TR-IT-0253

A Graphical User Interface for Chatr

Robert Grau

February 1998

Chatr is a powerful system and I like the quality of the speech synthesis, but the Unix style command line interface is not intuitive and the current code (Feb 98, V0.93) is not at all easy to understand. To improve the first disadvantage and to encourage some work on the second I had some ideas to build a native Windows 32 bit application with a graphical user interface. Due to my short stay at ATR (14 weeks) and the fact that I had to specify everything by myself it was obvious that I could not finish the job, so I will provide this documentation to help other people understanding my code without having to learn Visual C++ for some months.

Contents

Glossary

Button	Typical Windows control item that can be "pressed".
C++	Programming language built on C that features encapsulation, inherritage, polymorphism, see [STR] or [KRU].
Check Box	Windows control item that looks like ☒.
Class	C++ structure that can have member functions and variables.
Combo Box	Field (editable or not) with an arrow at the right border that drops down a list of items.
DOS	Saying "DOS" the most people think of MS-DOS, "Microsoft Disk Operating System". Old fashioned 16Bit operating system with a command line interface. See my introduction to Windows in paragraph 1.3.
Endian	Little/Big Endian means least/most significant byte FIRST (the word "endian" is a bit ambiguous). See 3.1.3.
Hungarian Notation	Building up names for variables or objects like "typeNameOfVariable" (e.g. int nLoop;) Integer n String str Bool b Double d Float f Filehandle file m_... for member variables (e.g. m_bValid) g_... for global variables (e.g. g_dPi) C... for class declarations (e.g. CMainFrame)
Icon	Symbol that activates a specific program or process.
List Box	Scrollable data field
Message	Information that is used in a Windows program to communicate between or within processes.
Method	Other name for member function of a class.
Microsoft	Huge software company from Redmond, US.
MFC	Microsoft Foundation Class – a huge library that provides many powerful functions and classes for programming, you only have to know it's possibilities.
Object	allocated memory that has the qualities of a corresponding abstract declaration like a variable type, class or structure. E.g. the command "int i;" will create an object of type "Integer" named "i".
Radio Button	Windows control item that looks like ⊙ that can be grouped.
Thread	Other word for process. In a Windows system multiple threads can be executed "simultaneously", but it takes some special programming techniques to synchronize them (see [KRU] or 2.5).
Visual C++	C++ Compiler from Microsoft with a very powerful GUI.
Wave file	File with a specific format. Windows uses RIFF files with the extension .WAV that have a header and may contain additional info at the end.
Windows	Graphical User Interface and Operating System from Microsoft.

1 Introduction

At the beginning of this report I want to introduce myself. My name is Robert Grau, I'm a native German electrical engineer with emphasis on information technology, acoustics and human machine interaction and I have written 16Bit Windows Applications before my stay at ATR. I am not a believer in Microsoft, but for practical reasons I use their powerful applications.
My scientific background is the Institute for Human Machine Communication (MMK) of the Munich University of Technology (TUM), the former institute of acoustics (former head: E.Zwicker). Members of this institute are M.Lang (general chair ICASSP 97), H.Fastl (psycho acoustics), Terhardt (acoustics) and G.Ruske (speech processing). My last project was a word level speech concatenation system for BMW car navigation systems with the emphasis on natural sound.
My stay at ATR was very short, so I was asked to improve some programming aspects of Chatr.

1.1 Objectives

My task at ATR was to build a Graphical User Interface for the Unix based speech synthesis software Chatr using Microsoft Windows and to design interfaces for software modules to simplify building in classic Chatr functions later.

My GUI should be able to access Chatr modules (these should be almost platform independent), and it should be possible for a C++ programmer without or little Windows programming knowledge to modify my code. Therefore my code has to consist of clearly written and documented modules completely separated from the Chatr Kernel. That Kernel has to be written by someone else redesigning the old Chatr C code with C++ and to make the Chatr core run on other systems, the code should be as portable as possible.

So I created a Graphical User Interface with many windows, buttons and menus described in Chapter 2, that should be easy to use for both newcomers and experts. Paragraph 2.6 deals with the modification of the existing interface.
To give other programmers an idea how to build up an interface between my code and their code I defined some Chatr modules, some of them with dummy code just to make my application run, others with real code like the Concat module with the streamed audio output. Check out Chapter 3.
To summarize what I have done and to give an outlook on future activities I wrote Chapter 4. After a short bibliography (Chapter 5) I will finally list a few receipts how to obtain some goals using Visual C++ in Chapter 6.

1.2 My first Impression of Chatr

Before talking about Windows I want to write some lines concerning Chatr version 0.93 11/97. Chatr is a really powerful speech synthesizer and the best one I have ever heard, but its strength and weakness are the same: the database and the unit selection.
As a German native speaker I had a closer look at the KKO database with its charming voice of Klaus Koehler. The database consists of several recording sessions with different recording methods that result in some audible discontinuities. Due to Koehler's characteristic (and in my opinion a bit British) way of pronouncing, the many similar sentences of recording and the English intonation module the synthesis of a sentence like "an der naechsten Kreuzung links abbiegen" ("at the next crossing turn left") sounds awful. Maybe new units from spontaneous speech and a German intonation module can help to improve this.

Anyway, with the parameters recommended by my predecessor Caren Brinckmann [BRI] the generated speech has some jitter in it due to prosodic discontinuities at concatenation points. To improve this I changed the weights of unit costs and join costs from 5.0 and 1.0 to 1.0 and 5.0. The result is obvious but in my opinion significant: the utterance will sound fluent due to the better joining units, the prosody will sound natural because many units will be taken in a row from a database utterance, but - of course - the f0 time function might be far away from the given or predicted target.

To judge by yourself, select the database `(speaker_KKO)`, and try:

```
(SayText "Frohe Weihnachten und ein gutes neues Jahr")
```

Now change the weights by :

```
(set nus_kko_params '((join_wt 5.0) (unit_wt 1.0)))
```

Try the first line again. Do you agree with me ?

By the way, there were two things I didn't like (therefore I started this Windows project):

- it took me several days to understand how to make Chatr do what I want due to it's not very intuitive command line interface and the huge documentation that has no pictures in it (I was told the latter has technical reasons).
- when I had a look at the code I was just frightened by the many different programming styles, modules, global variables, hard coded variables and comments saying "this is an old function".

So my first thought was: "someone should rewrite everything in C++ code with specified interfaces and encapsulated data", and the second thought was: "a graphical user interface (like the useful ChInspect application) would be nice". The third thought was: "I definitely don't want to be the one for the first job" so I decided to do the second as I had some experience with Windows applications :-)

1.3 Zen and the Art of Windows Programming

"Working with Windows requires deep inner peace of your mind"

There are many people that don't like Microsoft and their aggressive marketing strategy. But if you take a look at some pieces of software like Visual C++ (here Version 4.0) you can't deny that the Graphical User Interface (GUI) has some advantages compared to a command line driven interface, often experienced with Unix systems. So building a Windows application is not only useful due to Windows-is-the-standard-reasons, but also because of some aspects of human machine interaction.
People who are not familiar with Windows applications should know the following facts:

- There are several versions on the market (Windows 3.1x, 95, NT3.51, NT4.0 and the new 98), I worked with Win95 because by now it runs stable and the system requirements are not too high, so it will run on laptops, too (the application should also run under NT).
- DOS is an old 16Bit operating system that interacts somehow with Windows 95 (although Microsoft will deny that), but without special tricks it can't access Windows 95 functions.
- There are several improvements in Win95 compared to Win3.1x that make the life of a programmer MUCH easier (e.g. memory management: Win95 uses 32Bit addresses, Win3.1x uses 16Bit segment addresses and audio functions need locking of memory), so if you got frustrated with earlier versions try out Win95.
- According to David Kruglinsky [KRU], the learning curve for Windows programming and C++ is several months, but to understand this code you should know C++ and how to operate standard Windows applications.

If you are still interested in Windows programming (or if you have to) get the Kruglinsky book first. Anyway, here is some information on programming with Visual C++:

- Your program has to provide functions that handle "messages" from the operating system (so exactly the other way round like "normal").
- The Application Wizard provides a code skeleton and many default functions and handles that won't be visible. It will create several classes with member functions and variables.
- There are graphical tools to build resources like dialogs.
- Every item of your graphical user interface has an ID value that has to be mapped with a specific method.
- If you are not satisfied with a default handler you have to override it – in other words: if the automatically generated code won't work you have to do it "manually".
- To edit a function double-click it's name in the tree style Class View, this is very effective.

Although Visual C++ is very powerful there are many traps for the newcomer, so:

- backup VERY often, because it's hard to undo wizard actions (you have to erase files and edit the .clw file manually – like in Goethe's Faust: "The devils I called").
- if you want to create a graphical resource, specify FIRST what you want to do.
- make use of the online help, if you got used to it it's extremely helpful.
- if you got stuck don't worry, this happened to me all the time.

Enjoy.

1.4 A Quick and Dirty Guide to C++

Writing C++ code is not very difficult, but if you are used to C you have to change your point of view a bit. First of all get a C++ book (to understand it, the short paragraph in the appendix of the 4.0 Version of the Kruglinsky book [KRU] should be sufficient).

C Code

In C code you have many functions and maybe a header file for the definition of all the global variables. The most people group functions by using source code files organized in directories. So if you are new to a program the only hints you have are the filenames and the function names (and the comments if the programmer(s) was/were nice).
If you look at the header file(s) you will see all global variables, but you don't know which functions will access them (only if the variables have names like "accessed_by_functionx_that_does_this_and_that").

C++ Code

In C++ you will encapsulate functions and data. So if there are functions with a similar purpose like unit selection, we can put them together in a class. There are functions that may be called from outside the class and there may be functions that are only called within our class. The former are candidates for public member functions, the latter for protected ones. Then there is data that should be available for all these functions like global data, but some of these variables might be only necessary within the class. These are candidates for protected member variables. If there are variables that might be accessed from outside you can declare them as public. Initialization can be done in the constructor or in an extra function.
So if you look at the class definition you will see the class name that should have something to do with the functionality of the member functions (e.g. class CConcat). You will identify public member functions and you know you can use these from outside. The protected members are not interesting for you if you just want to use the module without changing it, so you can forget them (!). It's the same with the member variables, if you are only interested in the interface of your class you don't have to know protected or private data.
So one advantage of C++ is obvious: you can read the class declaration to understand the classes interface even if the documentation is not available or written in a foreign language.

Here are some general guidelines:

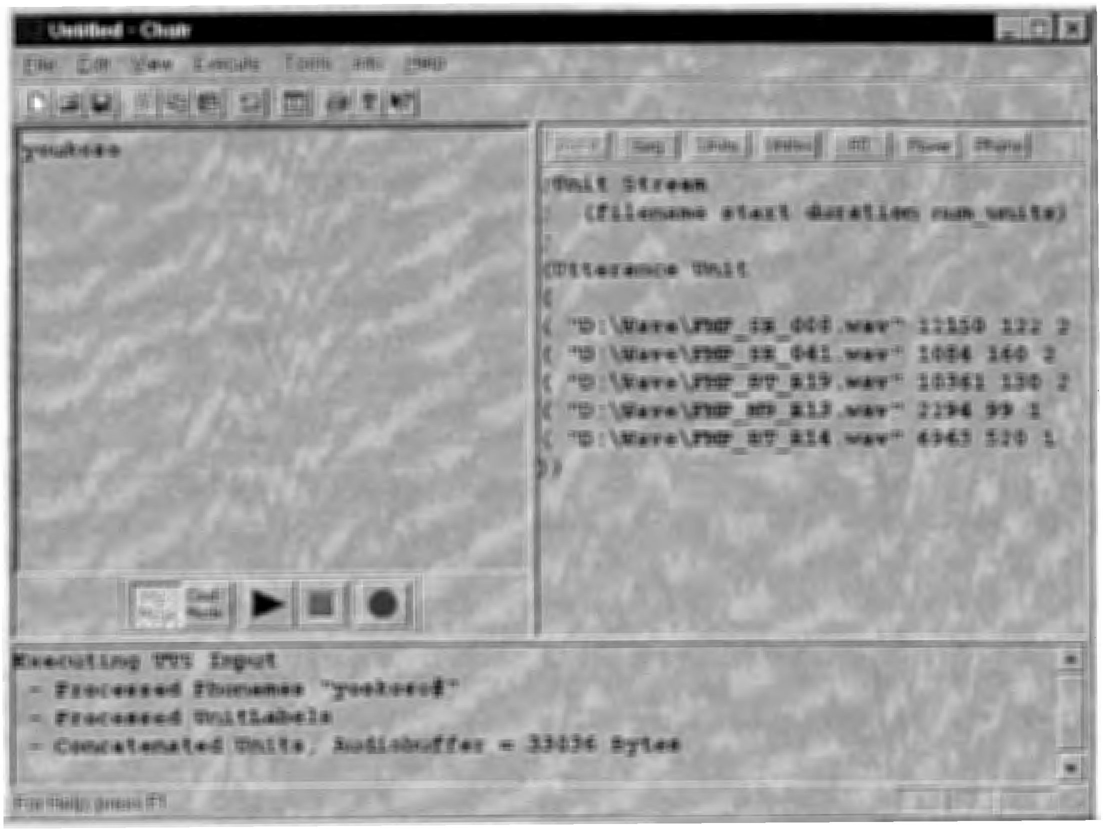
- Use Hungarian Notation for your variable names, because it makes the reading of code easier (→ Glossary)
- Use as few global variables as possible (everybody says this but nobody does it ☺).
  In our application I used one global object of a class with member variables that could be accessed like this: g.strConcatMethod (I omitted the m_... in the declaration of the member variables so that the expression looks similar to g_...).
- If you design a clearly written class definition with public and protected members, names that are easy to understand and comments everyone that has to read your code will be very happy because this task will not be more difficult then necessary.

