HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and Communications Engineering
Laboratory of Acoustics and Audio Signal Processing

Matti Airas

Development of a Mobile Interactive Musical Service

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, December 1, 2002

Supervisor: Professor Matti Karjalainen
Instructor: Dr. Tero Tolonen

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Matti Airas
Työn nimi:	Vuorovaikutteisen musiikillisen mobiilipalvelun toteutus
Päivämäärä:	1.12.2002 **Sivuja:** 67
Osasto:	Sähkö- ja tietoliikennetekniikka
Professuuri:	S-89
Työn valvoja:	Professori Matti Karjalainen
Työn ohjaajat:	TkT Tero Tolonen

Tässä työssä esitellään vuorovaikutteinen soittoäänipalvelu, jonka avulla palvelun käyttäjä voi laulamalla tai viheltämällä luoda omaan matkapuhelimeensa ainutkertaisen soittoäänen. Työssä käsitellään sekä järjestelmän äänenkäsittelykomponenttien että palveluarkkitehtuurin toteuttamista.

Palvelua käytetään soittamalla matkapuhelimella palvelunumeroon ja laulamalla lyhyt äänite. Äänite lähetetään soittoäänimuunnoksen tekevälle komponentille, joka automaattista nuotinnusta käyttäen muuntaa äänitteen soittoääneksi. Luotu soittoääni lähetetään edelleen lyhytsanomaviestinä käyttäjän matkapuhelimeen.

Työssä tarkastellaan eri menetelmiä perustaajuuden sekä melodian tunnistukseen ja esitellään valitut menetelmät sekä niihin työssä tehdyt parannukset.

Työssä kiinnitetään huomiota myös järjestelmäarkkitehtuuriin ja sen eri toteutusvaihtoehtoihin. Järjestelmän suorituskykyä tutkitaan sekä simuloimalla että testaamalla.

Lopuksi esitetään suuntaviivoja jatkokehitykseen tulevien mobiilistandardien valossa sekä käydään läpi päätelaitteessa toimivien musiikkisovellusten tekoon soveltuvia järjestelmiä.

Työn tuloksena syntyi toimiva soittoäänipalvelu. Palvelulla luodut soittoäänet ovat tunnistettavia ja laadultaan kohtuullisia, joskaan eivät virheettömiä.

Avainsanat: mobiilipalvelu, automaattinen musiikin nuotinnus, soittoääni, äänen perustaajuuden määrittäminen

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Matti Airas

Name of the thesis:	Development of a mobile interactive musical service
Date:	Dec 1, 2002 **Number of pages:** 67
Department:	Electrical and Communications Engineering
Professorship:	S-89
Supervisor:	Professor Matti Karjalainen
Instructors:	D.Sc. (Tech.) Tero Tolonen

This thesis presents an interactive ringing tone service, with which the user of the service can create a unique ringing tone to his mobile phone by singing or whistling. The implementation of both the sound signal processing components and the service architecture are studied.

The ringing tone service is used by calling a service number by a mobile phone and singing a short recording. The recording is sent to a ringing tone conversion component, which converts the recording to a ringing tone by using automatic music transcription. The created ringing tone is further sent to the mobile phone of the user by a short message service.

The work investigates different methods for fundamental frequency and melody detection and presents the chosen methods and improvements made to them.

Furthermore, system architecture aspects and different implementation possibilities are studied. The computational efficiency of the system is studied both by simulating and by testing.

Finally some possible paths for future developments in the light of upcoming mobile standards are presented and systems suitable for client-based music applications are reviewed.

As a result of the work a functional ringing ton service was created. The ringing tones created using the service are recognizable and of reasonable quality, although not perfect.

Keywords: mobile service, automatic music transcription, ringing tone, fundamental frequency detection

Preface

The work for this thesis has been carried out at Elmorex Ltd. and in the Laboratory of Acoustics and Audio Signal Processing at Helsinki University of Technology.

The work has been instructed by Dr.Sc.(Tech.) Tero Tolonen, with whom I worked closely in the initial stages of the project. I want to thank him both for his valuable insights on the design of the system, and for his helpful comments and suggestions on the structure and contents of this thesis. I am also grateful to my supervisor Professor Matti Karjalainen for providing insight on writing the thesis and setting target dates for the project. Without those deadlines brewing of this thesis probably would have gone on forever.

I wish to thank Jyrki Kohonen of Elmorex Ltd. for employing me for this project and supporting the writing of this thesis. Elmorex proved a unique working place for this fun and entertaining project!

Perttu Hämäläinen, a friend and a co-worker at Elmorex, has had an enormous impact on his thesis. We spent innumerable hours discussing subjects both on and off-topic, but those hours were something gained, not wasted. I always marvelled his enthusiasm on just about anything and his ability of getting things done. Thank you, Perttu!

There are also many other people whom I need to thank for either helping me out with this thesis, or for helping me keep my (relative) sanity. Listed in no particular order: my comrade-in-arms Henri Penttinen, Hanna Järveläinen, Riitta Väänänen and Ville Pulkki of Naistenhuone, Rami Laiho, Laura Turkki and others at Ihana.tv, Antti Kaihola, Laura Saarilahti, my sisters Katri and Laura and their families (especially Markku Huhta-Koivisto, who persuaded me to apply to HUT), and many, many others, whom I have forgotten to mention. Thank you, all of you! I also wish to thank Trurl and Klapausius for company and not chewing all my clothes.

Finally, I would like to thank my parents Kaija and Kalervo for their love and their support throughout my studies and showing keen interest on this thesis. My mother has shown also great skill in patching my chewed clothes, for which I am grateful.

Helsinki, December 1, 2002

Matti Airas

Table of Contents

Abbreviations

ACELP	Algebraic Code Excited Linear Prediction
AIX	IBM's UNIX operating system
AMD	Advanced Micro Devices, Inc.
API	Application Program Interface
ARM	A provider of embedded RISC microprocessors
CASA	Computational Auditory Scene Analysis
CDMA	Code-Division Multiple Access
CEPT	European Conference of Postal and Telecommunications Administrations
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
DTMF	Dual Tone Multi-Frequency
EFR	Enhanced Full Rate
EMS	Enhanced Messaging Service
EPOC	An operating system designed for mobile computing devices
FFT	Fast Fourier Transform
FIFO	First in, first out
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HFC	High Frequency Content
HP	Hewlett-Packard
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
ISDN	Integrated Services Digital Network
IVR	Interactive Voice Response
JIT	Just-in-time (compiler)
JSP	Java Server Pages
LP	Linear Prediction
LPC	Linear Prediction Coefficients
MiB	Mebibyte, 2^{20} or 1,048,576 bytes
MIDI	Musical Instrument Digital Interface
MIDP	Mobile Information Device Profile

MIME	Multipurpose Internet Mail Extensions
MMS	Multimedia Messaging System
MOS	Mean Opinion Score
MPEG	Moving Picture Experts Group
.NET	A software platform developed by Microsoft Corp.
OS	Operating System
PC	Personal Computer
PCM	Pulse Code Modulation
PRI	Primary-Rate Interface
PS	Processor-sharing
PSTN	Public Switched Telephone Network
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RPE	Regular Pulse Excitation
RTC	Ringing Tone Conversion
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UML	Universal Modeling Language
UMTS	Universal Mobile Telecommunications System
UNIX	A popular multi-user, multitasking operating system
VSELP	Vector Sum Excited Linear Prediction
WAP	Wireless Application Protocol
WLAN	Wireless Local Area Network
WSDL	Web Services Definition Language
WWW	World-wide web
XML	Extensible Markup Language

Symbols

A	Companding rate in A-law quantization
a	Traffic intensity
B_c	Call blocking probability
$c_x(k)$	Cepstrum function
$E(\nu)$	Energy error function for peak classification
f_0	Fundamental frequency
$\phi_k(m)$	Short-time autocorrelation function
F_D	Discrete Fourier transform
h	Average holding time
λ	Peak acceptance threshold
λ	Service request arrival rate (traffic theory)
μ	Companding rate in μ-law quantization
μ	Service rate (traffic theory)
ν	Autocorrelation lag value
σ	Voicedness threshold
$v(k)$	Voicedness value of a frame
$w(n)$	Time-domain window function

List of Figures

Chapter 1

Introduction

The growth of mobile short messaging services has been huge in the last years. Even though first Finnish commercial SMS content services were launched in 1998, the market value in Finland in 2001 was already 44 million euro [Min01, Min02]. The largest single service group was ringing tone services, amounting to 24% of all service requests in 2001. It is predicted, however, that the value of ringing tone services rises only slightly in the future, as the market will already saturate [Min02].

Currently all ringing tone services on the market are very similar to each other. Pre-made ringing tones can be selected from a list and ordered to user's own or other defined mobile phone number. The services differentiate each other only by branding or by the quality of pre-made ringing tones.

Given the market situation outlined in the previous paragraphs, it is obvious that new ringing tone services, which clearly differentiate the service provider from the competition, are sought for. So, when Dr. Tero Tolonen in 1999 had an idea of making an interactive ringing tone service, the idea was aptly researched, and a project to make such a service was begun.

The service implemented in this work—dubbed *Rring*—is a ringing tone service, in which the user can sing, whistle or hum himself a new mobile phone ringing tone. Present technologies are utilized in a way that a novel service is created, and so the technologies are extended to new markets.

To use the service, the user dials a telephone number. He or she hears a instruction recording, in which he is asked to sing or hum after a beep. After recording the performance, the service transforms the recorded melody to a symbolic, or object-based notation, which is then transformed to a vendor-specific ringing tone format and sent to the user.

