking aloud, he or she could disturb others nearby or the information spoken may be confidential.

- Speech as the most effective option

Users without visual ability will not see the screen and users without manual ability will not be able to use the keyboard and the mouse. For these users speech may be the best, if not the only, input method to operate a computer. The reasons for the disabilities could be for example physical or environmental.

13.3 Adding speech to applications

When all design issues are considered speech can be added to the application. Speech designs can be categorised into three major groups.

13.3.1 Speech as an Add-on

When there already is an existing application, this is the solution that requires the least amount of work. No code changes are needed and the application GUI remains unchanged. Speech features are provided using a third-party add-on, and the application remains unaware of the presence of speech. For example, a commercially available speech application could be installed and the user could dictate into the application without making any changes to the application.
13.3.2 Designed for speech

This type of speech-enabled application requires minor changes to the GUI. The application is usually aware of the speech components that are directly integrated into the program. Speech features supplements the existing features in the application. This is the preferred mode for a speech design, as it offers flexibility by multi-modal input. This sort of applications can also shorten the learning phase for new users. If both graphical and audio instructions were provided in an application, this would allow users to read the instructions as well as hear them. This would increase the chances of users knowing what to do or say.

13.3.3 Speech Required

In this category, the entire application is designed with speech as the primary user interface. This requires a complete re-write of the interface code, and demands the most amount of work. This mode of speech user interface is often used for telephones and mobile devices because there is no other input mechanism that is easy to use. The application may or may not have a GUI, but many features will be accessible only by speech.
# 14 Grammars

A grammar file contains a collection of rules comprised of words and phrases, which determines what can be recognized from speech. A speech recognition engine uses a grammar to enhance its ability to recognize specific combinations of spoken words and phrases. Grammars can range from simple one-word commands such as “open” or “print”, to more complex sentence structures, for example ordering airline tickets or scheduling appointments.

With dictation recognition, a recogniser uses a large dictionary to match the words, and also performs contextual analysis to ensure that it returns the correct word. Ideally, all allowed phrases in a language could be recognised with this technique. This leads to a huge number of possible phrases. The advantage of dictation is that all legal phrases can be recognized without having to specify the whole grammar first. The disadvantage is that chances of mis-recognition will be very high, at least with today’s technology. The term mis-recognition denotes cases when a phrase is being recognized as a different phrase. A specified grammar is not used for the dictation recognition.

Grammar recognition, used for command and control, is context free. A recogniser only matches against the rule definitions in the grammar, specified for the application. The grammar can be rather limited, only phrases that are relevant for the application are included and the chances of mis-recognition are reduced. The disadvantage is that the grammar has to be manually specified.

There are two types of command and control grammar; static and dynamic. A static grammar is completely predefined and loaded at the start of the application. It is not possible to change any of the rules during runtime. In the dynamic grammar, the contents of the grammar can change during runtime.

## 14.1 Functions of Grammars

Explicitly listing the words or phrases in a grammar has several advantages:

- Limiting vocabulary: The grammar contains only the exact words or phrases to match, shortening searches and improving recognition accuracy.
- Recognition filtering: The engine determines what the word spoken is, and the recognized word or phrase is then matched to the grammar. SAPI only returns a successful recognition event if the grammar is matched. This limits the recognition results to those identified as meaningful to the application.

- Rule identification: When a successful recognition occurs, the rule attributes is passed back from SAPI to the application. If the application must sort the results, for instance in a series of case or switch statements, the rule name may be used.

Microsoft Speech API 5 context-free grammars are defined using Extensible Markup Language (XML).

### 14.2 Extensible Markup Language

An XML grammar uses markup tags and plain text. A tag is a keyword enclosed by angle bracket characters (< and >). Tags occur in pairs, the start tag `<keyword>` and the end tag `</keyword>`. Between the start and the end tag, other tags and text may appear. Everything from the start tag to the end tag is called an element.

For example, all grammars contain the opening tag `<GRAMMAR>` as follows:

```xml
<GRAMMAR>
    … grammar content
</GRAMMAR>
```

**Example 14.1**

A tag may have attributes inside the brackets. Each attribute consists of a name and a value, separated by an equal sign, and the value must be enclosed in quotes.

```xml
<GRAMMAR LANGID ="409">
    … grammar contents
</GRAMMAR>
```

**Example 14.2**

Here, the grammar element has an attribute, called LANGID, which must be a numeric value. The attribute LANGID specifies the language of the grammar, and the value 409 corresponds to the English language.
If one element contains another, the containing element is called the parent element of the contained element. The contained element is called a child element of its containing element. The parent element may also be called a container.