2 GUI

This chapter includes the specification of the GUI I created for our Chatr Application and will show some aspects of the coding.

2.1 Frame Window

Who has ever worked with Visual C++ will notice that its complex and powerful user interface inspired my to create the main screen of our Chatr application.



```
youkoso
```

```
;Unit Stream
;  (filename start duration num_units)
;
(Utterance Unit
(
( "D:\Wave\FMP_IS_003.wav" 11150 122 2
( "D:\Wave\FMP_IS_041.wav" 1084 160 2
( "D:\Wave\FMP_ST_819.wav" 10361 130 2
( "D:\Wave\FMP_MS_813.wav" 2294 99 1
( "D:\Wave\FMP_ST_814.wav" 4963 520 1
))
```

```
Executing TTS Input
 - Processed Phonemes "youkoso#"
 - Processed UnitLabels
 - Concatenated Units, Audiobuffer = 33036 Bytes
```

The application's frame window is resizable and contents following elements:

- a menu bar with "pulldown" menus providing items like "Open File", "Undo" etc.
- a toolbar with buttons as shortcuts for menu items (can be hidden)
- an editable input window with a toolbar at the bottom
- an info window (read-only) with a toolbar for displaying information
- an output window for displaying info about the running processes (like std out)
- a status bar for displaying help information

2.1.1 Input Window

The input window provides a RichEditControl object that is in principle an editable field we can use for typing in text. At the bottom of the window a tool bar (I called it "PlayBar") offers some buttons (these are discussed in 2.1.4). Two of these buttons decide whether the window content has to be interpreted as a ASCII-Text or a script of command lines.

The content of the window can be saved and restored. The standard copy & paste operations work, also marked selections from the other windows can be copied with drag & drop. I added a simple undo function. Building in new functions like find and replace are just some lines of code because everything is provided by the MFC and you only have to activate existing functions.

A RichEditControl is capable of varying fonts, colors and styles, so these features can be used in future versions to highlight unknown words in a text or errors in a command script. Using this features the user can type information on the utterance of an ASCII-Text by using colors, fonts or font styles like Tobi labels.

In this version of Chatr, the Execute button will start a new process that will access the text of the window (or a selection) and will pass it to the Chatr core for execution.

2.1.2 Info Window

In the existing Unix version of Chatr the user has to type a command sequence like (Save UnitLabels "-") to get the specific data. He can display this data in a teletype manner on the standard out device or can write it to a file. Unlike the Unix version our GUI provides menu items or buttons in a toolbar that activate functions to visualize the requested information in the info window. The Info window can display this information in a scrollable field, so this is a more comfortable way to look at the data. In the current version of the GUI the user can access UnitLabels, UnitLabels+, Segments, F0 and Power.

Of course it is possible to copy a selection to the input window by copy&paste, but you can also make a selection with the mouse and drag&drop it to the input window. If you want to save the content of the info window to disk, you don't need to copy it to the input window because I built in an extra menu item (Info→Save As) to serve for this purpose.

The info window also provides context sensitive information on the continuation of a commandline, so the user can activate an assistant that will show him valid input possibilities (see 2.3.2).

2.1.3 Output Window

The output window serves as a standard out device where information on the current process is displayed in an old-fashioned teletype manner. The window content will be cleared at the beginning of a process execution, is read-only and can be scrolled. Of course selections can be copied to the Clipboard.

2.1.4 Menubar and Toolbars

The following table lists all items from the Menubar together with the Shortcuts. If a corresponding Toolbutton is available, a cell in the right column is checked (M=Mainbar, I=Infobar, P=Playbar).

Menu	Description	Hotkey	M	I	P
File→New	Create a new document	Ctrl+N	☒		
File→Open	Opens an existing document	Ctrl+O	☒		
File→Save	Save the active document	Ctrl+S	☒		
File→Save As	Save the active document with a new name		☒		
File→Print	Print the active document	Ctrl+P	☒		
Edit→Undo	Undo the last action	Ctrl+Z Alt+back	☒		
Edit→Cut	Cut the selection and put in on the Clipboard	Ctrl+X Shift+del	☒		
Edit→Copy	Copy the selection and put it on the Clipboard	Ctrl+C Ctrl+ins	☒		
Edit→Paste	Insert Clipboard contents	Ctrl+V Shift+ins	☒		
View→Toolbar	Show or hide the Toolbar				
View→Status Bar	Show or hide the status bar				
View→Default	Set default configuration for split windows	Ctrl+D	☒		
View→Info	Maximize input window				
View→Info	Maximize info window				
View→Output	Maximize output window				
Execute→Start	Start execution	Ctrl+E			☒
Execute→Stop	Stop execution	ESC			☒
Execute→TTS	Switch to TTS mode				☒
Execute→Comand	Switch to comand mode				☒
Execute→Synth	Start unit selection for existing target				
Execute→Say	Start audio output for existing unit list				
Execute→Macro	Execute user specific macro	F2			
Tools→Inspect	Start ChInspect application		☒		
Tools→Record	Start Voiceinput application				☒
Tools→Options	Change Chatr properties		☒		
Tools→Default Options	Set default configuration				
Tools→Load Options	Load configuration from file				
Tools→Save Options	Save current configuration to file				
Tools→Wave Edit	Display waveform				
Info→UnitLabels	Display current UnitLabels			☒	
Info→UnitLabelsPlus	Display current UnitLabelsPlus			☒	
Info→Segment	Display current Target Segment			☒	
Info→F0	Display current Target F0			☒	
Info→Power	Display current Target power			☒	
Info→Phonemes	Display current Target phonemes			☒	
Info→Say	Play content of audiobuffer			☒	
Info→Save	Save content of info window to file			☒	
Help→Help Topics	List help topics		☒		
Help→Assist	Command line input assistant			☒	
Help→About Chatr...	Display program information, version number and copyright		☒		

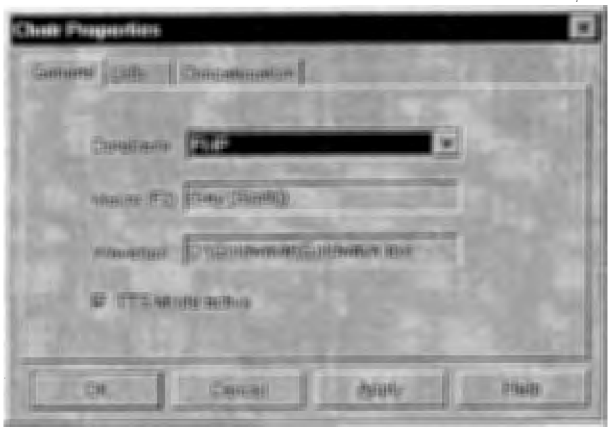
2.2 Property Sheet

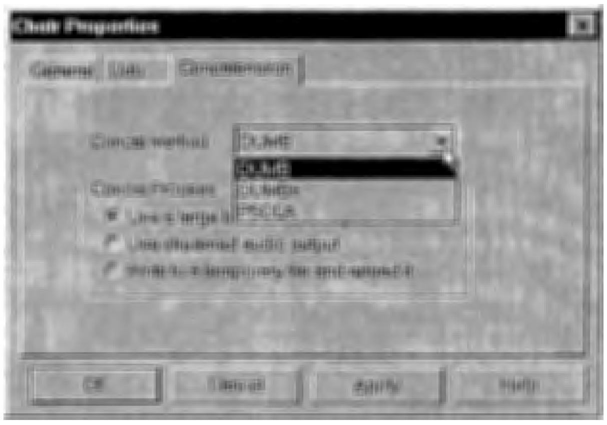
2.2.1 Introduction

In Unix Chatr, environment variables are manipulated by command lines like these:

```
(set nus_kko_params '(
        (join_wt 5.0)
        (unit_wt 1.0)
              ...
        ))

(Parameter Concat_Method DUMB+)
```

Because it is not very comfortable to type this in a single command line most Chatr users prefer to create a text file and execute it with a (Load "filename") command, this is similar to the ".ini files" of Windows 3.1 or the ".profile" files on Unix. But Windows applications (and many graphical Unix applications too) offer access to variables by graphical property sheets like this:





Concat method: DUMB
DUMB
DUMB+
PSOLA

Our Chatr application will support both styles:
the graphical interface will provide a quick overview whereas the command line interface will obtain compatibility with the Unix Version. This is possible because both methods can access the same environment variables (of course I provided a mechanism that this can't happen simultaneously).

I added a property sheet to our Chatr application, that will be accessible with the menu item Tools→Options. Because there were other tasks for me to do I only created a preliminary sheet with three pages and some control items on them. These control items are related to environment variables, so someone has to fill in all the control items needed for the access of the desired variables. How to modify a property sheet and how to change the dialogs is displayed in paragraph 2.6.1.

Our preliminary property sheet has several buttons and a tab control to access property pages, which are described in the following paragraphs, just to give you an idea how it works. Of course there may be better ways to group the variables, so this here is just an example.

2.2.2 Buttons

The following buttons are provided by the framework of our application when we create the property sheet the way I did:

Name of Button	Description	ID
OK	Leave Dialog and apply changes	ID_OK
Cancel	Leave Dialog and discard changes	ID_CANCEL
Apply	Apply changes	ID_APPLY
Help	Display help	ID_HELP

2.2.3 Property Page General

I created this page for general variables that affect the whole application like a command macro or the name of the database. The last column shows the name of the corresponding global variable, but the pages have local variables (m_...) to buffer the global ones.

Name of field	Control	Description	Variable
Database	Combo Box	Choose name of database form given list	strDatabase
Macro	Edit Field	Edit a user specific macro	strMacro
Wavetool	Edit Field	Path of a tool that can display a waveform in RIFF format	strWaveTool

2.2.4 Udb-Page

The unit selection algorithm has many environment variables, so maybe it's worth two property pages or a bigger one (you can change the size). In my opinion it's not necessary to put every variable in here, because some of them are experimental or of minor importance.

Name of field	Control	Description	Variable
beam width	Edit Box	Number of paths to continue at each stage	nBeamWidth
cand width	Edit Box	Number of new candidates to build paths through	nCandWidth
join weight	Edit Box	Overall weighting of distance measures according to continuity	dJoinWeight
unit weight	Edit Box	Overall weighting of distance measures according to match of a unit	dUnitWeight

2.2.5 Concat

The concatenation module offers several concatenation methods like Dumb, Dumb+ etc. to join the units and there are different ways to handle the output.

Name of field	Control	Description	Variable
Concat Method	Combo Box	Name of the concatenation method	strConcatMethod
Concat Process	3 Radio Buttons	Integer value representing one of three possible output processes (a large buffer in memory, streamed output, write temporary files).	nConcatProcess

2.3 Help System

Windows offers a sophisticated help system which is described in [KRU]. For Chatr, I modified the traditional Windows help system created by the Application Wizard with contents, index and search, and created a homemade command line assistant that is similar to the "TAB-TAB" key combination on Unix systems.

2.3.1 Windows Help System

A Windows help System needs three files (in "hlp" directory of our Chatr project):

- a Rich Text Format document with topics and links (Chatr.rtf)
- a hlp file with information for the compilation (Chatr.hlp)
- a cnt file with tree information for visualization (Chatr.cnt)

To edit the RTF file you will need a word processor like WinWord, the hlp file is processed by the Windows Help Workshop ("hcw.exe" in "Msdev" directory) and the cnt file has a simple ASCII format and can be edited with a standard editor or with Help Workshop. Although there is a good explanation of these three files in [KRU] I will add some information on the RTF files:

The RTF file contains pages (create new page with CTRL+Enter), headed by up to three footnotes with custom footnote marks.

Footnote Mark	Footnote Text e.g.	Description
#	HID_ASCII	Help context ID of this page (the content page has the ID HID_CONTENTS)
$	How to synthesize an ASCII Text	Topic Title
K	Synthesize: ASCII	Keyword text → in a help index this keyword will be listed and the topic title will be displayed when you click that keyword, so it is important that keywords of similar topics match.

To insert a new footnote place the cursor at the beginning of the page, choose menu item "Insert→Footnote" and choose your "Custom mark".