The service is implemented as an interactive voice response system, but so that it is as easy to use as leaving an answering-machine message would be. Since the service only utilizes user interface elements already known to the user, the barrier to use is kept low.

The title of this thesis is *Development of an mobile interactive musical service*. The title can be broken down as follows. *Development* means that the work covers the algorithm selection and implementation, as well as the service framework design and

implementation. *Mobile* refers to the fact that the service is characteristically a mobile phone application. *Interactive* means that the user is not just a passive consumer of the service, but actually actively participates in the process. *Musical* stands for the musical properties of the service. The user sings, hums or whistles to interact with the service. *Service* means that the main effort in the thesis has been to create a service, which customers can use with ease. The barrier of use should be kept low and usability and attractiveness should be important design aspects.

The structure of this thesis is as follows.

Chapter 2 gives an overview of the service and describes the different high-level components. It is recommended to read this chapter first to familiarize oneself with the overall structure of the service.

Chapter 3 reviews different algorithms and background information related to the subject of this thesis.

Chapter 4 describes the particular algorithms chosen for this work and their implementation specifics. The emphasis is in the digital signal processing part of the service.

Chapter 5 describes the architecture of the service and several implementation issues such as processing power requirements, traffic analysis and scalability of the service.

Chapter 6 reviews some upcoming technologies relevant to mobile music entertainment services. These include further development ideas and an operating system review to chart viable platforms for client-side music entertainment services.

Chapter 7 concludes the thesis, analyzing the achievements of the work and summarizing the results.

Chapter 2

Overview of the system

This chapter gives an overview of the service and familiarizes the reader with the terms and concepts connected to it.

As mentioned in Chapter 1, Rring is a ringing tone service, in which the user creates himself a new, individual ringing tone. The following section gives some usage scenarios for the service. Then, in the light of these scenarios, the structure of the service is discussed.

2.1 Use cases

Usage of the Rring service is illustrated using terminology and concepts coined by Universal Modeling Language (UML) [Obj02]. Basic use is described using two use scenarios. The respective UML Use case diagram is shown in Figure 2.1. The stick figure represents the user, who is the actor in these cases. The user may either use the service to create a ringing tone, or forward a created ringing tone to someone else.

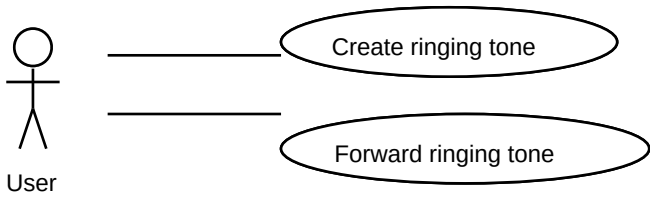


Figure 2.1: Rring use case diagram showing the common use cases.

The first scenario consists of a basic interaction between a user and the system.

> *A high-school student, having listened to her friend's new personalized ringing tone, decides to make one to her own mobile phone. She dials the advertised service number, and listens to the welcome and instruction recording. After a tone she whistles the theme of her all-time favorite piece, Europe's The Final Countdown. After whistling, she hangs up as instructed.*

> *The service then converts the melody to a ringing tone and sends it as an SMS message to the phone number the user dialed from. Almost immediately her mobile phone beeps and tells a new ringing tone has arrived. She listens to the ringing tone, which resembles her whistling, including all the errors she made while whistling. She finds it incredibly amusing and sets it as the ringing tone of her phone.*

The first scenario describes the service actually implemented as a result of this thesis.

The second scenario is a sequel to the first one. In it, the user forwards a ringing tone to her friend, thus giving added value to the created ringing tone.

> *The generated ringing tone, while having an unmistakable home-baked flavor, still is recognizable as the original piece, and the user wants to share it with her boyfriend. As the ringing tone-arrived, it was assigned a unique name. She sends a forwarding request to an SMS service. The service then re-sends the ringing tone to the number she gave.*

While planned as a future addition, the service outlined in the second scenario was not implemented in this thesis.

2.2 Structure of the service

The use sequence of Rring service is shown in Figure 2.2. As can be seen in the Figure, the service consists of three separate functional modules. The first module, the Interactive Voice Response (IVR) system, automatically handles the telephone conversation, answering the call, reading prompts and recording user input. The second module, ringing tone conversion (RTC), receives a recording from the IVR system and converts it to a ringing tone. The third module is the SMS gateway, through which the ringing tones are sent to the user.

Figure 2.3 shows a more detailed view of the ringing tone conversion module. After receiving a recording, the RTC server sends it to the pitch detector. The pitch detector then creates frame-based pitch and voicedness trajectories from the recording. This information is then sent to the event creator, which creates musical event data from the trajectories. This event data is then sent to a music transcription program, which creates MIDI and ringing tone files from the event data. The RTC server then sends the ringing tone file to the SMS gateway.

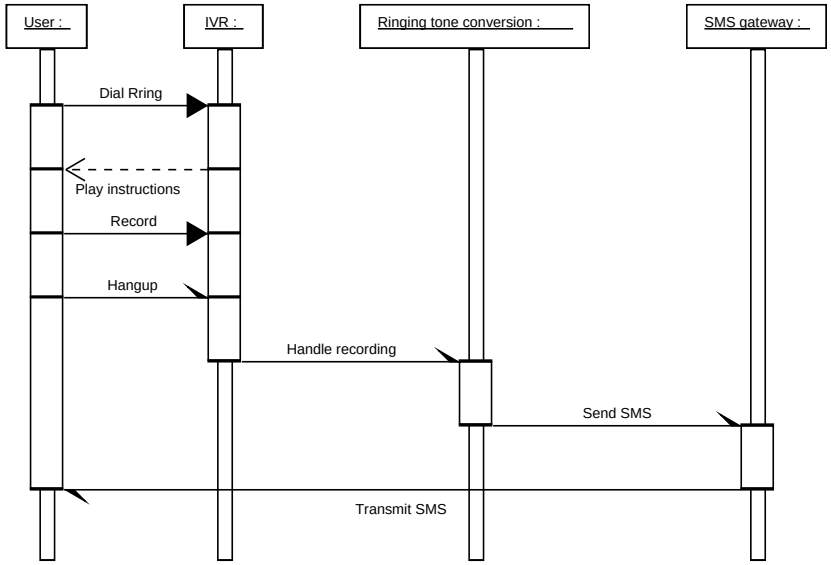


Figure 2.2: Rring use sequence. Different modules can be seen as vertical bars, with time advancing down on the vertical axis. Interaction between the modules can be seen as the different arrows.

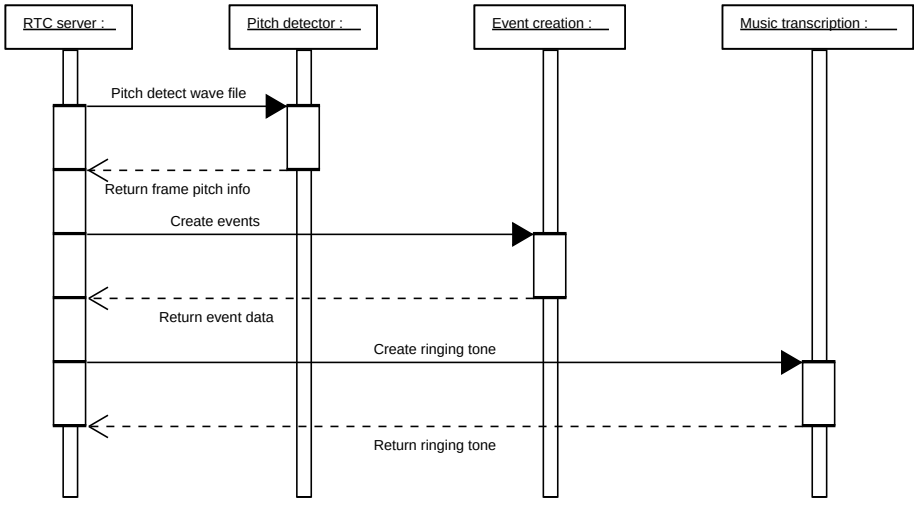


Figure 2.3: Ringing tone conversion use sequence closeup. Time advances downwards on the vertical axis, and different arrows represent interaction between the different modules.

Chapter 3

Background concepts and theories

This Chapter first gives an introduction to human speech production, after which a primer to relevant voice telecommunication technologies is given. Some different fundamental frequency detection methods are reviewed, and finally event creation and music transcription technologies are discussed.

3.1 Speech production

According to Rossing [Ros90], human speech production system is as follows. In speaking, air is forced from the lungs through the larynx into the three main cavities of the vocal tract: the pharynx and the nasal and oral cavities. From the nasal and oral cavities, the air flows through the nose and mouth, respectively. In order to produce speech sounds, the flow of air is interrupted by the vocal cords or by constrictions in the vocal tract made with the tongue or lips. The sounds from the interrupted flow are appropriately modified by various cavities in the vocal tract and are eventually radiated as speech from the mouth and the nose. The human vocal organs and a representation of their main acoustical features are shown in Figure 3.1.

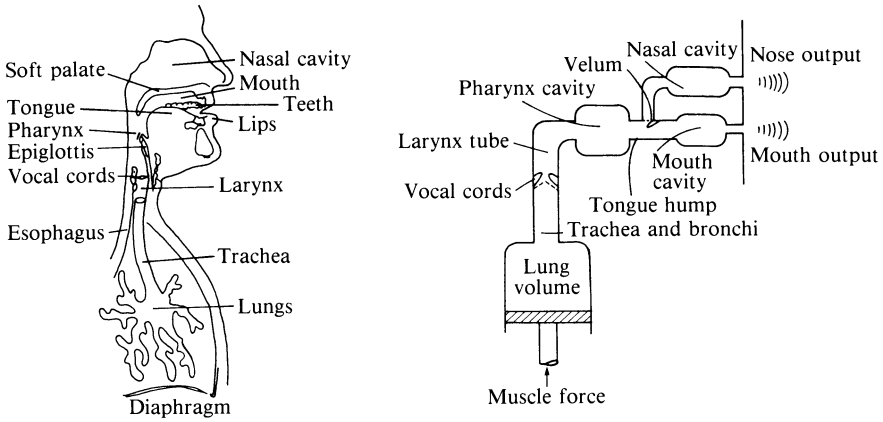


Figure 3.1: Human vocal organs and a representation of their main acoustical features. (After J. L. Flanagan [Fla65].)