An XML grammar consists of rules. Each rule specifies some user input that can be recognized. The `<rule>` tag defines a rule in an XML grammar. Each rule definition has a name, specified by the `id` attribute. A rule’s name must be unique within the scope of the grammar that contains the rule. The following examples illustrate how to implement a grammar and the examples consider a game of solitaire.

```xml
<GRAMMAR LANGID="409">
  <RULE NAME="new game" TOPLEVEL="ACTIVE">
    <P>new</P>
    <P>+game</P>
    <O>-please</O>
  </RULE>
</GRAMMAR>
```

**Example 14.3**

In the example above a top-level rule called `new game` is defined. The rule is made ‘active’ by default, so the rule is available as soon as speech is activated in the application. The `<P>` tag defines the phrase to recognize and the `<O>` tag defines optional words that could be spoken.

The `+` defines that high confidence for the word `game` is required, to avoid accidental recognition of this important rule. The `–` defines low confidence, to make the command phrasing more flexible.

```xml
<RULE NAME="play card" TOPLEVEL="ACTIVE">
  <O>please</O>
  <P>play the</P>
  <O>…</O>
  <P>RULEREF NAME="card"/></P>
  <O>please</O>
</RULE>
```

**Example 14.4**

In this example rule reference and garbage words are demonstrated. The `<O>…</O>` element specifies garbage words, which allows the user to say for example “play the little ace of spades” without breaking the voice command.
A rule reference is a reference to a rule specified elsewhere. Using rule references is similar to reusable components in an object oriented programming language.

Example 14.5

Here a reusable card grammar is created. The \(<L>\) tag specifies a phrase list, where one of the \(<P>\) elements is spoken. Note that this rule is not a top-level rule, since it is only used by other top-level rules and is not directly recognisable. This rule can be made more effective as follows:

\[
\begin{align*}
\text{Example 14.6} \\
\text{In this solution the suit and rank are defined as own rules.}
\end{align*}
\]
15 The application

The developed application is a version of ActiViewer™, expanded with voice control. The purpose of the application is to allow the user to navigate in a large amount of structured information, and to receive desired parts of the information either visually or verbally. The application is developed in Visual Studio .NET 2003, in the programming language Visual Basic .NET. For the voice control the Microsoft Speech Software Development Kit 5.1 is used.

In this part of the thesis, an explanation will be given on how the application works, and some of the techniques behind it. First the different parts of the interface will be described, followed by a description of the functions and techniques of each part.

The interface of the application consists of four different parts, which will be referred to as:

1. Navigation list
2. Tab control
3. Text and image area
4. Resize button.

See figure 15.1
Chapter 15. The application

15.1 Navigation List

The information amount, which can be displayed by the application, is stored in different xml-files. For all this information there is a separate file which describes the relations between all the files, as a tree structure, and it also contains the URL for each xml-file in the structure. This information about the relations is also stored in xml-format. These files together with a web service are used when the user navigates in the application. See further explanation below.

The navigation list, at the top of the interface, contains a list of items. These items are the siblings at the current level of the tree.

The navigation in these lists works in the same way as in Windows Explorer, where a mouse click on the plus sign adjacent of the item will open a new list containing the children of the chosen item. A click on the item itself will display the connected text and image of the item, in the text and image area.

Figure 15.1 1. Navigation list  2. Tab control 3. Text and image area  4. Resize button

Removing the constant velocity joint housing

1. Tap off the constant velocity joint housing.
2. Use a brass drift.

Note! Only tap the inner ring. Take care not to damage the ball holder or outer ring.
15.1.1 Read Items

The voice commands implemented in the application allow the user to navigate in the same way as described above, but there are several ways to open an item or read its content with the voice commands. The first command to be presented is the “Read items” command. When the user gives this command, the application will read the items in the list, one after another, until the end of the list. The current item is marked blue, to visualize which item is being read.

It is possible to interrupt this reading of items by speaking the command “Stop”. The current row will then be stored and used if the user wishes to continue the reading of the list. The command “Continue” is used for this purpose, and when using this command the reading continues from the row where the reading was stopped. There is also a possibility to step back and forward in the list of items, by issuing the commands “Previous item” and “Next item”. After speaking these commands, the marking is moved to the new item, and a spoken confirmation of which item is the current item is given.