A link to a page is created by a text in a specific style corresponding to the type of the link, followed by a text with "hidden" style that contains the ID of a page. There are three different link styles:

Text Style	Description
Double underlined	Normal link
Single underlined	Pop-up-jump, a small pop up window will appear with the contents of the referring page
~~Strikethrough~~	

To have an example how this works look at the files by using the applications I described above.

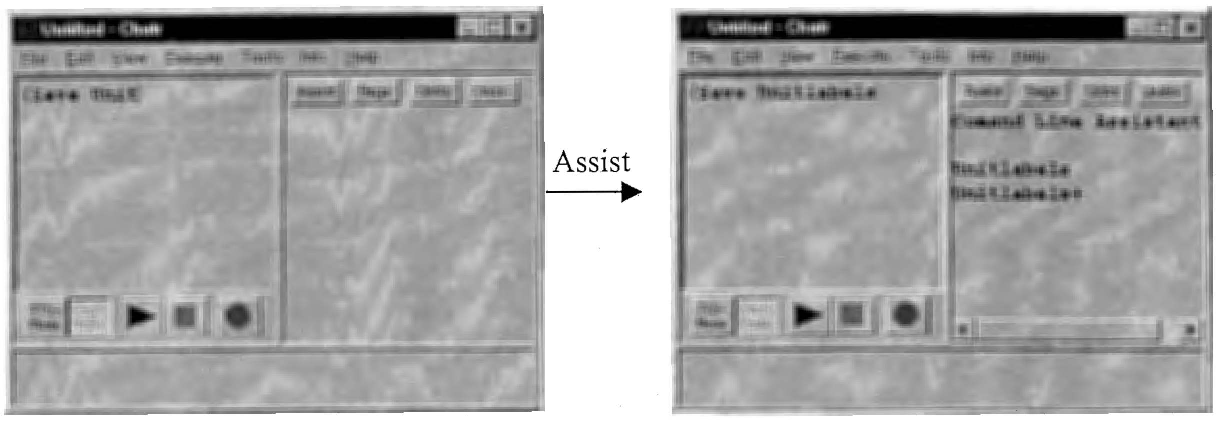
2.3.2 Command Line Assistant

The help function is powerful but not enough. Chatr has many commands, so the user often wants to know what the parser allows at a certain place of a command line without starting the online help. On Unix you have to doublepress the Tab key to accomplish a word or to get a list according to the content of the current command line. This is fast and effective once you got used to it. It would be nice, if the Windows version could even improve this mechanism, so I tried to "merry" both worlds and created the assist functionality triggered by the "assist" button in the toolbar of the info window.

In Comand Mode, pressing the Assist button will cause the following actions:

- the word in front of the cursor position will be selected
- the selection will be replaced with the string the possibilities have in common
- if there are more possible patterns than one they will be displayed in the info window
- the cursor will be positioned at the end of the word

For example the line (Save UnitLabels "...") could be typed like this:



After typing in '(Save Unit' the user presses Tab-Tab, the parser will add 'labels' and the info window will display "UnitLabels" and "UnitLabels+".

Compared with the original Unix style this method will not scroll the command line up, so in my opinion it is a bit superior.

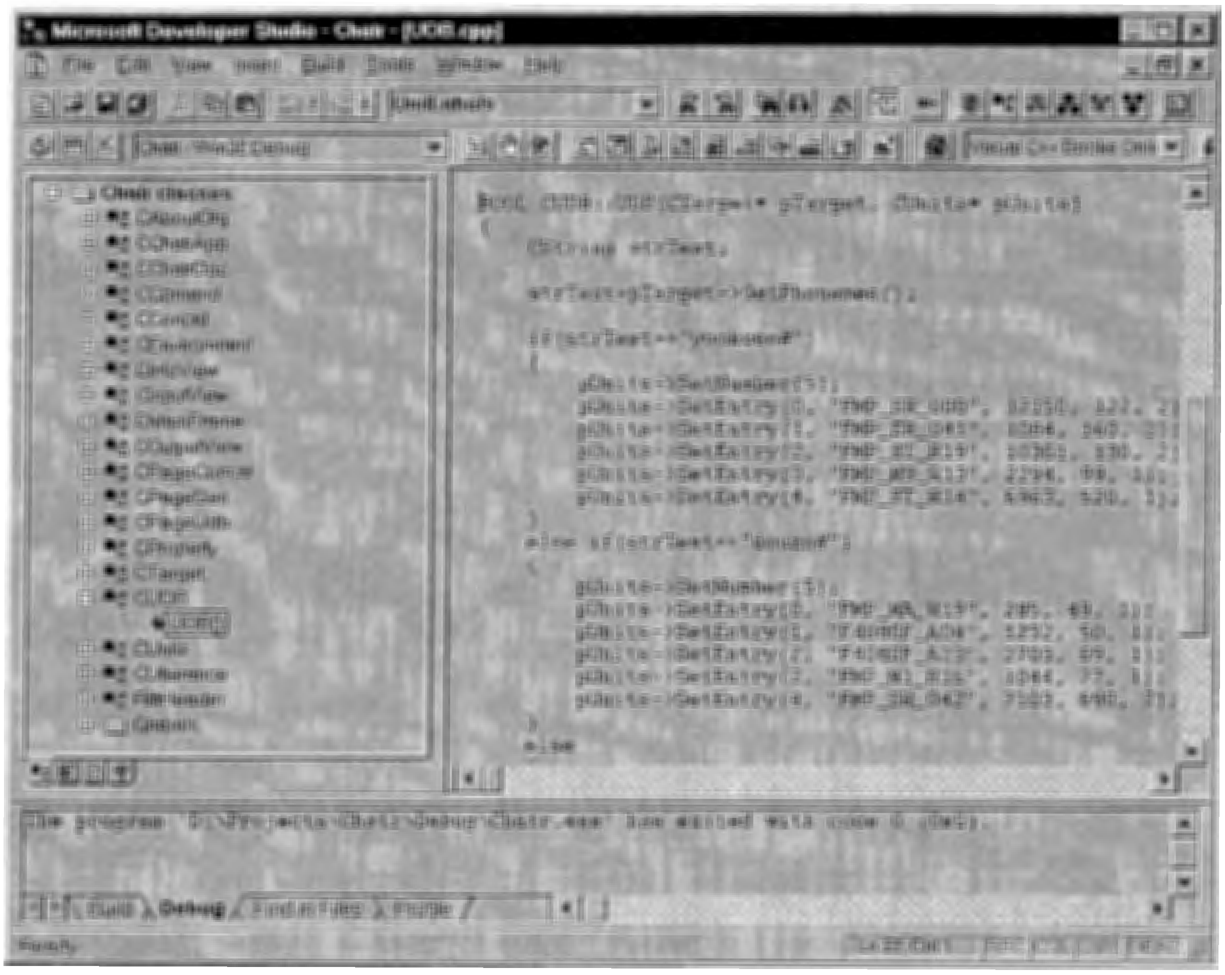
An alternative is a context sensitive popup menu activated by the right mouse button (or the "Syskey" on Win95 keyboards) that shows the possible keywords. The selected menu item in this list will replace the current selected word. This version is more "Windows like" but we will have to add a popup menu to the RichEditCtrl item of the input window (via Component Gallery). To access the entries we would have to add the menu items to the popup menu with a fixed ID (CMenu::InsertMenu()). So we would have to define IDs with a text displayed in the menu and a help-text. The messages could be handled in the RichEditCtrl item class by a ON_COMMAND_RANGE macro that maps all the IDs to one specific function. Of course this is possible but it will take some time only to create the IDs (the coding is much easier then you would expect due to the Component Gallery), so I decided on the first solution.

2.4 Technical Overview of GUI Code

Our GUI consists of several modules represented by objects of classes described in the following paragraphs, but first I want to explain to a newcomer how to read the code.

2.4.1 How to access the code

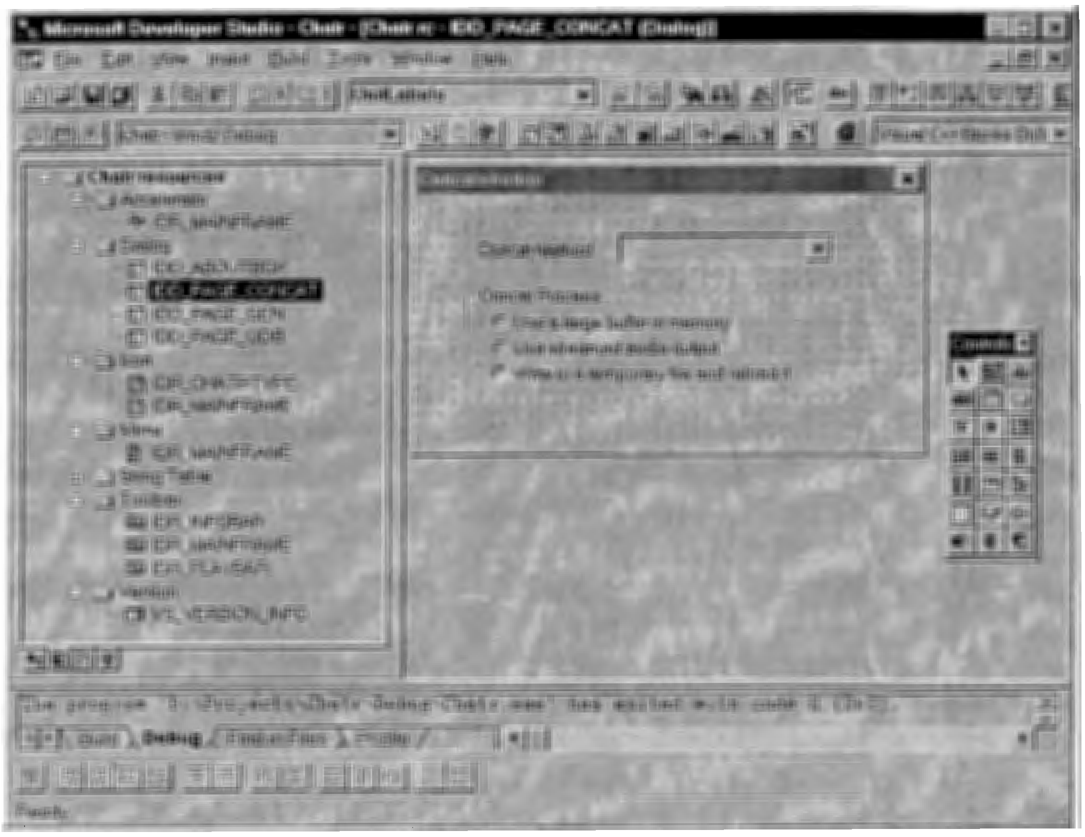
When you work with Visual C++ (here V4.0) load the workspace "Chatr". The main screen of Visual C++ has a tree control in the window on the left to display the class definitions. When you double-click a class name the edit window on the right will show the class declaration that will usually be in a header (.h) file. When you expand the tree you will have access to the code of member functions and variables that are listed in alphabetical order (normally a .cpp file).



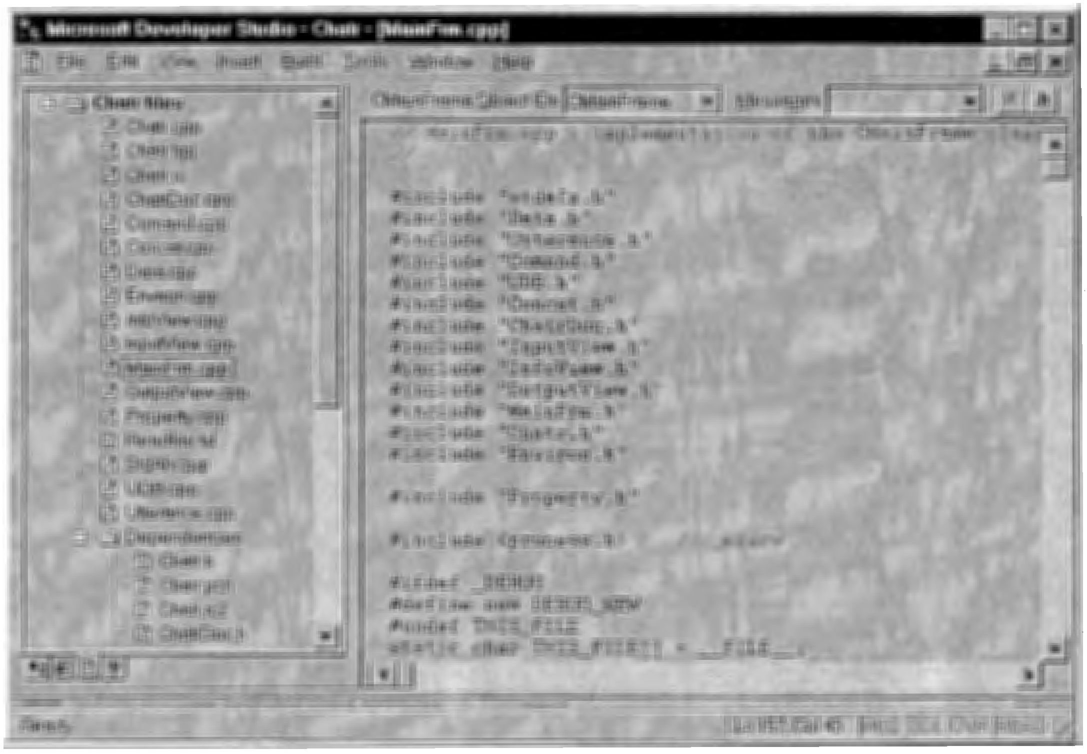
```
[illegible]
{
    CString strText;

    strText=pTarget->GetPhoneme();

    if(strText=="[illegible]")
    {
        [illegible]
    }
    else if(strText=="[illegible]")
    {
        [illegible]
    }
    else
```

You can access code independent from the file where the code is realized because Visual C++ will search for you.
The Globals section lists all global variables and functions, in our case the object "theApp" which is the object that represents our application, the object "g" that contains all environment settings and the function "Thread" that has to be global for technical reasons (see also 2.5).
The GUI consists of the classes CMainFrame, CInputView, CInfoView, COutputView, CChatrDoc, CProperty, CPageGen/Concat/Udb and ChatrApp, we have a structure called RiffHeader (used for creating and reading audio files in RIFF format) and CAboutDlg (handles the "about" dialog). The other classes are part of the Chatr core described in 3.