Voiced sounds are produced by vocal cords. They modulate the air flow by rapidly

opening and closing, and this vibration produces a buzzing sound from which voiced sounds are created. These *glottal pulses* generated by the vocal cords define the fundamental frequency of speech. During normal speech, the vibration rate may vary over one actave, although the range of a singer's voice is more than two octaves. Typical frequencies used in speech are 110 Hz in the male, 220 Hz in the female and 300 Hz in the child, with wide individual variations. The pitch of a male singer can be as low as 80 Hz.

Different voiced sounds are produced by the vocal tract. The vocal tract, as shown in Figure 3.1, can be considered a single tube extending from the vocal cords to the lips, with a side branch leading to the nasal cavity. It transforms the sounds produced by the vocal cords into distinct vowels and voiced consonants. This is accomplished by changing the shape of the vocal tract to produce various acoustic resonances. These resonances, or peaks in the sound spectra of vowels, exist independently of pitch, and are called *formants*. They appear as envelopes that modify the amplitudes of the various harmonics of the source sound.

The relative positions of the formants differ for different vowels. Generally speaking, different vowels can be distinguished from each other if the two first formats of speech are transmitted. However, for sound quality and recognition of individual speakers, transmission of more than two formants is preferred.

Peterson and Barney [PB52] measured the formant frequencies of vowels that were perceived to be equivalent. Their results are shown in Figure 3.2, which shows second formant frequency as a function of first formant frequency for several vowels spoken by men and children. The broad ellipses show the approximate range of variation in formant frequencies for each of these vowels.

3.2 Voice telecommunication technologies

Modern public switch telephone network (PSTN) is a direct descendant of first telephone networks of 1880's. While the transmission technologies have fundamentally changed from original dedicated twisted pairs to carrier-based technologies to digital PCM transmission using broadband coaxial cables and optical transmission, the customer interface still remains essentially the same. The PSTN telephone connection still is a copper twisted pair with same electrical specifications as decades earlier. Therefore also the transmission band and dynamic range are "set in stone". This Section intends not to give a complete review of history of telephony development, but to summarize modern voice telecommunications technologies relevant to this thesis.

3.2.1 Public switched telephone network

Figure 3.3 shows the frequency response of a typical telephone band integrated codec/filter. Also shown are the transition band boundaries. The telephone transmission band is defined to be 300–3400 Hz. The necessary bandwidth was defined to (barely) fit the first four formant frequencies for vowels in male speech. This improves the legibility of speech and assists in personalizing the sound. The lower limit of the bandwidth was

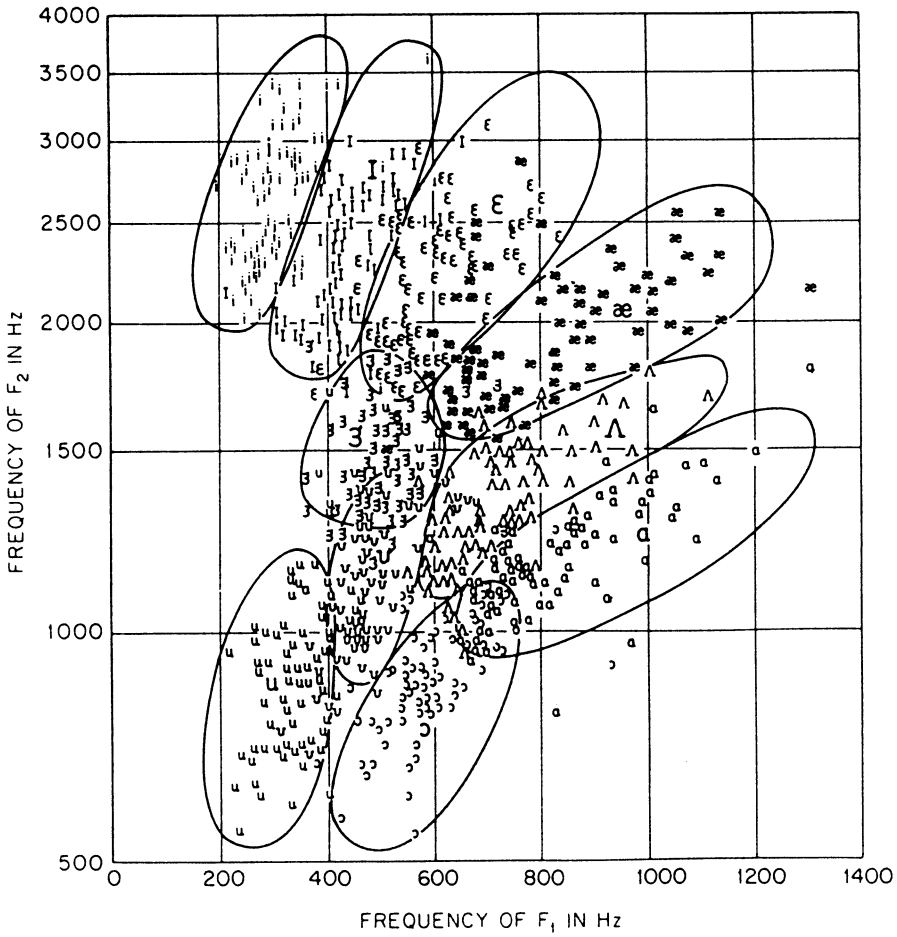


Figure 3.2: Plot of second formant frequency versus first formant frequency for vowels by a wide range of speakers. (After Peterson and Barney [PB52].)

set to provide an ample transition band for a high-pass filter. The high-pass filter was deemed necessary to eliminate any possibility of the 50 Hz (in North-America 60 Hz) mains hum coupling to the telephone signal. The higher limit was set so that the signal can be sampled at a sampling rate of 8 kHz. The 600 Hz transition band was needed so that analog anti-aliasing filters could be implemented economically. Figure 3.3 shows that especially the lower transition band slope is allowed to be quite gentle, with only 23 dB of attenuation at 60 Hz. It also has to be noted that while the lower limit of telephone band is 300 Hz, the maximum attenuation at 200 Hz is still less than 2 dB.[1]

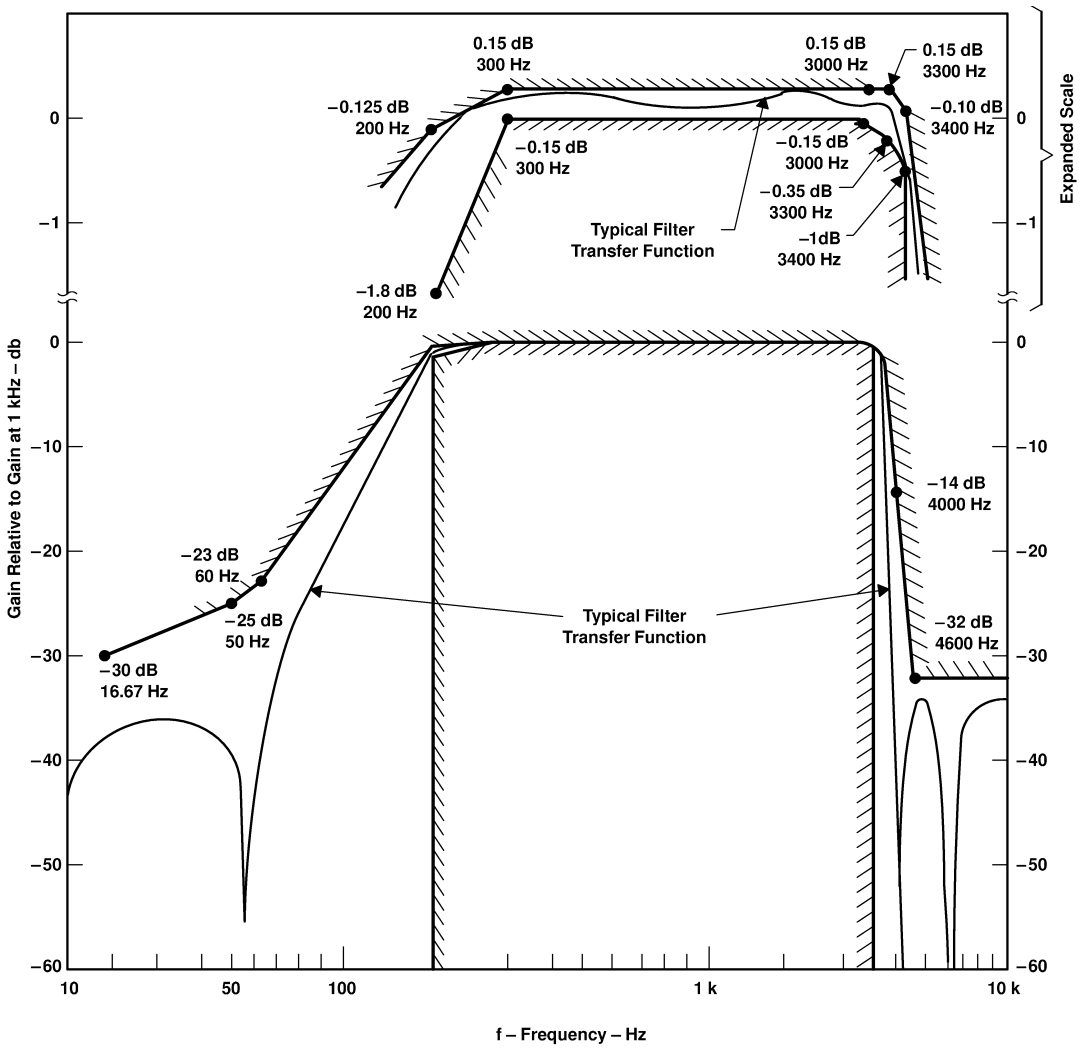


Figure 3.3: Telephone band limits and transfer characteristics of a Texas Instruments TCM29C13 single-chip PCM codec and filter. (After [Tex86].)