The command “Read items” can be used for several purposes. One use of the command is if the user would like to know which items there are in the list, and which items he or she could open or display information about. The user receives this information by asking the application to read all items in the list. As mentioned before, if an item is possible to open there is a plus sign in front of it, and when the items is read, the application reads this plus sign as well, to inform the user of which items can be opened. Another use of the command is when the user wishes to open an item in the list or display its text and image, see the next paragraphs.

15.1.2 Open

There are two ways to open an item, that is, to display its children in a new list, by voice commands. The first way is to use the “Read items” command. When the application is reading the item to open, the user can either issue the command “Stop” and then say “Open item”, or just use the command “Open item”, without stopping the reading. The current and marked item will be opened.

If the next item in the list is read while the user is speaking the command, the correct item will still be opened. This is because when the user starts to say a command, the row that is current at that moment is stored, and this row is then used to open the correct item of the list. When the speech recognition engine has matched the command as “Open item”, the element is opened, a new list is displayed and the application gives a confirmation that the item has been opened by saying, “Click”. If the setting for automatic reading is turned on, the application will start to read the items in the new list.

The second way to open an item is to use the “Open” command together with the name of the item to open. Thanks to the dynamic grammar used, which contains all
the items in the current list, this solution is possible. If the list items are for example car models, to say “Open S70” will display a new list with the children of this item.

The technique behind the opening of an item in the list is the use of a web service. A web service is a software component that can be used remotely, and is accessed via standard web protocols, such as HTTP or SOAP. In this case, the web service component is used to handle the communication between the application and the xml-file with the tree structure. When the user gives a command to open an item, the application asks the web service component which children the current node has. The web service component uses the xml-file with the tree structure to find the answer, which is sent back to the application and stored as an array. This array of items, or more correctly array of siblings, is displayed in the navigation list. When the speech synthesis is asked to read the items, it will read the first element in the created array, then the second and so on.

15.1.3 Read and display

There are two ways to display an item’s connected text and image, and get the application to read the text displayed. These are of the same principle as the commands used to open an item. The first way is to use “Read items” together with command for displaying and reading the text, “Read content”. Also here the user can choose between first stop the reading or issue the command as the application is reading the items in the list. When the command “Read content” is spoken the item’s text and image are displayed in the text and image area, and the application starts to read the text displayed.

The second way to get the application to read a text and display it in the text area is to speak the command “Read” together with the name of the item.

If the user only wishes to display the text and image of an item, and does not want the application to read the content, the command “Show content” or the command “Show” together with an item’s name could be used.

The technique behind displaying and reading text is the same as the one used for opening items. In this case, the application asks the web service component about the URL to the xml-file containing the information connected to the current item. The component finds the answer in the xml-file with relations and URLs, and returns the URL to the application. The application then fetches the file from the given URL and displays the text in the application. The technique to get the speech synthesis to read the displayed text is handled by a component in the application. When this component receives the content of the xml-file it stores every element of it without the tags in an array. Together with each element a keyword is stored, to be read before the actual element, to notify the user about the structure of the things being read. See further explanation and examples in chapter 16, Process of work.
When the user navigates in the tree structure, the path chosen by the user is stored in a list below the navigation list. This makes it easy for the user to navigate between the different levels of the tree. The commands “Next level” and “Previous level” will move the focus to another level, either down or up in the tree structure, and a confirmation is given to inform the user which level he or she is on. To just receive information about which level the user is at, the command “Which level?” is used.

To show or hide the navigation list, the commands “Open list” and “Close list” are used.

### 15.2 Tab Control

The tab control, to the left in the application, has two tab pages named “Search” and “Speech Settings”. The first page is used to perform a search for a word or expression. The other page is devoted to the options of the speech and also contains a list of all commands to help the user in controlling the application with speech.

The search tab consists of a text field to enter the word to search for and some checkboxes with different constraints for the search. The list box below is for the search hits to be displayed. The search tab functions are not implemented in this version of the application, only the interface is. Read more about the future function of the search function in Continuation of the project, chapter 17.

The speech options at the speech settings tab are “Speech recognition” and “Auto reading”, as checkboxes. The speech recognition checkbox is used to turn the recognition on and off. This can be done in three different ways, by clicking the checkbox, by pressing Alt + S or speaking the commands “Speech recognition on” or “Speech recognition off”.