If you want to access resources, click on the second tab at the bottom of the left window.



Now you will see all the resources like accelerators (hotkeys), dialogs, icons, menus, string tables (the list of all ID values together with accompanied help texts and tool tips), toolbars and the version info. To use a graphical tool just double-click the resource's name.



You can have a file tree like in the picture above, but the class view is more comfortable.

2.4.2 The Frame Window

The frame window is represented by an object of class CMainFrame derived from class CFrameWnd. It handles the three child windows, all the update command handlers, and many message handlers, so it is the core of our application.

Member variables	• m_wndSplitter1/2 to create the split windows. • The Boolean m_bCreated prevents an access violation when OnSize() is called before the child windows are created.
OnCreate()	builds up the window and loads resources like the Menubar and the toolbar (both IDR_MAINFRAME).
OnCreateClient()	constructs the nested split window view and therefore creates the objects for the CView classes. It uses CSplitWnd::CreateStatic() and CreateView() to create the panes of the splitted window.
OnSize()	is called by the framework after OnCreateClient() or when the frame window has been resized, and arranges the configuration of the child windows.
UpdateHandlers	define if a toolbar or menu item is enabled, checked and so on. These handlers are valid for all the child windows (represented by the nested views), so all the items will be updated by the frame window's handlers.
CommandHandlers	Here are the handlers for all "global" messages: • copy/paste functions: to select the right RichEditControl object GetFocus() is called. • Properties: initiating the Dialog, load/save and default. • Change or restore the current window configuration • Some of the handlers for messages that can be accessed by the toolbars in the input or info window call the views of this windows to handle the message. This has something to do with the windows messaging system that doesn't send messages to the frame window under specific conditions.

2.4.3 The Views

The View classes handle the visualization of our three child windows. Our three Views are derived from the base class CView and have some code in common:

RichEditCtrl	The RichEditCtrl object m_rich provides a window within the child window that can handle text. This control is VERY powerful and can handle many different styles, fonts, colors, OLE-objects and so on. I prefer it (alternative: CEdit object or CEditView) because of its few limitations and its high versatility. The edit client is constructed by its member function Create() where all the window styles are set using predefined flags. The member SetOptions() will set the properties of our edit field that may vary due to the window's purpose.
OnCreate()	builds up the window and can load a resource like a local toolbar.
OnSize()	is called by the framework. Here it arranges the size of the RichEditCtrl object and the toolbar objects if available (not output view).

a) CInputView

The input window provides the input field and the "Playbar".

Toolbar	The OnCreate function will add a toolbar (Resource IDR_PLAYBAR) to the view that is attached to the window's bottom (CBRS_BOTTOM).
RichEditCtrl	The input window has a border (WS_BORDER) to highlight it, is editable and the selection will not be hidden if the control looses input focus (ECO_NOHIDESEL). If we double-click on a word the whole word will be selected (flag ECO_AUTOWORDSELECTION)
CommandHandlers	Here are some command handlers for the messages sent by the local toolbar at the bottom of the window. These are the handlers for the "TTS mode" and "Comand mode" buttons that will set the g.bTTS variable, the "Start" "Stop" and "Record" buttons. The handler for the Start button will start a new Thread that can be stopped with the Stop button, see also 2.5.
OnDraw	Normally this function is used to build up the display of the window, but here we use it to handle the printout.

b) CInfoView

The info window has a read only edit field and a toolbar at the top.

Toolbar	The OnCreate function will add a Toolbar (Resource IDR_INFOBAR) to the view, that is attached to the child window's top (CBRS_TOP).
RichEditCtrl	The info window's RichEdit object is read-only (ECO_READONLY) and will loose its selection if the control looses input focus.
CommandHandlers	These will handle messages triggered by the buttons of the Infobar. They will clear the info window and then display the requested information by printing strings provided by the functions for visualization of the class that contains the data (see 3.3).
OnInfoSave()	This message handler will save the content of the info window to a file using a dialog provided by the MFC class CFileDialog. According to the type of the information displayed in the window it will propose a specific extension for the filename.
Clear	Clears the window by setting the window text to empty.
Type	Displays a string on the window.

c) COutputView

The output window is the simplest of all, is used as a teletype style display and is read only without special tricks.

Toolbar	The OnCreate function will add a Toolbar (Resource IDR_PLAYBAR) to the view, that is attached to the child window's bottom.
RichEditCtrl	The input window is editable and the selection will not be hidden if the control looses input focus.
Type, Clear	Like CInfoView

2.4.4 The Document

Windows programming makes use of the so-called Document/View Architecture that uses a Document class to store data and one ore more associated View classes to display the information. This method is very powerful because like this it is easy to handle quite complex data structures.
In our case the document class CChatrDoc is very primitive (almost superfluous), because our document only consists of the ASCII text displayed in the input window. The only interesting member functions have to do with loading and saving data.

Serialize	This member function will do storage and loading using an archive object created by other functions. I tried to use the << and the >> operator, but the loading didn't work with normal ASCII files, only with saved files. So I used common functions like WriteString() and Read() that don't look very elegant but will do their job.

2.4.5 The Property Sheet

The dialog we use to access the environment variables has graphical resources and needs two types of classes, CPropertySheet and CPropertyPage. I derived some classes from these two base classes and put them all together in two files ("property.cpp" and "property.h") :

a) CProperty derived from CPropertySheet

This class is used for providing the frame for the property pages. It uses objects of the other classes as member variables, so it is important that the declaration of this class comes after the declaration of the property pages (otherwise the compiler will complain).

Constructor	The constructor is very simple and uses AddPage() to link objects of the property pages to the property page.

b) CPageGen/Udb/Concat derived from CPropertyPage

A property page is associated with a corresponding dialog resource. To build a property page, make a new dialog resource and run Class Wizard to create the class (derive from CPropertyPage) with it's member variables and the following handlers:

OnInitDialog	This handler (WM_INITDIALOG) will copy the global variables to the member variables of the associated dialog resource. If the dialog has control items that need initialization (e.g. Listboxes, Comboboxes, Spincontrols) it can be done here (you can also define the list items in the resource).
OnApply	Here the contents of the local member variables will be copied to the global environment variables.

2.5 Multiple Threads

To provide a stop key for a running process, similar to the ESC-Key or CTRL+C in Unix, I used multiple threads, synchronized by two Booleans, g.bRun and g.bStop. When using threads we have to ensure that our "worker thread" can't be started multiple times, it will finish when it has to and finally that no data processed within the thread can be accessed from outside at the same time.

To start and stop the thread we will use command handlers for the "start" and the "stop" button which are enabled according to the value of our two Booleans. The code shows that the thread can be started only once, and when the thread ends it has to set the g.bRun variable to false.

```
void CInputView::OnExecuteStart()
{
     if(g.bRun==FALSE)
     {
          g.bRun = TRUE;
          AfxBeginThread(Thread,0,THREAD_PRIORITY_HIGHEST);
     }
}

void CInputView::OnExecuteStop() { g.bStop = TRUE; }
void CMainFrame::OnUpdateStart(CCmdUI* pCmdUI) {pCmdUI->Enable(!g.bRun); }
void CMainFrame::OnUpdateStop( CCmdUI* pCmdUI) {pCmdUI->Enable( g.bRun); }
```

The thread itself looks like this:

```
UINT Thread(LPVOID lParam)
{
     g.pComand->ExecuteComand( "(SayText \"hello\")" ); // for example
     ...
     g.bRun = g.bStop = FALSE;
     return 0;
}
```

In my code I used the lParam parameter to pass a char pointer. If it is NULL, the thread will get the command string from the input window, otherwise from lParam.

The functions called by the thread should have code like this at strategic points (e.g. in loops):

```
if(g.bStop==TRUE) return FALSE;
```

To prevent the user to access data that could influence the thread at runtime we will have to modify ALL corresponding update handlers like load, save, options, info etc.

```
void CMainFrame::OnUpdateXXX(CCmdUI* pCmdUI) {
          pCmdUI->Enable( ... && !g.bRun); }
```

To make sure our thread has finished when the user closes the application we will override the handler for the WM_ON_CLOSE message:

```
void CMainFrame::OnClose()
{
     if(g.bRun){
          MessageBox("Please cancel the running process first !");
          return;
     }
     CFrameWnd::OnClose();
}
```

2.6 Customizing the GUI

2.6.1 How to modify a Property Page

To access an environment variable with the property dialog of our application five steps are necessary (also look at 2.4.5):

- declare the variable as a public member of the CEnvironment class (.h file)
- add a default initialization to the SetDefault() member function of the CEnvironment class (.cpp file)
- modify the graphical resource (dialog IDD_PAGE_XY) of the desired property page with the graphics editor (e.g. fill in an edit field and a static text)
- use class wizard to add a corresponding member variable to the related class of the graphical resource (class CPageXY)
- modify (or create) the member functions OnInitDialog() and OnApply() of the related class (same as above) and add some code to transfer data from and to the global environment data (look at the existing code how it is done).

Remarks:

- I don't recommend to link the global variables directly to a dialog resource, because many functions like the controlling of the range of a variable can't be accessed that way.
- If you add control items like Listboxes or Spincontrols you have to initialize them in the OnInitDialog() function of the related class. One of the existing pages has a Combobox in it, look at the code how the initialization is done.

2.6.2 How to modify the Menu

Modifying the menu bar is easy:

- just add your menu item to the graphical resource of the menu bar.
- you will have to enter an ID for your menu item like ID_MYMENU, the system will generate a default, but you might to change that name
- add a help text and add '\n' to the text, followed by a short name (this will be displayed in the tool tip)
- use class wizard and add a command handler and an update handler to a class (e.g. CMainFrame) by clicking at the message name (ID_...)
- write the code to your new functions (look at the other handlers how it is done)
- for a better overview move your code to a place in the source code where it fits best, e.g. move a command update handler to the place where all the others are, or if you prefer group your command handlers and update handlers together with a comment so that other people can find them easily.

2.6.3 How to modify a Toolbar

Like the Menubar the toolbar has a graphical resource that can be modified very easily. The procedure to bind a function to the button is the same as described above because menu items and Toolbuttons trigger messages the same way and messages are unambiguous defined by their ID.

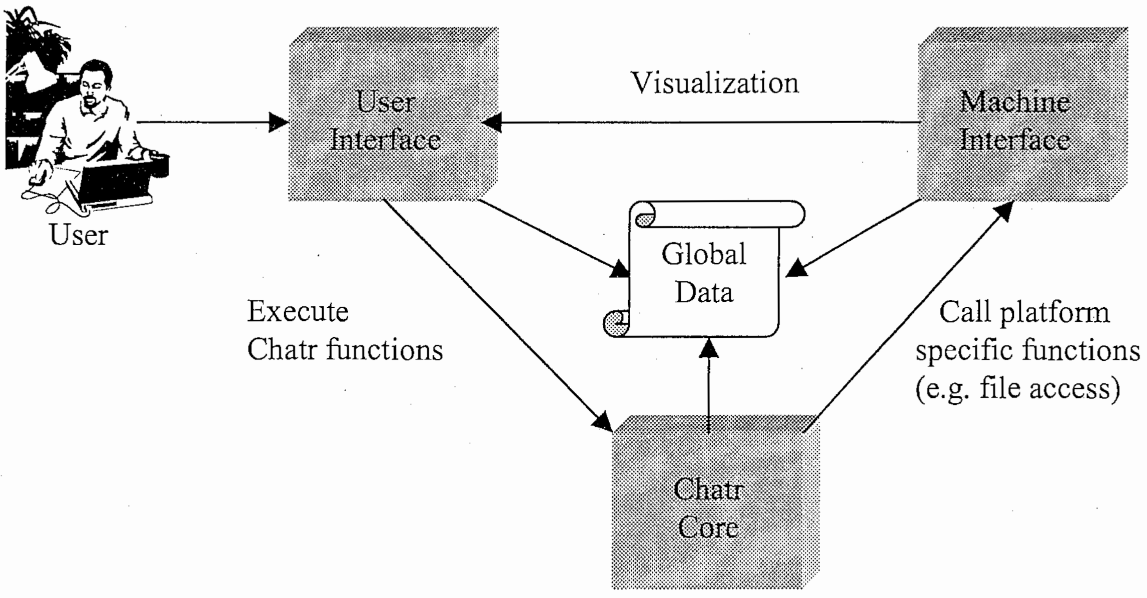
3 Modules

3.1 A Proposal for Chatr Modules

Huge programs like Chatr are much easier to read and to modify if all the functions are grouped in several modules with specified interfaces, so I tried to design a framework for our Chatr Application.