The signal is sampled at 8 bits per sample, so that the voice channel becomes represented by a stream of pulses with a repetition rate of 64 kHz. Uniform quantization of the signal, while the most straightforward and obvious choice, is not statistically justifiable. The probability distributions of message signal amplitudes are rarely, if ever, uniform. In speech, the probability of occurrence of a small amplitude is much greater

[1]There may be additional filtering performed in other parts of the telephone network, though.

than a large one. Experiments have shown that the probability distribution of speech signal amplitudes is approximately exponential [Dav52]. Furthermore, the range of amplitudes that can occur within a transmission system is considerable. Telephony, in particular, demands that quantizing equipment be designed to accept a wide range of amplitudes due to the wide attenuation differences of transmission lines within a switched telephone network [Owe82]. (The difference in level between two telephonic-speech signals can easily exceed 30 dB.) Consequently, it is appropriate to allocate many quantization steps in the small amplitude region, and only a few in the large amplitude region. This technique is referred to as *nonuniform quantization*.

The nonuniform quantizations in use are μ-law and A-law. They are used in Americas and in Eurasia, Africa and Australia/Pacific respectively. The companding algorithm of μ-law is

$$y = \frac{\log(1+\mu x)}{\log(1+\mu)}, \tag{3.1}$$

where μ is the companding rate (normally $\mu = 100$). The respective algorithm for A-law is

$$y = \frac{Ax}{1+\log A} \qquad \text{for } 0 \le x \le \frac{1}{A} \tag{3.2}$$

$$y = \frac{1+\log(Ax)}{1+\log A} \qquad \text{for } \frac{1}{A} \le x \le 1\ , \tag{3.3}$$

where A is the companding rate (normally $A = 87.6$).

Figure 3.4 represents signal-to-quantizing-noise ratio of an A-law companded signal. It can be seen that the ratio is 38 dB for a wide range of signal amplitudes, and begins to fall a lot later than with uniform quantization.

3.2.2 GSM

The abbreviation GSM originates from French words *Groupe Spécial Mobile*, which was the name of CEPT (*Conférence Européenne des Administrations des Postes et des Télécommunications*, or *European Conference of Postal and Telecommunications Administrations*) workgroup founded in 1982 [HAHN93]. It was assigned to define a next generation digital mobile phone system. To ensure creation of a pan-European mobile phone system, a memorandum of understanding to implement GSM in most European countries was signed in 1987. The first GSM commercial network was opened in 1991. Nowadays it is the leading second-generation mobile phone standard in the world.

GSM works on several frequency bands. In addition to the original 900 MHz band, a 1800 MHz band is allocated as well to reduce bandwidth congestion. In North America, GSM phones transmit at 1900 MHz. In the frequency domain, the allocated bands are divided to several 200 kHz bands. In the time domain, these bands are further divided to eight different time slots. The transmission rate in the channel is 22.8 kbit/s, which is divided so that 13 kbit/s is the speech codec bit rate and the rest is allocated for error correction.

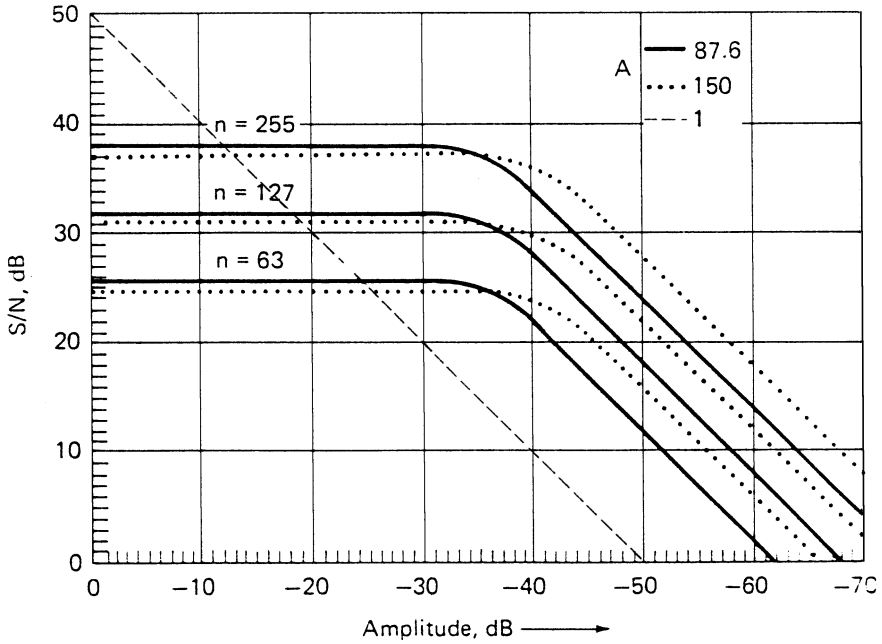


Figure 3.4: Signal-to-quantizing-noise ratio, plotted against signal amplitude for continuous, nonuniform (A-law) quantization. The dashed line represents uniform quantization with $n = 255$. (After [Owe82].)

The speech coding method used in GSM is RPE-LTP (Regular Pulse Excitation with Long Term Prediction). It works by first creating eight linear prediction coefficients (LPC) from a signal frame [Alk99]. These short term LP coefficients are then suitably represented and quantized.

The LP coefficients are calculated so that the original signal frame can be filtered using those coefficients. This leaves a residual signal, which can be efficiently quantized and coded. However, for harmonic signals an impulse structure is left to the residual. These periodic impulses limit the quanizing, and so it is removed using a one-pole long-term linear predictor.[2]

The residual left by the long-term LPC processing is then compressed by down-sampling it in 3:1 ratio. Four different down-sampled subframes are created, and the one with most energy is selected. This frame is then scaled and quantized for transmission.

The RPE-LTP decoder is lot simpler than the encoder. It basically just up-samples the residual and then inverse filters it with the long- and short-time LP coefficients, thus recreating a signal close to the original.

In the late 90's, two alternate codecs have become available in GSM phones. First of these, ACELP (Algebraic Code Excited Linear Prediction) is a so-called EFR-codec (Enhanced Full Rate), which operates at the same bit-rate as the RPE-LTP codec, but with a better sound quality. The second new codec is VSELP (Vector Sum Excited Linear Prediction), which is a half-rate coder, that compresses the speech band twice as

[2]Since the long-term linear predictor matches the impulse string caused by the periodicity of the signal, it effectively detects the fundamental frequency of the signal.

effectively as the RPE-LTP-codec. This enables theoretical doubling of GSM transmission capacity. The half-rate codec is primarily used in large metropolies of South-East Asia, while the EFR codec has been more widely adopted. All these speech coding methods are based on LPC-analysis.

Codec speech quality is commonly measured using mean opinion scores (MOS) [itu96]. The MOS quality metric scale is represented in Table 3.1. The RPE-LTP codec has a MOS value of 3.5, while the EFR codec reaches 4.0. The half-rate codec also has a MOS value of 3.5. For comparison, the MOS value of the A-law companded PSTN telephone transmission path is 4.5 [Sch01].

Score	MOS
5	excellent
4	good
3	fair
2	poor
1	bad

Table 3.1: Mean Opinion Score scale.

3.3 Fundamental frequency detection methods for monophonic signals

Fundamental frequency detection is an algorithm with a myriad of applications. Musical applications, which this presentation mainly handles, include for example automatic notation, pitch-shifting (so-called auto-tuning) and speech coding applications.

Fundamental frequency is physically the lowest frequency in the harmonic structure of sound. The perceived pitch is a complex phenomenon which includes much more than just detection of fundamental frequencies. For example, the human ear can follow several pitch trajectories simultaneously, and detect slight but expressive pitch deviations such as vibrato and microtonal intervals. The ear can be led into hearing non-existing pitches such as fundamental frequencies implied by the presence of their harmonic series. Other examples of cheating the ear include illusory pitch trajectories such as sounds that appear to be continuously ascending or descending [Ros90].

While not accurate from the psychoacoustic viewpoint, in signal processing applications fundamental frequency and pitch are often considered to be synonymous.

The majority of research in fundamental frequency detection methods is focused around few distinct applications, pitch detection of speech being most prominent of them. Using speech oriented pitch estimators in music applications is not without problems, though. Speech pitch estimators are generally accurate on a relatively narrow frequency band, about 100–600 Hz. In musical applications the pitch varies over a wide range of frequencies. For example, it can be assumed the input may be sung or whistled. Pitch of a whistled melody may vary from 700 to 2800 Hz [Ros90], while the pitch of a male singer can be as low as 80 Hz. So, a musical pitch detector should be able to handle a pitch range of 80–2800 Hz, a range of over 5 octaves.