When speech recognition is turned off, a new grammar file is loaded. In this grammar there is only one phrase for the speech engine to recognize. The application listens only for the command “Speech recognition on”. When the recognition is turned on, the default grammar is loaded into the application again.

The checkbox for auto reading regulates if the items should be read automatically when a new list is opened. If the auto reading is turned off, an item is opened without the new list is being read. To turn the auto reading on and off the user either clicks in the checkbox, presses Alt + A, or uses the voice commands “Auto reading on” and “Auto reading off”.

To shift between the two tab pages the user can say the commands “Search” or “Speech Settings”. The speech settings page is also displayed when the user uses the command “Help”, as the list of commands at the page is considered as the help.
Chapter 15. The application

15.3 Text and Image Area

The text area is the part of the interface where the text connected to the items is displayed. To allow the user to control and influence the reading of the text displayed, there are commands to stop the reading, continue it, repeat a row, step back and forward among the rows and also go to links in the text. When the command “Stop” is used, the application stops reading immediately. If the user wishes to continue the reading, he or she uses the command “Continue”.

When “Continue” is used in this context the application continues reading, from exactly where the reading was stopped. This differs from the usage of the same command when reading items in the list. In the list navigation the continuation starts from the current row, as the items are rather short in comparison to the rows in the text.

When the application is told to read the text, or when it is told to continue reading after a stop, it reads the whole text, until the end of it. If the command “Repeat” is used, the current row is read again, and when the row is read the reading stops. The same method is used for the commands “Next” and “Previous”, when navigating back and forward among the rows, the reading stops after each row read. Only one row at the time is read, in this way of navigating.

There are links in the texts, which lead to other texts, and to follow a link the command “Go to link” is used. This feature is not implemented in this version of the application, so when the command is issued, the user gets spoken information about this.

The image area is placed above the text area and this is where the images connected to the text are displayed. Not all texts have images, and some texts have several images. In the upper part of the image area four buttons and a label are placed. The three leftmost controls are handling the navigation among the connected images of the current text. The label in the middle of the three controls functions as an image counter, which shows which the current image is and the number of images available for the current text. When text is displayed, the first image connected to the text is not shown and the counter shows “0/2”. This is to avoid time consuming loading of the image, if the user does not want to see the image and has a slow connection. The two buttons to the left and to the right of the image counter are used to step back and forward among the available images.

**Figure 15.2** Controls Previous image, Image counter and Next image

The fourth and fifth controls are both buttons, one for printing the image displayed; the other is used to show the user where in the text the current image belongs. By pressing this button when one of the images is current, the text is scrolled down to
the part of the text, which the image belongs to. The function of these two rightmost buttons has not been implemented in this version of the application.

![Print and Image reference](image)

**Figure 15.3 Controls Print and Image reference**

The intention is to add appropriate icons to the text area as well, which would be done in the continuation of the project.

### 15.4 Resize button

The red square in the middle of the interface, between the text area and the image area, is used to resize the different parts of the application interface. When passing over the square the default cursor transforms to a size all cursor, to indicate its function. When moving the square, the text area, image area and tab control is being resized, and the user can decide the proportions of the different parts. The button is moved by the drag and drop method. There are two commands to change the dimensions of the text and image area. If the user says, “Show only text”, the text area will cover the complete application area except from the navigation list, and the “Show only picture” makes the image area covers the complete area.
Voice navigation was something totally new to the author, and the first step in the process was thereby to search for information about speech, speech-enabled applications, speech software etc. Information was found about Microsoft Speech SDK, a software package used to build speech-enabled applications. SDK contains Microsoft Speech API (SAPI), a speech recognition engine (SR) and a text-to-speech engine (TTS). This software package can be downloaded and used for free and our decision was to use this for the development of the application.

Initially some samples were studied to gain understanding and ideas of the functions and possibilities of speech recognition and speech synthesis. All samples were made for Visual Studio 6.0, but the objective was to develop the application in Visual Studio .NET, as it is the latest version of the program, and also because of the benefits of knowing about the new .NET technologies. The decision was to try to develop speech in the latest version, even though the samples were for a different version.

A primary structure was designed of how navigation with speech in a large amount of information or a large document could be done. The next step was to refine this structure, and a sketch was developed of how the navigation should work.