3.1.1 Overview

My proposal consists of three modules: Core, User Interface and Machine Interface, a module for global data can be separate or included in the core (it has some platform dependent environment variables in it).



The user interface provides the frame of our application. When the user wants to access a Chatr function, the user interface will call the Chatr core. Obviously there are many basic functions like file access and visualization that depend on the used platform, so the core has to call these platform specific functions that can be combined in an own module called machine interface as macros, inline functions or ordinary functions. For visualization the machine interface may call functions provided by the user interface. All modules have access to the global data that contains environment variables, databases etc. (if a module has local environment variables, these can be added to the global data for a better overview).

3.1.2 Communication

a) Man Machine Interface

When the user wants Chatr to execute a command line he communicates with the UI first. With our GUI the user has several degrees of freedom, because he can type the text in the input window, can load a text from an existing file, he can execute a predefined macro or click on a button that will perform some commands. To access data we can use the common Chatr command lines or functions provided by the GUI which can be triggered with buttons or menu items.

b) The UI calls the Chatr Core

To execute a Chatr command line the UI will pass a string (here: the whole content of the input window, just a selected area or a text created by the GUI itself) to a specific Chatr module that will check the syntax and process the string. So Chatr has to provide a module that can process that string and return some information on syntax and success of computation. The UI can call the core also for assistance in checking and completing command lines (see also 2.3.2).

c) The UI accesses Global Data

The UI can access the data modules if it wants to display some data like the current unit list. The data module(s) has/have to provide member functions for visualization. The UI will also access the global environment variables when the user activates the property dialog.

d) Chatr Core calls the Machine Interface

Each platform has its own specific functions for file access and especially audio output. For example the Microsoft Foundation Class MFC (a huge library) provides many useful functions for a Windows application, whereas a Unix Code makes use of standard devices and streams. To integrate this in our application we can work with macros or inline-functions, defined in our machine interface module. C++ offers a very effective method of overriding functions and operators. For example if you want to use something like “strText >> stdout” in the core module it will run on a Unix based system without any change. For our application we will have to override the >> operator in combination with stdout and redirect the information to the output window.

e) Chatr Core accesses Global Data

The core will access the data for processing the speech synthesis (see also 3.2.1), but also for thread synchronization and to get the environment information.

f) The Machine Interface calls the UI

There may be platform specific functions like the output on the screen that are defined in the User Interface. To access them we will need a function or macro in the machine interface that will call the corresponding GUI function if we don’t want the core to call the UI directly.

g) The Machine Interface accesses Global Data

Primary the machine interface will need environment variables.

3.1.3 Problems

The modularization of the existing Chatr code is not an easy task and will consume many working hours. But before we can do the coding we have to solve some problems to design a good specification.

a) Platform specific code

This should be our biggest problem, because Chatr has many functions that use hardware or operating system dependent code. Defining all these functions in an extra module ("machine interface") sounds very easy but I am sure it will be quite hard. Actually there will be modules with only a few relevant functions like the Comand module and some modules with lots of them (like the Concat module). So someone has to define a method that balances overview, speed and is appropriate for this application. For the most modules inline definitions should be appropriate, but for the Concat module it might be better to provide the functions for the concatenation and call them from the machine interface that handles the file access and the audio output.

b) Synchronization versus Portability

Windows applications use multiple threads all the time, even if you don't notice it, so I introduced this mechanism to Chatr. To synchronize the threads I use a very simple mechanism, but of course this is Windows specific. If you have a look at paragraph 2.5 you will see the code is very simple (if g.bStop return FALSE), but this line has to be everywhere in the Chatr core at strategic positions, even if the code will be ignored on other systems (using inline functions).

c) Little/Big Endian problem

Chatr processes many precompiled databases like audiofiles, unit lists and so on. Some of these databases contain values with more then one byte length (i.e. integer values with four bytes or 16Bit audio data with two bytes). Intel based systems like Windows or Linux use Little Endian (least significant byte first) whereas systems with Motorola or Sun architecture like the common Unix systems use Big Endian (most significant byte first).

So there are two possibilities:

- a database has to be available in Big Endian and in Little Endian format, so there should be an external converting program, or
- there is only one database, and one of the two platforms has to perform byte swap but this online converting causes slowdown in performance.

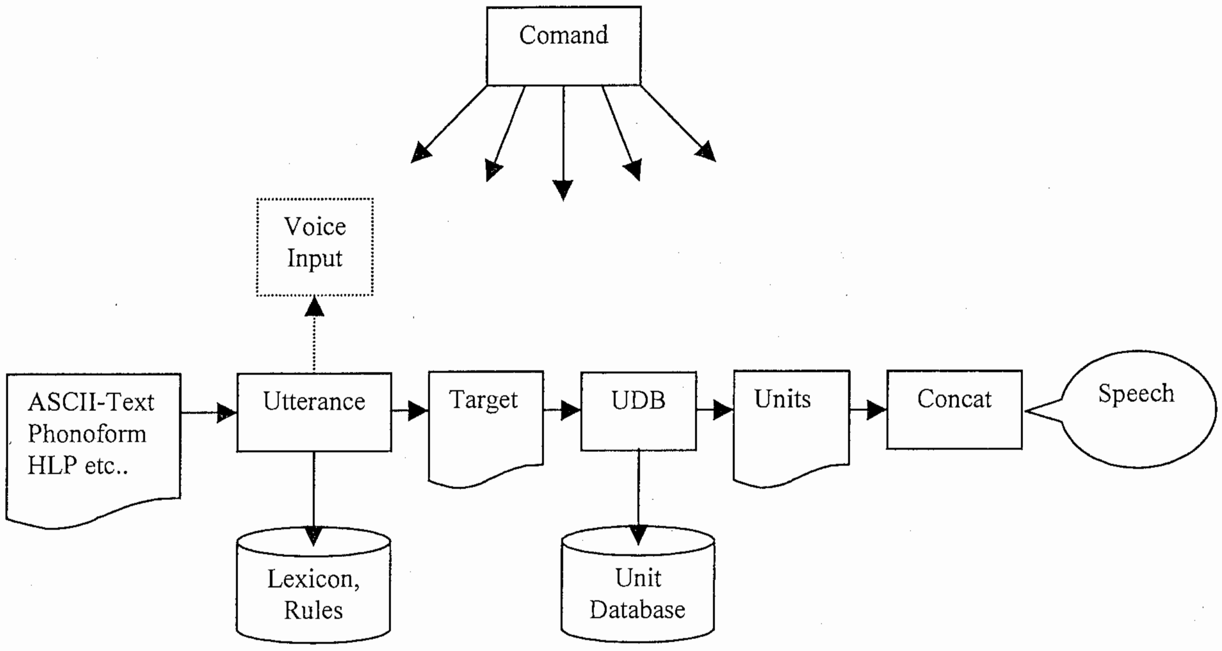
I would prefer the first solution, even if it means that we have to maintain two versions of our databases.

3.2 The Chatr Core

To separate the UI and the classic Chatr functions, I had to define modules and an interface between those modules. This chapter will show my proposal for Chatr modules that will contain functions that I filled with dummy code to test my GUI. So this is only a proposal and might be far away from the final realization, but since nobody has done a modularization of the Chatr code when I arrived at ATR I had to invent something just to run my code. The only module with real code is the Concat module that provides streamed audio output that can be used for unit concatenation in real-time.

3.2.1 Overview

Normally Chatr processes speech synthesis like this simplified flowchart:



So there is an obvious way how to divide the whole process into separate modules:

We first need a <u>Comand</u> module that will handle the command line input and will control all other modules. To handle all the different input formats for an utterance like plain Text, PhonoForm, PhonoWord or HLP, we will need a module that combines other modules like lexicon, intonation, phoneme duration or prosody. I named this module <u>Utterance</u> according to the related Chatr command, it will receive the input in any of the mentioned formats and will build a structure representing our target for synthesis. The target consists of a list with phonemes, duration, power and f0. To process this data it is a good idea to encapsulate it in a class I named <u>Target</u>. Another module is the <u>UDB</u> module that will access the unit database to select the best units according to some selection criteria. The list of units is another data field that can be encapsulated in a so-called <u>Units</u> class. Finally the listed units are concatenated using a specified concatenation method. The corresponding module can be called <u>Concat</u>.

3.2.2 Comand Module

The Comand Module is represented by the class CComand. It will handle the execution of all Chatr commandos and is the only module that the UI calls when it accesses the Chatr core.

I propose three public member functions that can be called by the UI:

ExecuteComand	BOOL ExecuteComand(CString strComand); This function will receive a pointer to a zero terminated string that contains the commands to be interpreted, e.g. (SayText "hello"). It will parse the string and will call the functions that will handle the specific commands. The return value will be true if all commandos could be processed successful, otherwise a false return will indicate an error or termination by the user (with the g.bStop variable).
TestComand	BOOL TestComand(CString strComand, CString& strErrMsg, int& nStart, int& nEnd); The method will check the zero terminated string for syntax errors and will return true if the syntax is correct, otherwise false. In the second case, the string strErrMsg will contain an error message and the integer values the position of the first and the last character of the passage that caused the error. This will enable the UI to display the error and highlight the corresponding selection in the text.
GetComand	int GetComand(CString strComand, CString& strResult); The UI will call this function if the user wants the system to complete an unfinished command line contained in the string strComand (see also paragraph 2.3.2). If there is a match with existing patterns the return value is the number of matches and the string strResult will contain the list of matches (strings divided by CR+LF "\n"). For example, the call GetComand("(Sa", strResult); will return the value "3" and strResult will be "Say\nSayText\nSave".

Then there may be many protected member functions to handle the specific Chatr commands like (Say), (Synth), (Save) and so on. If a function from outside the Comand module wants to access a Chatr command, it has to pass the command as a string to the ExecuteString function like this:

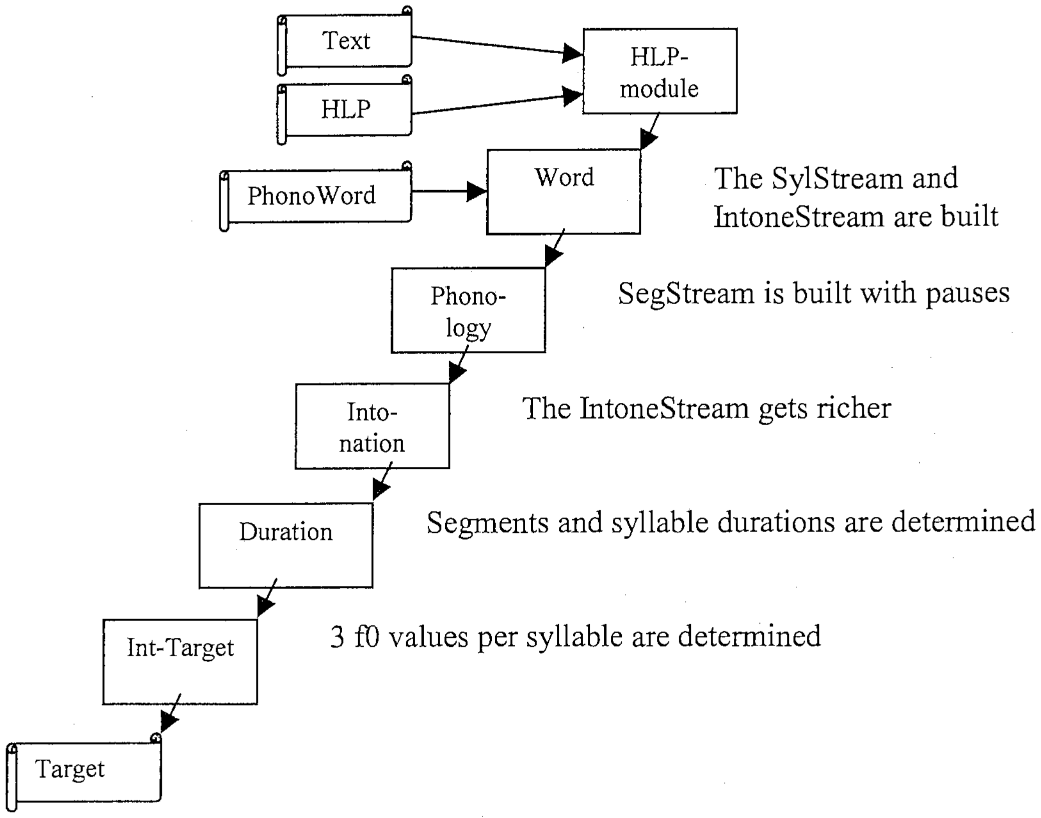
```
g.pComand->ExecuteString("(Say)");
```

This high level interface seems clumsy in the first moment, but on the other hand a programmer can access Chatr functions without knowing the name of the method and all the parameters, he (or she) only has to know the name and the syntax of the abstract Chatr command that is independent from the coding. Obviously this only makes sense if the execution is not time critical.