The pitch detection methods can be divided into three categories depending on the domain they operate in: time-, frequency- and cepstrum-domain [RCRM76]. Examples of time-domain algorithms include waveform-based methods and autocorrelation methods. Frequency and cepstrum domain methods usually make use of the discrete Fourier transform (DFT).

Some pitch detection algorithms are reviewed below shortly. Waveform-based methods are discussed first. After that, autocorrelation and DFT methods are discussed. The Section ends with a short review of cepstrum-based pitch detection methods.

3.3.1 Waveform-based fundamental frequency detection methods

One of the oldest waveform-based pitch detection methods is the zero-crossing method. A zero-crossing is a point where the sample trajectory changes sign. Figure 3.5 illustrates zero-crossing fundamental frequency detection. Both Figure (a) and (b) have the same fundamental frequency, but Figure (b) has five additional harmonic frequencies. Zero-crossing points are sought, after which the fundamental frequency can be calculated. In the case of the simple harmonic signal shown in Figure 3.5 (a), the fundamental frequency can be easily detected. However, as Figure 3.5 (b) suggests, in case of a more complex signal, the basic zero-crossing method proves inadequate. The additional zero-crossings caused by the harmonics baffle the simplistic detection routine.

While simple and computationally inexpensive, zero-crossing pitch detectors are less accurate than more elaborate methods [Roa98]. Therefore they are mainly of historical interest only.

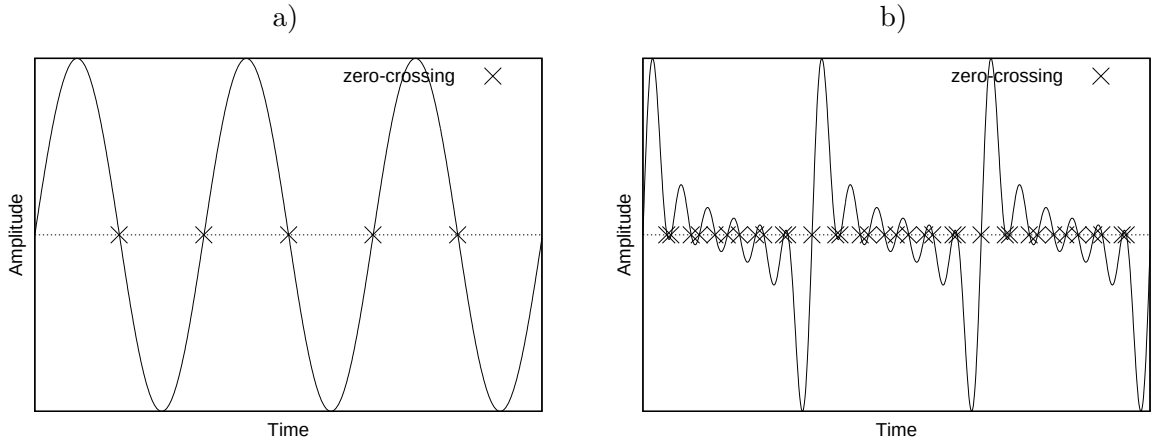


Figure 3.5: The principle of zero-crossing pitch detector.

3.3.2 Autocorrelation-based fundamental frequency detection methods

Autocorrelation is a transform of the signal, describing how well a signal correlates with itself delayed by different durations. At those delay values, in which the correlation is

strong, periodicity of the signal can be assumed. If the basic period t_0, the lag value of the first correlation peak associated with the desired signal, can be extracted, the fundamental frequency f_0 can be extracted trivially:

$$f_0 = \frac{1}{t_0}. \tag{3.4}$$

The fundamental frequency at each instance of time is calculated using *short-time autocorrelation analysis* [Rab77]. In short-time autocorrelation analysis, the signal is split into short segments, for which the autocorrelation function is calculated.

The short-time autocorrelation function of a short segment of signal $x(n)$ is defined as

$$\phi_k(m) = \frac{1}{N}\sum_{n=0}^{N-1}[x(n+k)w(n)][x(n+k+m)w(n+m)], 0 \leq m \leq M_C - 1, N \leq M_C \tag{3.5}$$

where M_C is the number of autocorrelation points to be computed, N is number of samples in the segment, k is the index of the starting sample of the frame, and $w(n)$ is the time-domain window function, such as the Hamming or Hanning window.[3]

For efficiency, short-time autocorrelation analysis is often implemented using fast correlation, which requires of computation of two discrete Fourier transforms:

$$\phi_k(m) = \frac{1}{N}F_D^{-1}\Big[F_D^*[x(k)]F_D[x(k)]\Big], \tag{3.6}$$

where F_D is a discrete Fourier transform, $()^*$ is a complex conjugate function and $x(k)$ is the signal segment.

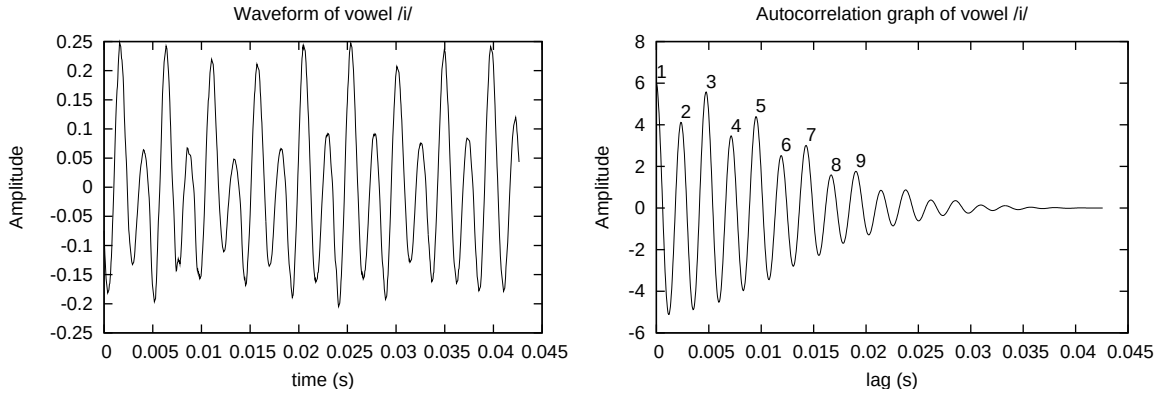


Figure 3.6: Waveform and autocorrelation graphs of vowel /i/.

Autocorrelation graphs, such as that in Figure 3.6, always have their largest peak at lag value 0 (peak 1 in Figure 3.6), at which point the signal correlates with itself. The envelope of the autocorrelation graph for periodic signals follows the autocorrelation of the windowing function being used. The highest real peak of the autocorrelation graph (peak 3 in Figure 3.6) is caused by the periodicity at the fundamental frequency (which

[3] If windowing is omitted, a rectangular window is effectively used.

is the strongest periodicity in the signal). There may be peaks between the zero lag value and the fundamental frequency peak (peak 2 in Figure 3.6), which correspond to the harmonics of the fundamental frequency. The other peaks in the autocorrelation graph are caused by the fundamental and the higher harmonics delayed by more than one cycle. Thus peaks 3, 5, 7 and 9 in Figure 3.6 are also caused by the fundamental frequency, and peaks 2, 4, 6 and 8 by the second harmonic (which also adds up to peaks 3, 5, 7 and 9).

Fundamental frequency can be extracted from the autocorrelation data by taking the lag value of the greatest peak of the autocorrelation graph and applying Equation (3.4) to the value. For example in Figure 3.6, the greatest peak (peak no. 3) is at lag value of 4.7 ms, so the fundamental frequency is 210 Hz.

One of the problems with basic autocorrelation is its poor resolution. Only very few signal components can actually be distinguished in the autocorrelation graph. Harmonic overtones also tend to slightly dislocate the peaks, giving false fundamental frequencies. To improve the resolution, Judith C. Brown has proposed a narrowed autocorrelation algorithm [BP89, Bro91], which makes the autocorrelation peaks significantly narrower, allowing more detail to show. The computational requirements are significantly higher, though.

Another problem with autocorrelation based methods is their subjectiveness to harmonic errors. In such case, the autocorrelation graph becomes distorted so that a peak caused by a higher harmonic for some reason is higher than the peak caused by the fundamental frequency. In Figure 3.6, this would happen if peak 2 were higher than peak 3. Some proposed methods to alleviate this problem include Sondhi's center-clipping method [Son68] or methods that aim at whitening the spectrum of the signal. In speech analysis, the use of Linear Predictive Coding (LPC) residual is common. This thesis also addresses the harmonic error problem in Section 4.1.

3.3.3 DFT-based methods

The Fourier transform is a signal transform, in which a spectral representation of a time-based signal is computed. Figure 3.7 shows the magnitude of the Fourier transform of time-based signal displayed in Figure 3.6. In ideal conditions, the fundamental frequency simply equals to the location of the strongest peak in the Fourier transform. In practice, the harmonic series of the frequency is often used to gather additional information.

In telephony applications, usefulness of methods based on DFT are limited because of the limited bandwidth of the transmission path. According to Rossing [Ros90], typical fundemental frequencies of speech are 110 Hz for male speakers, and 220 Hz for females. In singer's voice this may vary for more than two octaves. Since the lower limit of telephone bandwidth is 300 Hz, the transmission path causes the fundamental frequencies either to attenuate strongly, or disappear altogether. Because of this, harmonic matching methods have to be utilized in these conditions, and even they have to be adapted for missing fundamental frequencies.