The next step was to develop some speech-enabled applications, to get more knowledge of the speech recognition and synthesis. Three small test applications were developed, one with speech synthesis, i.e. text-to-speech, one with speech recognition and one with the both techniques combined. The speech synthesis worked very well, the application read expressions, files, etc. as specified. Unfortunately, it did not work that well with the speech recognition. When speaking into the microphone, nothing happened at all. No response was given from the application, and it seemed like it did not get any input from the microphone. To find the solution of this inconvenient problem, SDK was installed on another computer and the test applications were tested there with success. The reason for this was probably that the laptop used did not have a sound card good enough to deal with speech-enabled applications. The project continued on another computer with better performance.

The development of the final application started, first with the navigation in the list at the top of the application. The voice commands “Read items”, “Read content”, “Read” + list element, “Open item”, “Open” + list element, “Stop”, “Continue”,
“Previous level” and “Next level” were implemented as buttons. When every button functioned satisfyingly, the grammar was to be created.

At a first step an ordinary static grammar was built in XML, with the voice commands mentioned above. However, this solution had to be reconsidered. If the grammar were to be static, it would not be general for all contents. The grammar could not be used when another information set is used in the application, without requiring modifications of the grammar. Another drawback of a completely static grammar is that it would be a tremendously large grammar, as all the items in every list have to be implemented in the grammar. The application has to be general, to fit all kind of contents, so another solution had to be found.

One solution would be to skip the possibility for the user to use the commands “Read” and “Open” together with list elements. This would require the user to use the command “Read items” and when the speech synthesis reads the desired element to open or read the user could say “Open item” or “Read”. As this solution would be very inconvenient and time consuming for the user, a decision was made to solve the problem by using a combination between static and dynamic grammar.

All commands for the navigation were specified as a static grammar, and all items in the lists were specified in a dynamic grammar. Every time a new list with items is opened, these items are loaded as the actual grammar at that point. This makes the size of the grammar manageable and the defining of the different items in the grammar is made automatically. To make this dynamic grammar work it had to be implemented as XML in the application, and then saved into a file called grammar.xml.

The next step in the development was to implement the voice commands to replace the buttons in the application. To enhance the results of recognition, the speech recognition engine was trained to understand the voice of the author with its nuances. The recognition results increased a lot after the training of the engine.

When the voice commands implemented were tested a problem was encountered. The recognition engine does not understand numbers written as digits, only as text. Some of our list elements in the temporary information of our application are years and car models, for example 2001 and V70. It would look odd and be inconvenient to write these elements as text, for example ‘two thousand and one’ and ‘V seventy’.

A solution to this problem was to make a translation from digits to text. When the application encounters a digit, it translates it into text for the recognition engine. At this point yet another problem was encountered. The translated digits were 0-9, and when two or more digits were used together, as in 70, the recognition engine did not understand it. A solution to this problem could be to make a loop that checks how many digits the number contains and then translate this number to text. As this application is meant to be as general as possible, we had to not only translate the few numbers that occurs in the information used in this project, but all the numbers.
up some limit, for example 2010. This solution is too work and time consuming, and
is not practical. Our decision was to read every number as single digits, to keep the
application general for all contents. Probably Microsoft will enhance its speech
recognition engine, so that it understands digits together as numbers.

At this stage, the application understood voice commands and it read lists. The
navigation was enhanced with additional commands, such as “Show content”,
“Show” + list element, “Previous item” and “Next item”. To make the application
easier to understand, spoken error messages were added. The application will seem
friendlier if the user does not get the same error message over and over again. For
every error that can be made, for example to give the command “Open” when the
actual item can not be opened or to give the command “Previous item” if the user
already is at the top of the list, different messages are given depending on the order
in which the same error is repeated. See the example below.

<table>
<thead>
<tr>
<th>User</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Open v70”</td>
<td>“Sorry”</td>
</tr>
<tr>
<td>“Open v70”</td>
<td>“Sorry”</td>
</tr>
<tr>
<td>“Open v70”</td>
<td>“Sorry, this item can’t be opened”</td>
</tr>
<tr>
<td>“Open v70”</td>
<td>“The item contains no information.”</td>
</tr>
</tbody>
</table>

Example 15.1

Error messages were implemented for the commands “Open item”, “Open” + list
item, “Previous item”, “Next item”, “Previous level” and “Next level”.

For the commands “Next level” and “Previous level” a confirmation was
implemented, to inform the user which the current level is.