My Comand module is a primitive parser that handles the commandos (SayText "..."), (Save Wave "filename") and (Play "filename") just for demonstration, so it will become obsolete when someone fills in real code.

3.2.3 Utterance Module

The Utterance module will have functions to convert several input formats of utterances into a target structure with information on phonemes, prosody, power etc. Here is a more detailed flowchart of the Utterance module according to [HER]:


Text
HLP
HLP-module
PhonoWord
Word
The SylStream and IntoneStream are built
Phono-logy
SegStream is built with pauses
Into-nation
The IntoneStream gets richer
Duration
Segments and syllable durations are determined
Int-Target
3 f0 values per syllable are determined
Target

All these modules build up streams and therefore will share data, but from a more global view they will receive an utterance to create a target. In my eyes it makes sense to combine them in a huge class where all these streams are declared as protected members so that all the member functions can access them like global data. This has the advantage that we only have to change the existing C code due to the new variable names and the class scope.
According to the Chatr commandos (Utterance Text "..."), (Utterance PhonoWord (...)), (Utterance PhonoForm (...)) or (Utterance HLP (...)) I propose the following public member function of the Input class that will be accessed by the Comand module:

Utterance	BOOL Utterance(CString strInput, CTarget* pTarget, int nType); According to the type of the input specified by the nType parameter, this function will pass the input string strInput to other member functions as shown in the flowchart above. The pointer pTarget to an object of the CTarget class will enable our modules to build up the target. The function's return value will indicate success, error or termination.

3.2.4 Udb Module

The "heart" of Chatr computes the selection of units according to weighted unit costs, join costs, prosody and syllable context. This module will have many member functions due to its complexity, but extern modules like the Comand module only need one public member function to initiate the selection process:

Synth	BOOL Synth(const CTarget* pTarget, CUnits* pUnits); This function will call other protected member functions to perform unit selection. It makes sense to pass pointers to the input and the output data, because sometimes we want to process local objects. The return value is true if successful and false if not or the user terminated (g.bStop=TRUE).

The Udb class I created contains only a dummy function that will produce a list of units if the target is the word "youkoso" or "douso". That's really primitive but enough if we only want to know how the interface between modules works. Maybe we will have to add a possibility to exclude specific units to enable features like Testseg (perhaps with a global list ?).

3.2.5 Concat

The Concat module manages the unit concatenation and the platform dependent audio handling, so it is hard to decide if parts of it should be defined elsewhere in the machine interface module. Anyway, it will receive a pointer to a unit list and will create audio output or a wave file. There are several methods to concatenate units (Dumb, Dumb+, PSOLA ...) and different ways of output (write to a file, use a large buffer, streamed output ...). In my module I defined two public member functions:

Concatenate	BOOL Concatenate(const CUnits* pUnits, CString strFilename = NULL); The function will choose a concatenation method according to the global environment variables. If no filename is given it will perform audio output, otherwise it will create a wavefile with the given name.
Play	BOOL Play(CString strFilename); This function will perform audio output for the given Filename. If the file has no valid Riff format (Windows Wave file) the function will use the global wavefile parameters (raw PCM data, 16Bit, mono, 16kHz).

Here is a proposal for some of the protected member functions:

Dumb2Buffer	allocate a large buffer and create a Windows Riff file in memory.
Dumb2File	use one small buffer and write units to a Windows Riff file.
Dumb2Stream	use two small buffers to play a unit in a buffer while the other one is loading into the other buffer. This function is time critical because the unit concatenation has to be processed in real-time.
DumbP2Stream	similar to Dumb2Stream but uses Dumb+ concatenation method and therefore requires three buffers, this function does not yet exist.
DumbP2File	like DumbP2Stream but writes to a file instead.

Because the Dumb2Stream function includes some interesting code like device independent, streamed low level audio output you may want to check paragraph 6.2.3.

3.3 The Data Modules

Our Chatr application will access data, and some of this data should be global so that the UI and the different Chatr core modules can access them. Of course all the modules have their own data encapsulated within their classes, but I think there are three categories of data that should be accessible by all modules: the predicted target utterance, the selected units and the environment variables. Here is my proposal for these data classes:

3.3.1 Target

The information on the phonemes, the intonation and everything that is needed to specify a target utterance will be described in an object of the CTarget class. There will be a global object but due to the encapsulation we can create new objects that can be used like user defined variables (maybe we want to write a script that will compare different target objects). The class will contain common variables, arrays, chained lists or whatsoever to represent this data, so here is a proposal for member variables:

nEntries	Integer with the list's number of entries (= number of phonemes). Indicates if the list is empty.

According to the principals of C++ we won't access the member variables directly, but with public member functions. Some of them can be used to visualize the data, and I think it is useful to combine data and visualization in one class.

Clear	void Clear(); This function will clear all entries and will set the entry counter to zero.
GetSegment	CString GetSegment(); The function will return a string for visualization, the string may look like this:

```
(Utterance Segment
(
 ( y  60  0  ((117  0)))
 ( o  77  0  ((117  0)))
 ( o  77  0  ((117  0)))
 ( k  67  0  ((117  0)))
 ( o  74  0  ((117  0)))
 ( s  104 0  ((117  0)))
 ( o  87  0  ((117  0)))
 ( #  600 0  ((117  0)))
))
```

Because there is a global Target object we can access the visualization functions from the Comand module or from our User Interface (the functions to display data in the info window make use of this).

My colleague Martin Holzapfel defined a data format for these information but unfortunately it was too late to make it part of the code, so someone else has to pick up his ideas and integrate them into Chatr.

3.3.2 Units

To concatenate units we will need a list of them, here represented by an object of a class called CUnits. Apart from the list this class will feature some protected member variables:

nEntries	Integer with the list's number of entries (= number of units). Indicates if the list is empty.

As always we will have to define some member functions to access the member variables:

Clear	void Clear(); This function will clear all entries and will set the entry counter to zero.

GetUnits	CString GetUnits(); and CString GetUnits(pTarget);
	
	Returns a string for visualization of the Unit Stream. The overloaded version with the pTarget pointer produces a string with target and database info in it ("UnitsLabels+"). The return string may look like this:
	`(Utterance Unit`
	`(`
	`( "D:\Wave\FMP_MA_R19.wav" 285 49 1 )`
	`( "D:\Wave\F408SF_A04.wav" 1232 50 1 )`
	`( "D:\Wave\F408SF_A23.wav" 2703 69 1 )`
	`( "D:\Wave\FMP_M1_R16.wav" 1044 77 1 )`
	`( "D:\Wave\FMP_SR_042.wav" 7103 690 2 )`
	`))`

3.3.3 Environment

The CEnvironment class will contain all the environment variables that are used by our application. We will create a global object of this class called 'g', so that we can access it's member variables by "g.typeName". Here are some examples for variables, but someone will have to create a complete list. The first group of variables is relevant for the GUI:

pMainFrame etc	These pointers are related to the global objects of our classes. These objects will be created in the constructor of the CEnvironment class.
bTTS	This Boolean indicates that TTS mode (interpret content of input window as a text) is active (otherwise "Comand mode").
bRun and bStop	Two volatile Booleans used for the synchronization of the worker thread that will execute the Chatr commands (see also 2.5)

Our class will have some member functions too, e.g. for initialization.

SetDefault	void SetDefault(); Initializes all variables, parameters like the name of the database can be set. The reason why I built an explicit function instead of using the constructor is that we can call this function at any time to reset our application (e.g. by Tools→Default Options). Maybe it's better not to hardcode default values but to load them from a write protected file.

4 Conclusion

4.1 The Result of my Work

In the 10 weeks I actually worked on this application I built a GUI that can be used for demonstrations and as example for coding. I also created the streamed audio output.

I kept the interface simple so it should be easy to join Chatr and GUI together. When I specified the Chatr modules I had a limited view, and maybe I didn't think of some aspects other people know, so everybody is encouraged to make a better design.

A problem is that my application is like a commercial program with limited customization, but Chatr is an experimental system that changes and grows all the time. E.g. it should be easy to add new databases or single modules without compiling everything new, but if the GUI has to reflect some of these changes we will have to modify it. DLLs (dynamic link libraries) can improve flexibility of the code, but the problem of customizing the GUI will persist.

4.2 Future Work

In the short period of my stay I had to concentrate on the main focus, so there were many things I couldn't do because of a lack of time. Here I will list some topics:

The biggest and most important task will be the rewriting of the Chatr modules with C++. This will take quite a long time but is worth the effort, because it increases the overview and makes modifying the code easier.

The next step is to connect Chatr and the GUI, perhaps by using the interface I defined. When the combined system runs stable it will be necessary to customize the GUI, especially the property dialog.

Another project is voice input for processing a f0 contour for a given text input. The GUI has a big red button for voice recording, but in the moment the corresponding command handler is a dummy function only. To integrate the tool I propose to start an extern application the way I "spawned" the sample editor in the handler for the wave edit function (CMainFrame::OnWaveedit). This function will write the current text from the input window to a temporary file and will start the voice input tool which can even run on a different platform. This tool will write a target utterance in segment form to disk which can be loaded by Chatr.

There are many ways to optimize and complete the GUI, so here is a list of necessary, useful and interesting but not necessary tasks that can keep someone busy for several months:

- Create a complete helpfile (see 2.3.1), this is MUCH work. Maybe it is possible to reuse existing data or to convert parts of the helpfile to HTML format to improve the existing help files.
- Change the dialogs of the property pages to reflect the environment variables and additional functions.
- Code for loading and saving environment variables (CMainFrame::OnToolsSave/ Loadoptions). Maybe it is better to use a combination of Chatr commands for this because it is Unix compatible and there is existing code for this (of course we have to add GUI specific data).
- Instead of CEnvironment::SetDefault() which is hardcoded use the above mentioned Loadoptions function to load a write protected configuration file.
- Maybe it's better to use the Windows Registry for storing environment information.
- Building in new environment variables according to the Chatr core and new functions of the GUI.
- Find a method that messages from InputView or InfoView can be mapped to MainFrame independent from the activated view, so that we don't need handlers in two classes.
- Find a better way to redraw the frame window (CMainFrame::ResizeClient())
- Change the SDI interface into a MDI interface the can handle many documents simultaneously, so that the user can switch.
- Adding eye candy like a "splash screen" that is a large bitmap image that will be displayed at the start of the application (like WinWord or Developer Studio).
- Multilingual support
- Create a downgraded version of Chatr for Text-to-Speech only that can be used for demonstration (something like "Chatr Box").
- Make use of different fonts, styles and colors for the input window to highlight syntax errors and to mark intonations.

Note: Realizing the GUI as a static dialog that cannot be resized would have reduced the programming effort by approximately 10 times. The major drawback is that the static size limits the user and is inappropriate due to varying screen sizes of PCs.

According to the introduction in the first chapter here are some proposals to improve the German synthesis:

- Use a large database, if possible from one recording session to avoid discontinuities.
- If you want Chatr to sound natural choose recordings from spontaneous speech, but be careful that you choose a speaker who is not too fast.
- If you have the choice between a system that follows your target but sounds awful and a system that sounds much better but has a dynamic of its own choose the latter and emphasize join costs more than unit costs (take this advice from someone who made tests with persons which voices they prefer in a car environment). Set the weights of join costs : unit costs 5 : 1.

4.3 Comments on my Work

Although I was very happy at ATR there are some aspects that could be improved from my point of view (when I left ATR some of my proposals have already been realized).

4.3.1 Infrastructure

When you walk round the laboratories you see many computers, in my opinion too many because this is not necessary. There are many old Sun compatible Unix workstations (SPARCstation 5, 10 and 20) and there are many expensive PCs that loose their value very quickly (e.g. six months ago a 200MHz Pentium with 64MB was high tech, now we have 333MHz PentiumII with 256MB, but 400MHz CPUs are expected in two months).

To save lots of money and space and to make the life of the TSG people easier I propose the following: Everyone should have a Sun station with a good monitor. For the number crunching there should be a server like a Sun Enterprise 450 (www.sun.com) that can use up to 4 UltraSparc CPUs. This 100kg box can use PC components like the extremely cheap memory DIM modules, so it is inexpensive to stuff it with some gigabytes of memory. It might be necessary to upgrade the network due to the higher load. Usually the monitor of a server will not be used very frequently, but the brand new monitor of the server might be better then the old monitors in the lab. So we can exchange them if they are compatible. Like this the researchers get better monitors.