Harmonic matching methods estimate the fundamental frequency using a theoretical

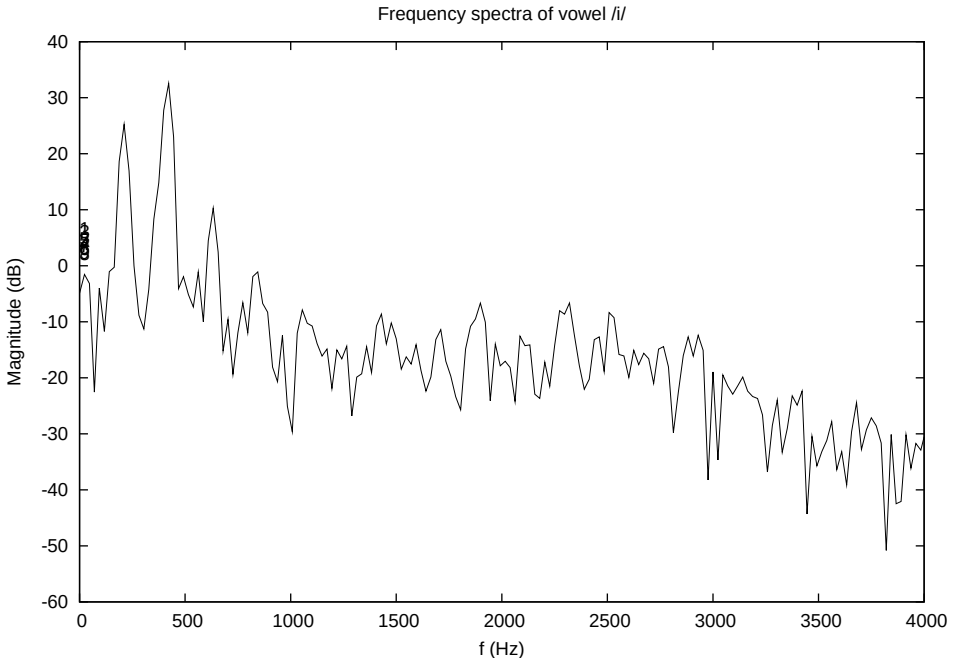


Figure 3.7: Spectrum of vowel /i/ after the transmission path.

model based on a maximum likelyhood of the fundamental frequency [DR91]. Harmonic components are matched to with fundamental frequencies that would best create such a harmonic series.

3.3.4 Cepstrum-based methods

Cepstral methods are commonly used in speech analysis, as they are able to separate the glottal frequency from vocal tract resonances. The Cepstrum $c_x(k)$ of a signal is obtained as follows:

$$c_x(k) = F_D^{-1}\Big[\log|F_D[x(k)]|\Big]. \tag{3.7}$$

The fundamental period can be derived from the peak in the cepstral domain representation.

The problems of cepstral pitch detection are that it can only analyze a rather narrow frequency range, and that it relies on the presence of harmonics in the input signal. It therefore fails to detect pitch in whistled input, as whistling produces a quite pure sinusoidal tone with few harmonics.

3.4 Automatic transcription of music

Transcription of music is defined to be the act of listening to a piece of music and of writing down musical notation for the notes that constitute the piece. In other terms, this means transforming an acoustic signal into a symbolic representation, which

comprises notes, their pitches, timings, and a classification of the instruments used [Kla97].

Computational auditory scene analysis (CASA) as an area of research greatly influenced by Albert Bregman and his book "Auditory Scene Analysis" [Bre90]. CASA tries to automatically analyse the acoustic information coming from a physical environment and to interpret the numerous distinct events in it. The scope covers everything from simple onset-detection algorithms to complete sound-description systems that aim to 'explain' unrestricted real-world sound ambiences into symbolic representations reflecting the perceived source structure [Ell96].

It is common in computational auditory scene analysis to distinguish from the auditory perception process three separate levels of representation. The low-level representations are those appropriate to describe the sound signal reaching the cochlea, whereas the high-level representations convey meaning, and are those to which human beings have cognitive access, such as "Someone is whistling the James Bond theme".

Between the high and low-level representations lies a network of representations labeled mid-level, about which there is little direct knowledge, as the relevant physiology of the brain is just beginning to be understood. The knowledge of these representations arises mainly from the constrains imposed on them at the lower and higher levels about which more is known [Ell95].

Despite the lack of knowledge of mid-level representations, computer models can be built to bridge the gap between low and high-level representations.

Since the low-level functions of hearing and signal processing are thought to be relatively well known, the approach of making mid-level representations has mostly been bottom-up. In these approaches, information is observed in an acoustic waveform, combined to provide meaningful auditory cues, and passed to higher-level processes for further interpretation. No information flows in the other direction. Because of the order of the information flow, the approach is also called *data-driven* processing [Kla97].

However, while bottom-up processes are needed to process the acoustic waveform information, to apply them to the whole auditory perception process would be a gross oversimplification. The mammal auditory perception process has a lot of *a-priori* knowledge or expectations of the auditory signals to be heard. This knowledge interacts with the hearing process so that information actually flows in both directions, and high-level representation contributes to the mid-level processes. This is called top-down, or *prediction-driven* processing. An extensive presentation of this approach can be found in Daniel Ellis' PhD Thesis [Ell96].

In music-related CASA the low-level representation is commonly considered as the acoustic waveform. While note as a high-level representational symbol of music has traditionally been implicitly accepted, contemporary knowledge of musical perception and cognition does not support this view [Sch96]. Ellis and Scheirer present a prediction-driven model of music perception and cognition, in which the listening process is not seen as a means of virtually transcribing the music into separate note patterns, but rather in which psychoacoustic cues in the data, such as "tracks", noisy regions, or

onsets in the time-frequency spectrum are highlighted. These psychoacoustic cues are compared against predictions based on the current musical context, and the agreements or disagreements between prediction and realization are reflected in a new representation of the musical situation. The representation often is chimeric, combining different simultaneous instruments in a single coherent event with its own individual properties, questioning the notational representation of the perceived sound [Bre90].

This work, however, concentrates on transcribing music to a form that resembles the original melody. Although trying to represent the original melody perceptually as closely as possible theoretically is very much of interest, the arbitrary limits imposed by the ringing tone formats practically reduce the problem back to more conventional automatic music transcription.

3.4.1 Bottom-up method of Bello, Monti and Sandler

Bello, Monti and Sandler describe a monophonic transcription system, which works with a simple bottom-up orientation [BMS00]. The scheme of their transcription system is shown in Figure 3.8. Their system uses a very straight-forward method of calculating pitch values of audible signal segments, converting the pitch values to MIDI key numbers, and finally collecting the single frame with similar MIDI key numbers to longer events. Their collector detects sound onsets based on the beginning of the steady state part of the signal, thus ignoring problems with noisy signal attack phase.

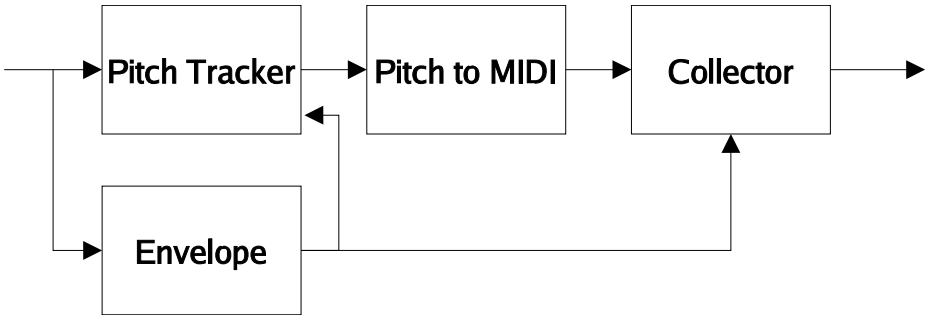


Figure 3.8: Scheme of the transcription system presented by Bello, Monti and Sandler.

3.4.2 Methods for signal segmentation

Signal segmentation, i.e. extracting information of the onset and offset of events, is of great importance in creation of ringing tones. The storage space for the melody in ringing tones is very limited, and avoiding detection of false event boundaries in the signal allows for longer melodies.

In his Master's Thesis Tristan Jehan divides signal segmentation field to frequency and time-based approaches [Jeh97]. Frequency domain method presented by him is based on calculating values of *High Frequency Content function* and *energy function*

for each frame and using them to calculate the detection function. This approach is also explored by Bello, Monti and Sandler [BMS00].

The energy function is calculated as a sum of the magnitude squared of each frequency bin:

$$E = \sum_{k=2}^{\frac{N}{2}+1} (|X(k)|^2) \tag{3.8}$$

where E is the Energy function for the current frame, N is the FFT array length, and $X(k)$ is the kth bin of the FFT.

The High Frequency Content function is defined as

$$HFC = \sum_{k=2}^{\frac{N}{2}+1} (|X(k)|^2 \cdot k) \tag{3.9}$$

where HFC is the High Frequency Content function of the current frame.

Since the noisy onset periods of events usually have a lot of high frequency content, a *detection function* comparing high frequency content of two consecutive frames can be formulated as

$$DF_r = \frac{HFC_r}{HFC_{r-1}} \cdot \frac{HFC_r}{E_r}, \tag{3.10}$$

where subscript r denotes current frame and $r-1$ denotes the previous frame. Peaks of the detection function indicate onsets of events.

Jehan also experimented with alternate event detection methods which worked by high-frequency filtering the HFC trajectory or by comparing the difference of spectra of the signals. In his experiments these methods were as reliable as the method presented above, but they could also indicate the offsets of notes fairly well.

The time-domain approaches explored by Jehan modeled the signal by an autoregressive statistical model and then used these statistics to detect changes in the model parameters.