As mentioned in the chapter “Limitations of speech technology”, it is a good idea to
have the possibility to turn the speech recognition on and off. This function was
implemented and activated using a checkbox in the application. An alternative to
turn of the auto reading of the list items, when an item has been opened, was also
implemented. The commands “Speech recognition on”, “Speech recognition off”,
“Auto reading on” and “Auto reading off” were developed to check and uncheck
these checkboxes.

At this point in the development, when the voice navigation functioned satisfactorily,
efforts were made to work on the reading of the files. The easiest way of handling
the speech synthesis, was to divide it into two parts; one speech synthesis object to
handle the reading in the navigation list, which is already added, and to add a
second object to handle the reading of the texts displayed in the application. When
this solution was tested, it turned out to be a problematic solution. The two speech synthesis objects could not function independently from each other, which resulted in an incorrect solution and several strange bugs in the application. Because of this the development had to be limited to only one speech synthesis object. This lead to more complex programming, as flags had to be used to separate list reading and text reading.

The first step in the file reading development was to expand the already existing information in the application, to make it more realistic and easier to navigate in. Xml-files were built with contents from the Volvo database Nevis and imported the files to the application AID, an application developed by Information & Media AB. In AID the files were structured into a tree. The files were built in xml, to structure the content in a convenient way. If we consider an ordinary text file, without any structure, it would look like this:

```
Installing the mass airflow sensor. Note! Apply water to the O-ring in the air cleaner cover to facilitate installation of the mass airflow sensor. Install the new mass airflow sensor. Press the mass air flow sensor into place. Install the screws. Tighten. Install the inlet hose on the mass airflow sensor. Tighten the hose clamp. Press the connector for the mass air flow sensor into place.
```

Example 15.2

By specifying and using xml tags to structure the information, it will be easy to process the data and the text will be easier to read. The tags can represent for example headings, numbered lists, dot lists, links or images. Appearance properties for the different tags, such as font, size and spacing are specified in an xsl-file. Xsl stands for Extensible Stylsheet Language and is used to determine the appearance of the content specified in the xml-file. In this way the structure of the data is separated from the layout aspects, which is one of the big advantages of the use of xml. The file example above can be coded in xml as follows.
Chapter 16. Process of work

Example 15.3

The first two lines in the example above specify that it is an xml document, and that it uses an xsl-file as stylesheet. The following tags structures the content and together with the xsl-file the text will look as the example below.

**Installing the mass airflow sensor.**

*Note! Apply water to the O-ring in the air cleaner cover to facilitate installation of the mass airflow sensor.*

1. Install the new mass airflow sensor.
2. Press the mass air flow sensor into place.
3. Install the screws. Tighten.
4. Install the inlet hose on the mass airflow sensor. Tighten the hose clamp.
5. Press the connector for the mass air flow sensor into place.

Example 15.4

When the tree had been imported in the application, and every element had an adherent xml file, the reading of these files could be implemented. When testing the abilities of the speech synthesis to read xml-files, the following problem was encountered. If the speech synthesis is told to read an xml-file, it reads everything in
the file, the xml specification, the tags, the content etc., without any structure or pauses. An example of how the synthesis is reading the xml-file is seen below.

“Less than question mark xml version equals to quotation mark one point zero quotation mark encoding equals quotation mark iso hyphen eight five nine hyphen one quotation mark question mark greater than …..less than heading greater than installing the mass air flow sensor less than slash heading greater than…..” etc.

Example 15.5

To solve this problem and to be able to navigate in the file, the content of every row of the file was stored, without the xml tags, in an array, and the speech synthesis could then be told to read the content of the array. This allows the user to stop the reading, continue from the row where the reading stopped, to step forward or backward in the file etc. The problem in this solution was that the user receives the information in the file, but loses the structure of the file seen on the screen. The user gets no idea if the information given is a heading, a numbered list or a link. This problem was solved by keywords spoken when a specific tag is encountered. When a row is about to be stored as an element in the array, the tag encountered generates a keyword, which is stored together with the text of the row in the array. The results of this method are shown below. The keywords spoken are marked in bold font. The speech synthesis will say:

“ Heading: Installing the mass air flow sensor.
Note! Apply water to the O-ring in the air cleaner cover to facilitate installation of the mass airflow sensor.
Step 1 Install the new mass airflow sensor.
Step 2 Press the mass air flow sensor into place. “ etc.

Example 15.6

With this solution the user can imagine the structure of the information without looking at the screen.