To run standard PC applications like WinWord, Excel and Visual C++ we definitely don't have to buy big machines for everyone. Most of the time a PC waits for keyboard input anyway, so it is obvious that many users should share one PC. To do this we will need one big PC with lots of memory and a really fast hard disk, because this is the key component of a server. The operation system will be a modified version of Windows NT, called wind (Windows Distributed Desktop) by Tektronix (www.tektronix.com). With wind it is possible to get a remote access to the PC from a Unix workstation. All the tasks will run on the PC and the Sun will provide a "Windows window", and things like data exchange or copy&paste will work. This really clever solution of a thin client is 100% compatible, very stable and reliable. We use this at my institute in Germany, because we didn't want to buy a PC for everyone, only to use the latest version of Framemaker (by the way, the Framemaker license for Windows is much cheaper than the Unix license).

I was not very satisfied with the network because it was not possible to access workstations from a PC and vice versa, so I often had to use floppy disks for data transfer. ITL uses the Microsoft Network that does allow inhomogeneous networks, but at my German institute we use the NFS network solution from FTP Software (www.ftp.com) that allows far more sophisticated networking.

4.3.2 Software and Literature

It is amazing that there is so much software available in Japanese, but it is hard to get English software or literature. For someone like me it is impossible to use a Japanese Visual C++ because I depend on the online help. I propose that there should be some machines with international versions of operating system and standard programs (maybe a Japanese and an international WinDD server ?).

5 Bibliography

[HER] Herbert Tony & Campbell Nick, Prosody within Chatr,
ATR Technical Report, TR-IT-0202 (1997-01)

[KRU] Kruglinsky David, Inside Visual C++, Microsoft Press, ISBN

[STR] Stroustrap Bjarne, C++

[BRI] Brinckmann Caren, German in Eight Weeks – A Crash Course for Chatr,
ATR Technical Report, TR-IT-0236 (1997-09)

6 Listings & Receipts

6.1 A nested Split Window

The following procedure describes the creation of a Single Document Frame with three nested Views used in this Chatr Application.

Sources:	Kruglinsky's Inside Visual C++, Example 19B Sample-Sourcecode VIEWEX on Visual C++ CD-ROM (\msdev\sample\general\viewex)

- Start Developer Studio (Visual C++ 4.0 or 5.0)
- Use AppWizard to create a new Project (New→Workspace):
  - choose Single Document
  - from [Advanced Options→ Window Styles] check [Use split window]
  - rename View → InputView
  - finish
- With Class Wizard add classes CInfoView and COutputView, derive from CView
- Modify the headers of all .cpp files:

```
#include "stdafx.h"
#include "ChatrDoc.h"
#include "InputView.h"
#include "InfoView.h"
#include "OutputView.h"
#include "Chatr.h"
```

- Add CMainFrame member variables (MainFrame.h)

```
protected:
     CSplitterWnd m_wndSplitter1;
     CSplitterWnd m_wndSplitter2;

     BOOL m_bCreated;
     float m_fX;
     float m_fY;
```

- Edit the MainFrame constructor

```
CMainFrame::CMainFrame()
{
     m_bCreated=FALSE;
     m_fX=0.5F;     // For Nested Split View
     m_fY=0.8F;     // For Nested Split View
}
```

- Edit the MainFrame code for OnCreateClient and OnSize

```
BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT /*lpcs*/,
     CCreateContext* pContext)
{
     if ((!m_wndSplitter1.CreateStatic(this, 2, 1)) ||

          // 2nd row
          (!m_wndSplitter1.CreateView(1, 0, RUNTIME_CLASS(COutputView),
                                      CSize(10, 10), pContext)) ||

          // Split 1st row
          (!m_wndSplitter2.CreateStatic(&m_wndSplitter1, 1, 2,
               WS_CHILD | WS_VISIBLE,
               m_wndSplitter1.IdFromRowCol(0, 0))) ||

          // 1st col of 1st row
          (!m_wndSplitter2.CreateView(0, 0, RUNTIME_CLASS(CInputView),
               CSize(10, 10), pContext)) ||

          // 2nd col of 1st row
          (!m_wndSplitter2.CreateView(0, 1, RUNTIME_CLASS(CInfoView),
                         CSize(10, 10), pContext)))
     {
          return FALSE;
     }

     m_bCreated=TRUE;

     return TRUE;
}

void CMainFrame::OnSize(UINT nType, int cx, int cy)
{
     if(m_bCreated)
     {
          m_wndSplitter1.SetRowInfo(   0, (int)(cy*m_fY), 20);
          m_wndSplitter2.SetColumnInfo(0, (int)(cx*m_fX), 20);
     }
     CFrameWnd::OnSize(nType, cx, cy);
}
```

- Add to all View.h files

```
public:
     CRichEditCtrl m_rich;
```

- Use Class Wizard to add messages WM_CREATE, WM_SIZE to all CView-Classes and add code:

```
int C?View::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
     CRect rect(0,0,0,0);

     if (CView::OnCreate(lpCreateStruct) == -1)
          return -1;

     // Construction of the Edit Client
     m_rich.Create(ES_AUTOVSCROLL | ES_MULTILINE | ES_WANTRETURN |
                   ES_AUTOHSCROLL | WS_HSCROLL |
                   WS_CHILD | WS_VISIBLE | WS_VSCROLL, rect, this, 1);

     //Set options for edit control
```

```
    m_rich.SetOptions(ECOOP_SET,           // Set options
                   ECO_NOHIDESEL |         // Sel.not hidden
                   ECO_AUTOHSCROLL |       // Scroll option
                   ECO_AUTOVSCROLL |       // Scroll option
                   ECO_AUTOWORDSELECTION | // Word sel on double click
                   ECO_WANTRETURN |        // -> Prevent Default Button
                   /* ECO_READONLY | */    // Readonly-Flag
                   ECO_SAVESEL);      // Sel saved if focus lost
    return 0;
}


void C?View::OnSize(UINT nType, int cx, int cy)
{
    CRect rect;

    CView::OnSize(nType, cx, cy);
    GetClientRect(rect);
    m_rich.SetWindowPos(&wndTop,0,0,rect.right-rect.left,
                        rect.bottom-rect.top, SWP_SHOWWINDOW);
}
```

- to get the pointers of the Views use:

```
    m_wndSplitter?.GetPane(y,x);
```

for example:

```
    g.pInputView  = m_wndSplitter2.GetPane(0,0);
    g.pOutputView = m_wndSplitter1.GetPane(1,0);
    g.pInfoView   = m_wndSplitter2.GetPane(0,1);
```

- to obtain Copy/Paste functionality, start Class Wizard, modify CMainFrame Class and add Commands for ID_EDIT_COPY, ID_EDIT_PASTE and ID_EDIT_CUT:

```
void CMainFrame::OnEditCopy()
{
    ((CRichEditCtrl*)GetFocus())->Copy();
}

void CMainFrame::OnEditCut()
{
    ((CRichEditCtrl*)GetFocus())->Cut();
}

void CMainFrame::OnEditPaste()
{
    ((CRichEditCtrl*)GetFocus())->Paste();
}
```

If you want to do fancy things like fonts, toolbars etc. have a look at the MainFrame class of our Chatr applicaion.

Please notice: Windows messages will be sent to the RichEditControl Objects and NOT to the corresponding views, and GetFocus() will return a pointer to the RichEditCtrl object with the input focus on !!
So if you want to do contextsensitive popup help etc. you have to derive your modified class from CRichEditCtrl with class wizard and add a comand handler (e.g. for right button down).

6.2 Audio Output

In my opinion sample code is more important than reference books, so here are excerpts from the code of a little program (Project "Audio") I wrote just to play with audio. To include this code in your proggy use the Component Gallery to add multimedia support to or #include <mmsystem.h> and link "Winmm.lib" (Build→Settings→Link→Library modules). The end of this chapter contains a function I wrote for the Chatr application. I did some comment on this because I was told that some people might want to use this routine. Please note that I didn't add error handling procedures, so you perhaps might to add some.

Windows 95/NT offer an significant improvement of the audio handling to the previous Windows 3.1x. Because audiobuffers don't need no locking anymore, it is possible to modifiy contents of a buffer while the buffer is asynchronously played.

6.2.1 Simple Highlevel Output

The highlevel output uses the PlaySound() function described in the detailed MFC online help. For the following listings only the parameters SND_ASYNC, SND_SYNC, SND_MEMORY and SND_NOSTOP are important.

Topics: How to play a file, play from memory, play two files from memory, concatenate wavefiles and play the buffer, modify a buffer during play.

a) Play from file

Will play a Wavefile from disk asynchronously. Nice to mock up the boring About dialog.

```
void CMyApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    PlaySound("C:\\Windows\\Media\\Sound.wav", 0, SND_ASYNC);
    aboutDlg.DoModal();
}
```

b) Play from memory

Will load a file into memory and will play it afterwards

```
void CMainFrame::OnPlayPlaysoundfromfile()
{
    //PlaySound("C:\\Windows\\Media\\Sound.wav", 0, SND_SYNC);

    char* pBuffer;
    int dSize;
    unsigned long dRead;
    CFile cfile;
    CFileStatus status;

    // Open File
    if(!cfile.Open("C:\\Windows\\Media\\The Microsoft Sound.wav",
CFile::modeRead))
        return;

    // Get Length
    cfile.GetStatus( status );
    dSize = status.m_size;

    // Allocate Buffer
     pBuffer = (char*)malloc(dSize);

    // Load File
    dRead = cfile.Read(pBuffer, dSize);

    // Close Handle
    cfile.Close();

    // Play Sound from Memory
    PlaySound(pBuffer, 0, SND_MEMORY | SND_SYNC);

    // Free Memory
    free(pBuffer);
}
```

c) Play two files from memory

This proggy shows how to synchronize asynchronous audio output with high level functions. If you use this in Windows 3.1x you will get a deadlock, so you will have to add a Sleep(0) comand to the while() loop.

```
void CMainFrame::OnPlayPlaysoundtwobuffers()
{
    char* pBuffer[2];
    int dSize;
    unsigned long dRead;
    CFile cfile;
    CFileStatus status;
    int i;

    for(i=0;i<2;i++)
    {
        // Open File
        cfile.Open("C:\\Windows\\Media\\The Microsoft Sound.wav",
CFile::modeRead);

        // Get Length
        cfile.GetStatus( status );
        dSize = status.m_size;

        // Allocate Buffer
        pBuffer[i] = (char*)malloc(dSize);

        // Load File
        dRead = cfile.Read(pBuffer[i], dSize);

        // Close Handle
        cfile.Close();

        // Play Sound from Memory
        while(!PlaySound(pBuffer[i], 0, SND_MEMORY | SND_ASYNC |
             SND_NOSTOP));
    }

    // Free Memory
    MessageBox("OK","OK",MB_OK);
    for(i=0;i<2;i++) free(pBuffer[i]);

}
```