Chapter 4

Algorithm implementation

The primary motivation behind this thesis was to create an economically viable ringing tone service, in which one could sing, whistle or hum a new and unique ringing tone for his mobile phone. The intended application for a major part designated the algorithm and technological choices for the implementation. The limitations were as follows:

- Transmission path: the algorithm has to work with telephone bandwidth and over speech codecs such as the GSM codec.
- Ringing tone formats: upon beginning the work, only mobile phones made by Nokia supported sending ringing tones using short messages. Therefore, the service has to obey the limitations of this format.
- Feasibility: since the service has to be economically viable, development costs have to be capped, and the computational efficiency of the service has to be good.

The aforementioned limitations led to the following initial design choices:

- Monophony. Since the output format is strictly monophonic, polyphonic pitch detection would be of no use.
- Autocorrelation based pitch detection techniques are used to bypass the bandwidth limitations, since they work well even when the fundamental frequency is missing from the signal.
- The design will be kept sufficiently simple to cut the development costs. This means that only traditional bottom-up based transcription algorithms will be used.

In the following sections different design and algorithm choices will be addressed in detail.

4.1 Fundamental frequency detection

The fundamental frequency detection is implemented using traditional autocorrelation-based techniques already described by Rabiner [Rab77]. While these are often not said to be suited for fundamental frequency detection when large ranges of pitches and varieties of spectra are encountered ([DR91, RCRM76]), they have been recently used in such purposes as well [Tol00].

Autocorrelation-based f_0 detection methods usually work by extracting the position of the largest peak in the autocorrelation graph, interpolating its exact position and converting it to the frequency domain. This was the first method implemented in Rring. It was soon noted that when testing the system over GSM codec, there were often anomalies in the autocorrelation graph, causing false peaks to be higher than the one caused by the fundamental frequency. These false peaks invariably resulted in detection errors. The cause of these anomalies is still uncertain, although imperfections in fundamental frequency detection of GSM codec have been proposed by Dr. Tolonen as one possible cause.

Figure 4.1 shows correct and anomalous autocorrelation graphs. Both represent one 40 ms frame taken from the same vowel /u/ sung by a male person. The voice is stationary, and the fundamental frequency is about 195 Hz. The total length of the vowel is about 300 ms. In the right-hand column, the fundamental frequency cannot be readily interpreted by taking the highest autocorrelation peak, while in the left-hand column it is possible. It can be seen that the energy of the second harmonic varies quite strongly, and when it is attenuated, the autocorrelation becomes degenerate.

To face the problem of degenerate autocorrelations, the fundamental frequency detection was enhanced to take into account not only the strongest, but several strongest autocorrelation peaks. While this did not help with the extreme worst case blocks like the one shown in the right column of Figure 4.1, it still reduced the number of errors and clearly enhanced the audible quality of the end result. The block diagram of the f_0 detector used in this work is shown in Figure 4.2.

The different blocks of Figure 4.2 are described in detail below.

4.1.1 Frame separation

First, the incoming signal is split to separate frames. The frames and the hop size used in the separation define both the temporal and spectral resolution of the f_0 detection. As of this writing, Rring uses frame length of 1024 samples or 42 ms at a sampling rate of 24 kHz. The hop size is 128 samples or 5 ms, although nearly identical performance can be achieved on a hop size of 256 samples as well. Using smaller hop size appears to help in latter stages of processing by creating extra redundancy in the signal, however.

4.1.2 Windowing and autocorrelation

When calculating a discrete Fourier transform (or by extension, fast autocorrelation) of a signal frame, the frame is actually considered to be one periodic segment of a

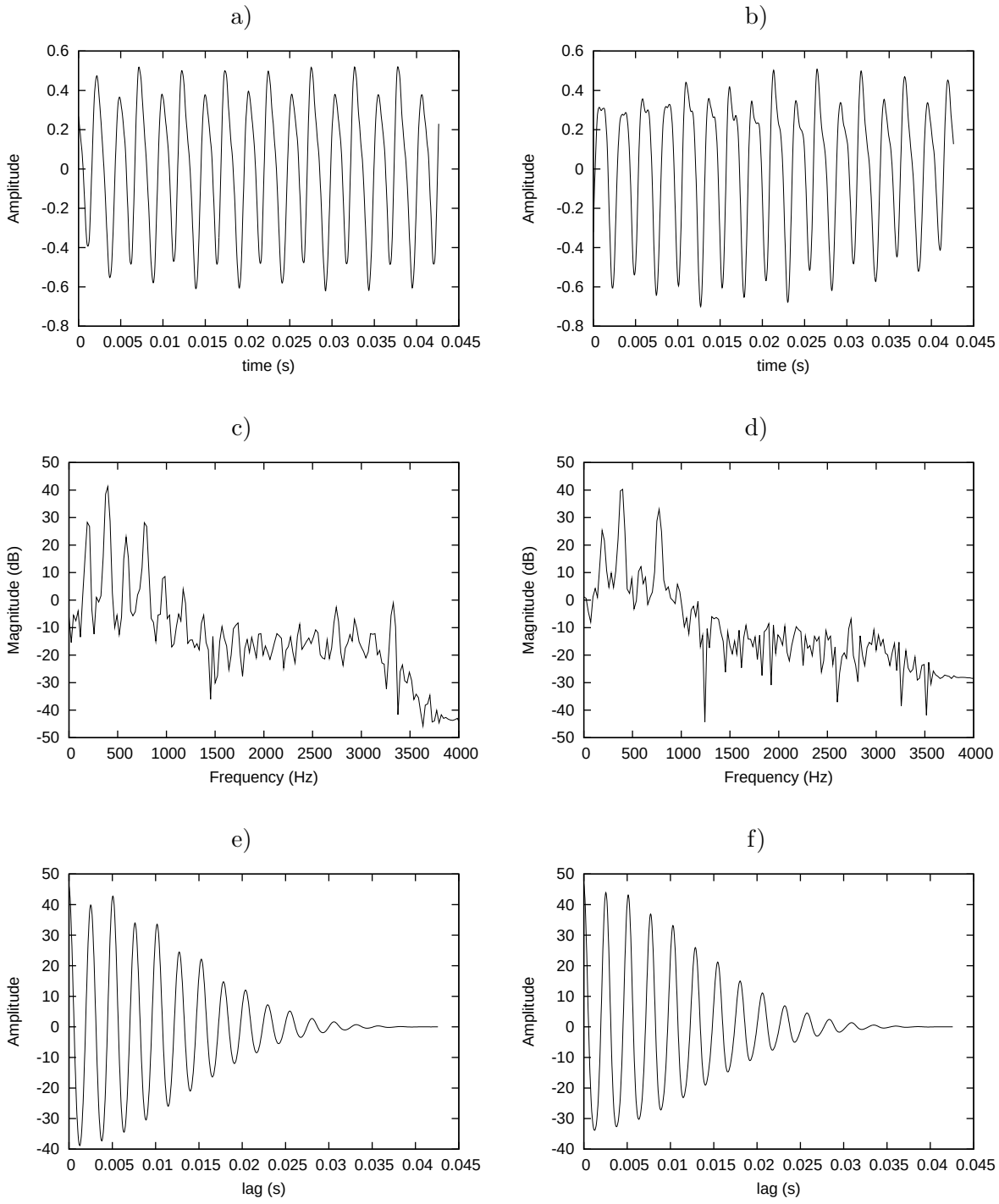


Figure 4.1: Waveforms (a, b), spectra (c, d) and autocorrelation (e, f) graphs of two recordings of sung vowel /u/, both having a fundamental frequency of approximately 195 Hz (corresponding to a lag value of 5.1 ms). The left-hand column (Figures a), c) and e)) represents a normal, while the right-hand column (Figures b), d) and f)) represent a recording leading to a degenerate autocorrelation. In the left-hand column, fundamental frequency can be derived from the autocorrelation graph. In the right-hand column, attenuated second harmonic distorts the autocorrelation and correct f_0 detection can be made no longer.

Figure 4.2: Block diagram of the fundamental frequency detector.

continuous signal. If the frequencies in the signal do not precisely match the observation interval, there exist discontinuities between the periodic segments. Figure 4.3 illustrates this phenomenon. These discontinuities exhibit so-called *spectral leakage* to the data set, which corrupts the transform [Har78].

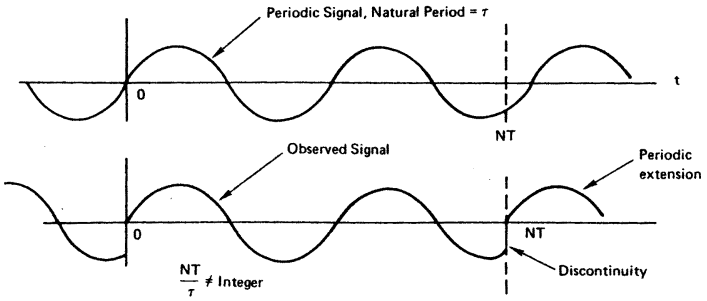


Figure 4.3: Periodic extension of sinusoid not periodic in observation interval. (After [Har78].)

According to Harris [Har78], windows are weighting functions applied to data to reduce the spectral leakage associated with finite observation intervals. The window is applied to data to reduce the order of discontinuity at the boundary of the periodic extension. This is accomplished by matching as many orders of derivative of the weighted data as possible at the boundary. Windowed data are smoothly brought to zero at the boundaries so that the periodic extension is continuous in many orders of derivative.

Many different window functions have been proposed, each having unique properties. However, all of the commonly used window functions (such as Hanning, Hamming, Blackman, Kaiser) reduce the spectral leakage efficiently enough to be useful in this work. Of these, Hamming windowing is used. See [Har78] for a thorough discussion of different window functions.

A short-time autocorrelation function is calculated from the windowed frames using Equation (3.6). This yields a list of autocorrelations of frames, from which the peaks can be detected and classified.