The commands to navigate in the content of the files, i.e. the array, were first implemented as buttons. The commands needed were “Stop”, “Continue”, “Repeat”, “Next” and “Previous”. As the commands “Stop” and “Continue” already are used when navigating in the Navigation list, the first thought was to call the commands “Stop list”, “Continue list”, “Stop text” and “Continue text”. These commands would be harder to remember and less convenient to use. After reconsideration the decision was to use the same commands for both situations and separate the actions by using Boolean flags for the different cases.

When the button commands functioned properly, the grammar was expanded with these commands and the implementation of the voice commands instead of the buttons was made.
To enhance the interface, a tab control was added, containing to tab pages. One for speech settings, where the checkboxes mentioned earlier was added, one for a search function, which was to be implemented later. A list of the available commands was also added to the speech settings tab page. Commands to change tab page was implemented as “Speech settings”, “Search” and “Help”.

The content of the xml-files was extended with links to images, to other parts in the same file or to other files. The command “go to link” was added in the grammar, but not implemented.

The display area was divided into two parts, one for displaying text, the other for displaying images. To display images the software Leadtools Raster Imaging Pro 13.0 was installed and the commands “Show only text” and “Show only picture” was added and implemented. To step among the images belonging to a text, buttons and commands were added. Both the buttons and the commands are named “Previous picture” and “Next picture”.

The final step was to navigate in the application to find bugs or errors, and to correct these. All the code of the application was commented and several smaller adjustments were made.
Chapter 17. Continuation of the project

17 Continuation of the project

If the project were to last longer than 20 weeks, several enhancements of the application could be done. This section describes the functions that would have been implemented if the project were to be continued.

If the user knows which piece of information he or she would like to see or go to in the information amount, for example to section “53 Parking Brake”, a good solution would be to have commands to let the user go directly to this section, without stepping through every level of the structure. This solution would save time for the user and is more convenient to use.

There are two problems to solve if implementing this function. The first is that in the current application the grammar consists of the available commands (the static part) together with the items in the current list (the dynamic part). This means that a command containing an item from another list than the current list, would not be recognized. So to implement this function, another grammar structure has to be considered. The second complication is to figure out for which car model and year the section “53 Parking Brake” should be opened.

Another useful function is to state level names when stepping among the levels already visited, instead of repeating the commands “Next level” and “Previous level” several times. The level names are found in the lower part of the Navigation list. This function is used when making a large step over several levels, to make it more convenient and less time consuming. In the example below, imagine that the user desires to go back to level “2002”, it is easier to state “2002” than “Previous level” four times.

Figure 17.1 The lower part of the navigation list

In the original version of ActiViewer™ the links in the text and the image links are working properly. In the demonstrator, this feature is not implemented yet, but will be in the continuation of the development of the application. As mentioned earlier in this thesis, the print button and the image reference button has no function implemented in this version of the application. These features will be implemented in the next version of the application.
Chapter 17. Continuation of the project

When the user speaks a command to display text and images of an item, the text is displayed in the text area and the images are available to be displayed in the image area. The image counter shows the number of images available and the number of the image shown at the moment. In the current version of the application, when text is displayed the first image connected to the text is not shown, the counter shows “0/2” to avoid time consuming loading of the image, if the user does not want to see the image and has a slow connection. An enhancement of this feature could be to have a setting, where the user chooses whether he or she wants to display the image directly or not, depending of the users connection.

As mentioned before, all the commands available to use are listed under the tab “Speech settings” in the tab control. An alternative to this list could be to ask the application about which commands that are available and the speech synthesis starts to read them from a list. This function is a good alternative for the user without ability to see the screen.

The xml-files containing the text and images to be displayed are of varying size. Some of the texts are just a few rows long and some of them are much more complex than that. To give a user without ability to see the screen, a hint about the size of the document that will be read, some information of the size of the text could be given. This could be stored as meta data in the beginning of the file.

The search function presented in chapter 15.2 is just an implemented interface, without any function behind the interface. This would be considered if the project were to be continued. When a search is performed the search hits are displayed in the list box “Search hits”. With a click on the desired item, the information containing the search hit would be displayed in the text and image area.

The objective of the continuation of the project is to implement all the existing functions of the original version of ActiViewer™, and to complete the solutions mentioned above.
18 Voice navigation: Discussion and conclusion

The results of this part of the project are very satisfactory. The speech recognition engine manage to recognize words and commands much better than expected, and the text-to-speech engine is easy to handle and use. However, the quality of the reading of the text-to-speech engine could be better. The development has come far in speech technology, but this is a clear drawback. With the current content of the demonstration application it is all right to listen to the speech synthesis, thanks to the small amount of text in every file. When larger amounts of text will be displayed and read, it will be rather tedious to listen to. Hopefully Microsoft will enhance the quality of the speech synthesis in the near future.