d) Concatenate wavefiles to build a large buffer

This function will concatenate parts of a file into a large buffer and will play this buffer at the end. The CreateWaveHeader() function is used to provide a selfmade header for the PlaySound() MFC function.

```
void CMainFrame::OnPlayConcatenatebuffer()
{
	char* pPlayBuffer;
	char* pPlayPosition;
	char* pLoadBuffer;
	int dPlaySize, dLoadSize;
	unsigned long dRead;
	HANDLE hfileHandle;
	int i;

	int dStart[3]={23400,45600,67800};		// Start in byte
	int dDur[3]={10000,10000,10000};		// Duration in byte

	////////////////////////////////////////////////////////
	// Allocate PlayBuffer
	////////////////////////////////////////////////////////

	// Get Length of PlayBuffer
	dPlaySize = 44; // Length of Header !!
	for(i=0;i<3;i++)
	{
		dPlaySize+=dDur[i];
	}

	// Allocate Buffer
	pPlayBuffer = (char*)malloc(dPlaySize);
	TRACE("Allocated %d Byte PlayBuffer\n",dPlaySize);

	////////////////////////////////////////////////////////
	// Allocate LoadBuffer
	////////////////////////////////////////////////////////

	// Get Length of LoadBuffer
	dLoadSize = 0;
	for(i=0;i<3;i++)
	{
		if((44+dStart[i]+dDur[i]) > dLoadSize)
		{
			dLoadSize=44+dStart[i]+dDur[i];
		}
	}

	// Allocate Buffer
	pLoadBuffer = (char*)malloc(dLoadSize);
	TRACE("Allocated %d Byte LoadBuffer\n", dLoadSize);

	////////////////////////////////////////////////////////
	// Concatenate
	////////////////////////////////////////////////////////

	pPlayPosition=pPlayBuffer+44;
```

```
    for(i=0;i<3;i++)
    {
         // Open File
         hfileHandle = CreateFile("D:\\Wave\\kko1.wav",
                                        GENERIC_READ,0,0,OPEN_EXISTING,
                                       FILE_ATTRIBUTE_NORMAL,0);

         dLoadSize=44+dStart[i]+dDur[i];

         // Load File
         ReadFile(hfileHandle, pLoadBuffer, dLoadSize, &dRead, 0);
         CopyMemory(pPlayPosition, pLoadBuffer+44+dStart[i],dDur[i]);
         pPlayPosition+=dDur[i];

         // Close Handle
         CloseHandle(hfileHandle);
    }

    /*hfileHandle = CreateFile("D:\\Wave\\kkol.wav",
                                   GENERIC_READ,0,0,OPEN_EXISTING,
                                  FILE_ATTRIBUTE_NORMAL,0);
    ReadFile(hfileHandle, pLoadBuffer, dLoadSize, &dRead, 0);
    //CopyMemory(pPlayPosition, pLoadBuffer+44,dPlaySize-44);
    CopyMemory(pPlayBuffer, pLoadBuffer,dPlaySize);
    CloseHandle(hfileHandle);*/

    // Create Wave-Header
    CreateWaveHeader(pPlayBuffer, dPlaySize-44);

    // Play Sound from Memory
    PlaySound(pPlayBuffer, 0, SND_MEMORY | SND_SYNC);

    ///////////////////////////////////////////////////////
    // Write File to Disk
    ///////////////////////////////////////////////////////

    hfileHandle = CreateFile("D:\\Wave\\TRACE.wav",
                                   GENERIC_WRITE,0,0,CREATE_ALWAYS,
                                  FILE_ATTRIBUTE_NORMAL,0);
    WriteFile(hfileHandle, pPlayBuffer, dPlaySize, &dRead, 0);
    CloseHandle(hfileHandle);

    // Free Memory
    free(pPlayBuffer);
    free(pLoadBuffer);


}

void CreateWaveHeader(char* pBuffer, int dSize)
{
    WaveHeader* pHeader = (WaveHeader*)pBuffer;

    strcpy(pHeader->riff, "RIFF");
    pHeader->l1      = 34 + dSize;
    strcpy(pHeader->wave, "WAVEfmt ");
    pHeader->l2    = 16;
    pHeader->tag   = WAVE_FORMAT_PCM;
    pHeader->channels=1;
    pHeader->rate = 16000;
    pHeader->flow = 32000;
    pHeader->block = 2;
    pHeader->bits = 16;
    strcpy(pHeader->data, "data");
    pHeader->l3    = dSize;
}
```

e) Modify a buffer during play

```
void CMainFrame::OnPlayPlayandmodify()
{
     short int* pPlayBuffer;
     int dPlaySize;
     int dRead;
     int i;

     dPlaySize=160000;

     pPlayBuffer = (short int*)malloc(dPlaySize+44);
     CreateWaveHeader((char*)pPlayBuffer, dPlaySize);

     PlaySound((char*)pPlayBuffer, 0, SND_MEMORY | SND_ASYNC);

     for(i=0;i< dPlaySize/2;i++)
     {
          *(pPlayBuffer+22+i)=(short int)(10000.0*sin((double)i/10));
     }

     /*hfileHandle = CreateFile("D:\\Wave\\TRACE.wav",
                                GENERIC_WRITE,0,0,CREATE_ALWAYS,
                               FILE_ATTRIBUTE_NORMAL,0);
     WriteFile(hfileHandle, pPlayBuffer, dPlaySize, &dRead, 0);
     CloseHandle(hfileHandle);*/

     MessageBox("OK","OK",MB_OK);
     free(pPlayBuffer);

}
```

6.2.2 Low Level Audio

Now it becomes interesting:

a) Create a tone with low level audio

```
void CMainFrame::OnPlayGeneratelowlevelaudio()
{
    UINT         wResult;
     short int*     pBuffer;
     int            i;
     HWAVEOUT    WaveOut;
     #define LENGTH 128000

     // Initialize buffer
     pBuffer=(short int*)malloc(LENGTH); // 4 seconds of audio
     for(i=0; i<(LENGTH/2);i++)
     {
          pBuffer[i]=(short int)(10000.0*sin(i/10));
     }

     WAVEHDR WaveHeader= {(char*)pBuffer,LENGTH,0,0,0,0,0,0};
     WAVEFORMATEX Format= { WAVE_FORMAT_PCM,1,16000,32000,2,16,0};

     if(waveOutOpen((LPHWAVEOUT)&WaveOut, WAVE_MAPPER,
          (LPWAVEFORMATEX) &Format,
          0L, 0L, CALLBACK_NULL))
     {
          MessageBox("Kein Open","Fehler",MB_OK);
          return;
     }
     if(waveOutPrepareHeader(WaveOut, &WaveHeader, sizeof(WAVEHDR)))
     {
          MessageBox("Kein Prepare","Fehler", MB_OK);
          return;
     }

     if(waveOutWrite(WaveOut, &WaveHeader, sizeof(WAVEHDR)))
     {
          MessageBox("Kein Write","Fehler", MB_OK);
          return;
     }


     // Use Polling method for synchronisation
     while(!(WaveHeader.dwFlags & WHDR_DONE)) Sleep(0);
     //MessageBox("OK","OK",MB_OK);

     waveOutUnprepareHeader(WaveOut, &WaveHeader, sizeof(WAVEHDR));
     waveOutClose(WaveOut);

     free(pBuffer);
}
```

b) Play a large file by using small buffers and a streamed audio output

Remark: 44 is the length of the header of a Windows RIFF wave file (*.WAV).

```
void CMainFrame::OnPlayPlaylargefile()
{
     char*          pBuffer[2];
     WAVEHDR        WaveHdr[2];
     HWAVEOUT    WaveOut;

     WaveHeader* pHeader;
     int dRead;
     int dSize=10000;
     CFile cfile;
     int i;
     int dLength;

     // Allocate Buffers
     for(i=0;i<2;i++) pBuffer[i]=(char*)malloc(dSize);

     // Open File
     if(!cfile.Open("C:\\Windows\\Media\\The Microsoft Sound.wav",
                         CFile::modeRead))
     {
          MessageBox("No CFile.Open","Error",MB_OK);
          return;
     }

     // Read Header
     dRead = (int)cfile.Read(pBuffer[0], 44);
     pHeader = (WaveHeader*)pBuffer[0];

     WAVEFORMATEX Format= {
          pHeader->tag,
          pHeader->channels,
          pHeader->rate,
          pHeader->flow,
          pHeader->block,
          pHeader->bits,
          0
     };
```

```
	dLength=pHeader->13;

	if(waveOutOpen((LPHWAVEOUT)&WaveOut, WAVE_MAPPER,
		(LPWAVEFORMATEX) &Format,
		0L, 0L, CALLBACK_NULL))
	{
		MessageBox("No WaveOutOpen","Error", MB_OK);
	}
	else
	{
		i=0;

		do
		{
			// Load next block
			dRead = (int)cfile.Read(pBuffer[i%2], min(dSize,dLength-
i*dSize));
			//TRACE("%d %d %d\n",i,dRead,dLength-i*dSize);
			// Prepare WaveHeader
			WaveHdr[i%2].lpData=(char*)pBuffer[i%2];
			WaveHdr[i%2].dwBufferLength=dRead;
			WaveHdr[i%2].dwBytesRecorded=0;
			WaveHdr[i%2].dwUser=0;
			WaveHdr[i%2].dwFlags=0;
			WaveHdr[i%2].dwLoops=0;
			WaveHdr[i%2].lpNext=0;
			WaveHdr[i%2].reserved=0;

			waveOutPrepareHeader(WaveOut, &WaveHdr[i%2],
sizeof(WAVEHDR));
			waveOutWrite(WaveOut, &WaveHdr[i%2], sizeof(WAVEHDR));

			if(i>0)
			{
				// Wait for previous block to be completed
				while(!(WaveHdr[(i-1)%2].dwFlags & WHDR_DONE)) Sleep(0);

				// Unprepare previous header
				waveOutUnprepareHeader(WaveOut, &WaveHdr[(i-1)%2],
sizeof(WAVEHDR));
			}

			i++;
		}
		while((dRead>=dSize) && ((dLength-i*dSize)>0));

		// Wait for last block and clean up
		while(!(WaveHdr[(i-1)%2].dwFlags & WHDR_DONE)) Sleep(0);
		waveOutUnprepareHeader(WaveOut, &WaveHdr[(i-1)%2],
sizeof(WAVEHDR));

		waveOutClose(WaveOut);
	}
	// Close Handle
	cfile.Close();

	// Free Memory
	TRACE("Completed %d blocks\n", i);
	//MessageBox("OK,","OK",MB_OK);

	for(i=0;i<2;i++) free(pBuffer[i]);

}
```

6.2.3 Streamed Low Level Audio Output for Unit Concatenation

The function Dumb2Stream concatenates audiofiles using two small buffers to load the files alternately. The filenames and the concatenation points are specified in a class named CUnitLabels, a pointer to an object of this class is the function's parameter.

At first the function will do the usual initialization. Then it allocates two buffers, the sizes depend on the largest unit in our list. After the audio device is prepared we will have to load a unit and then play it. When the next unit has been loaded we will wait for the previous unit (if there is one) to be finished, therefore our code polls the dwFlags entry of the waveheader. When all units have been played we will close the audio device and free the memory.

To modify this function for more advanced concatenation techniques like Dumb+ or PSOLA you have to use three buffers, so that every unit has a predecessor and a successor. Modifying the code should be easy, just use pPlayBuffer[(i-1)%3] and pPlayBuffer[(i+1)%3] for previous and next buffer and do the signal processing before the block is written to file. When the length of the unit may get bigger due to the processing, allocate e.g. 20% more memory.

```
int CConcat::Dumb2Stream(CUnits* m_pUnits)
{
     // Playbuffer
     char*          pPlayBuffer[2];
     WAVEHDR        WaveHdr[2];
     HWAVEOUT       WaveOut;
     int            nPlaySize;

     // Filestuff
     int            nRead;
     CFile          cfile;
     CString        strFilename;

     int i;
     int nNumber;             // Number of UnitLabel-Entries
     int nStart;
     int nDuration;
     char cUnits;             // just Dummy
     char pName[20];

     int returnvalue=0; // return value

     nNumber = m_pUnits->GetNumber();

     // Allocate smallest PlayBuffers for all units

     nPlaySize = 0;
     for(i=0; i<nNumber; i++)
     {
          m_pUnits->GetEntry(i, pName, nStart, nDuration, cUnits);
          nPlaySize=max((2*nDuration*SRATE), nPlaySize);
     }
     for(i=0;i<2;i++) pPlayBuffer[i]=(char*)malloc(nPlaySize);

     WAVEFORMATEX Format= {WAVE_FORMAT_PCM,1,16000,32000,2,16,0};
```

```
	if(waveOutOpen((LPHWAVEOUT)&WaveOut, WAVE_MAPPER,
		(LPWAVEFORMATEX) &Format,
		0L, 0L, CALLBACK_NULL))
	{
		returnvalue=-1;
	}
	else
	{
		i=0;
		while(i<nNumber)
		{
			// Load block i
			m_pUnits->GetEntry(i, pName, nStart, nDuration, cUnits);

			// Read file
			strFilename=DATABASEPATH;
			strFilename+=pName;
			strFilename+=".wav";

			cfile.Open(strFilename,CFile::modeRead);
			cfile.Seek(2*nStart*SRATE, CFile::begin );
			nRead = cfile.Read(pPlayBuffer[i%2], 2*nDuration*SRATE );
			cfile.Close();

			// DO SIGNAL PROCESSING HERE

			// Prepare WaveHeader
			WaveHdr[i%2].lpData=(char*)pPlayBuffer[i%2];
			WaveHdr[i%2].dwBufferLength=nRead;
			WaveHdr[i%2].dwBytesRecorded=0;
			WaveHdr[i%2].dwUser=0;
			WaveHdr[i%2].dwFlags=0;
			WaveHdr[i%2].dwLoops=0;
			WaveHdr[i%2].lpNext=0;
			WaveHdr[i%2].reserved=0;

			waveOutPrepareHeader(WaveOut, &WaveHdr[i%2],
sizeof(WAVEHDR));

			// Play current block
			waveOutWrite(WaveOut, &WaveHdr[i%2], sizeof(WAVEHDR));

			if(i>0)
			{
				// Wait for previous block and unprepare previous header
				while(!(WaveHdr[(i-1)%2].dwFlags & WHDR_DONE)) Sleep(0);
				waveOutUnprepareHeader(WaveOut, &WaveHdr[(i-1)%2],
sizeof(WAVEHDR));
			}

			i++;
		}

		// Wait for last block and clean up
		while(!(WaveHdr[(i-1)%2].dwFlags & WHDR_DONE)) Sleep(0);
		waveOutUnprepareHeader(WaveOut, &WaveHdr[(i-1)%2],
sizeof(WAVEHDR));

		waveOutClose(WaveOut);
	}

	for(i=0;i<2;i++) free(pPlayBuffer[i]); // Free Memory

	return returnvalue;
}
```