Concerning the grammar, it was noticed that the longer the command strings are, the less frequent are the mis-recognitions of the recognition engine. An example of this occurred with the use of the command “Open”. The recognition engine recognises anything it actually does not recognise as “Open”; words not specified in the grammar, a slam in the door or when putting the microphone down at the desk. This made the application “run away” on its own, when it managed to open item after item, going further down in the tree structure. The problem was satisfactorily solved when the command “Open” was replaced by the command “Open item”. The drawback of having longer commands is of course the inconvenience for the user. Longer commands are harder to remember and more tedious to state. A good command has to be found, not too long, but long enough to avoid mis-recognitions.

One of the primary objectives was to make the application as general as possible. This objective has been met, and everything in the application is general for any contents, except for the DTD used for the xml-files. The other objectives have also been met, there is a version of ActiViewer\textsuperscript{TM} where the user is able to navigate with voice commands, and the speech synthesis is reading texts. The application is controlled by natural speech and the user is given helpful error messages when needed.

This part of the project has been successfully performed, and the results agree with the objectives set in the beginning of the project.

The use of speech as one of the input methods to an application will probably become more useful as the technique becomes more common. And as the software
for speech recognition and speech synthesis constantly improves, the interest for speech applications will increase.

But the future has more to offer than just navigation in certain applications. The user of a speech system will be able to tell the computer what to do, to activate several functions in the car, television and microwave oven by voice commands. In some new luxury cars, speech control is already available. Especially the automobile industry will become a big market for speech control, there will be no more fumbling for the right switch on the radio or dashboard. A simple command will instead activate or inactivate a function.

Another feature for future use is speaker verification, which can be used in security systems. Nowadays the technique can be used if the sample of users is small, and the vocal differences are significant, such as high and low pitch, or male and female. Otherwise the technique is not sufficient yet.
19 Conclusion of master thesis

In this thesis I have treated two major areas, navigating in an interface with eye and head movements and navigating with the voice. The project has been going on for 20 weeks, where I spent 3-4 weeks working on the first part and the rest on the second part. The conclusions drawn from the different parts can be read in chapters 7 and 18.

The conclusion drawn from the complete project is that the work has been very interesting to perform, and I have learned a lot of it. My personal objectives were to implement something to get more skills in programming and to develop something linked to future use of computers. I consider that I have achieved these two objectives by this project, and I am satisfied with the results of it.

For more information about the areas covered in this thesis, I refer the reader to chapter 20, the bibliography.
20 Bibliography

Due to the nature of this project, a lot of the information regarding both theory and implementation has been found on the Internet. The most important documents and websites are found in this chapter.

Books:


Articles:


   http://www.idg.se

    http://www.idg.se

    http://www.idg.se

    http://computersweden.idg.se/

    http://computersweden.idg.se/

    http://www.unisys.com

    http://www.naturalpoint.com

    http://www.generation5.org

    http://www.generation5.org

[18.] Grammar reference, XML Speech grammars, Be Vocal Inc
    http://cafe.bevocal.com

[19.] Nuance: Speech Application Design
    http://www.nuance.com

[20.] Voxit, VoiceXpress™ 5.0 – Tala klartext med din dator, product description
    http://www.voxit.se

[21.] TrackIR™EG, product description
    http://www.naturalpoint.se

[22.] Dragon Dictate: Product specification, Frölunda Data AB
Chapter 20. Bibliography

http://www.frolundadata.se

[23.] Visual Studio .NET 2003 Documentation: Deploying a Windows Application

[24.] Smart Eye Mouse Ground Pro, Documentation

[25.] Smart Eye ActiveX, Documentation

Documents from www.microsoft.com/speech

- Microsoft .NET Speech Technologies
- Technology Overview
- Programming Approaches for Speech Enabled Applications
- Speech Design: When and how
- Design Guidelines for Voice User Interfaces

Documents from Microsoft Speech SDK 5.1 Documentation:

- Getting started for first time users
- Automation overview
- Text grammar format overview
- Designing grammar rules
- Dictation
- Text to speech
- Command and control
- SAPI 5.0
- XML TTS tutorial