4.1.3 Peak detection

Autocorrelation peak detection was designed to collect repeating maxima of the autocorrelation signal, while also correctly taking the effects of windowing of the signal into account. The peaks are detected from the graph by selectively collecting all the local maxima from the autocorrelation vector. The details of the procedure are described

below.

First, all peaks in the autocorrelation vector are detected, as shown in Figure 4.4. A peak here is defined as any point in the autocorrelation vector, of which neighbors have a lower value.

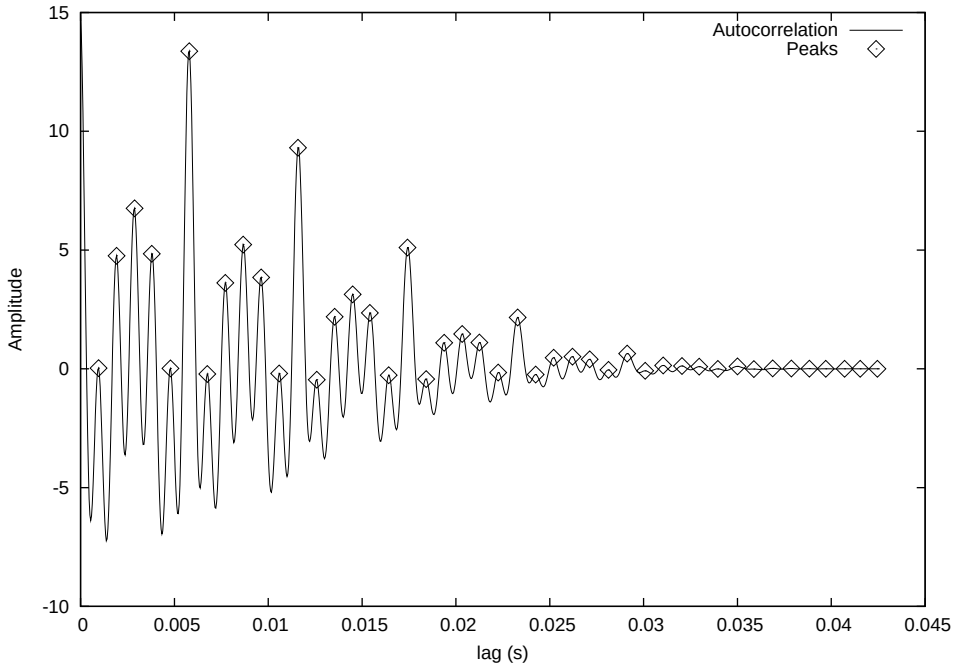


Figure 4.4: All local peaks in an autocorrelation vector. The frame is the same as in Figure 3.6.

After the local peak detection, the detected peaks are traversed in reverse lag order (i.e. from right to left). The first peak is automatically accepted, but after that the next peaks are accepted only if their height is at least 97% of the previous accepted peak (denoted as $\lambda = 0.97$). The value of λ is completely heuristic—it has been found experimentally, and there is no theoretical reasoning behind it.

Figure 4.5 shows the result of the peak detection algorithm. When peak no. 13 has been detected, the next peak to be detected has to have amplitude of at least λ (97%) that of peak 13. Thus, peaks 8–12 do not apply, but peak 7 clearly applies. After that, no peak exceeds the threshold. The first peak at lag value 0 is ignored, as it provides no information in f_0 detection.

Since the envelope of the autocorrelation vector follows the shape of autocorrelation of Hamming window (shown in Figure 4.6), it is evident that there exists a long low-amplitude tail, in which amplitude differences are negligible. Therefore there are plenty of false detections in the right-hand side of the autocorrelation vector. However, the whole length of the frame has to be analyzed, or low-frequency signals, which have sparse autocorrelation peaks, would not be detected at all.

The peak detection algorithm is presented in pseudocode below.

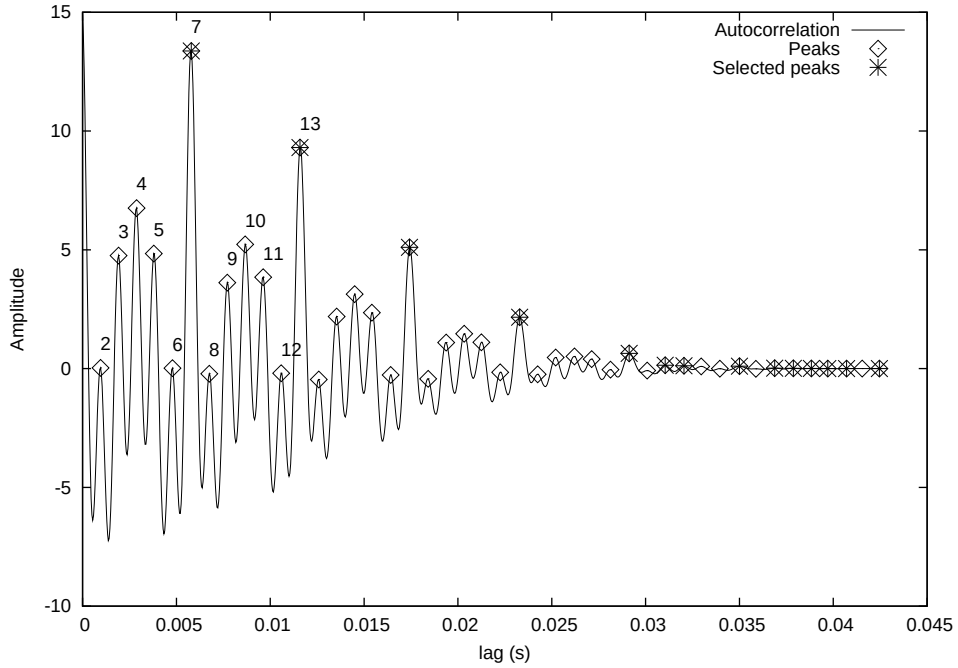


Figure 4.5: Peaks in an autocorrelation vector as detected by the peak detection algorithm.

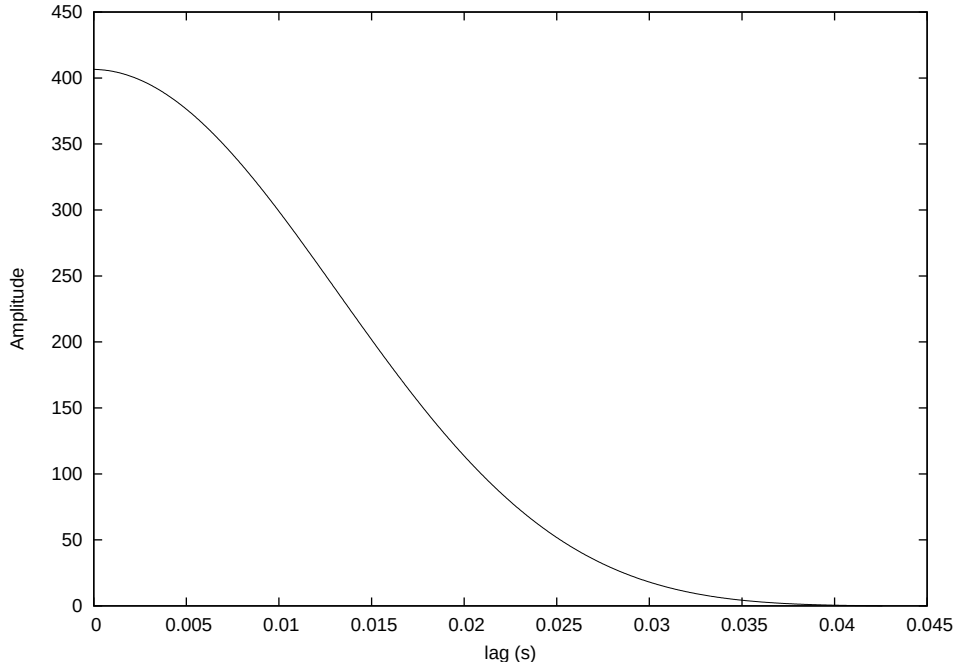


Figure 4.6: Autocorrelation of a Hamming window.

```
last_peak=0
lambda=0.97
for peak in reverse(peaks):
    if peak<lambda*last_peak:
        peaks=peaks.remove(peak)
    else
        last_peak=peak
```

4.1.4 Peak classification

For a periodic signal the autocorrelation function has a period equaling to that of the signal. Thus, assuming that part of the gathered autocorrelation peaks belong to the sequence defined by the fundamental frequency and the rest are peaks with no underlying periodicity, the peak corresponding to the fundamental frequency is the one that has the most integer 'harmonics' in the autocorrelation function:

$$n_N = N * n_0 \qquad \text{where } N \in \mathbb{N}, \tag{4.1}$$

where n_N is the N^{th} periodic of the fundamental lag n_0.

Using this knowledge, the following energy function for retrieving the fundamental lag can be deduced:

$$E(\nu) = \sum_{i=1}^{N} \min^2 \left(\frac{1}{2} - \left| \frac{1}{2} - \text{frac}\left(\frac{p_i}{\nu} \right) \right|, \kappa \right), \tag{4.2}$$

where $E(\nu)$ is the total error for the decimal lag value of ν, frac() is a function returning the fractional part of its argument, N is the total number of accepted autocorrelation peaks, p_i is the lag of i^{th} autocorrelation peak, and κ is an arbitrary maximum value for the error induced by a single peak. Currently, κ is set to a value of 0.1.

Equation (4.2) is basically a least-square optimization algorithm ($\sum \text{f}^2()$) with a maximum error limit ($\text{f} = \min(\text{g}, \kappa)$) added. Normally error samples are handled in least-square algorithms by dropping L samples with largest errors out, but since the number of peaks in the calculation varies greatly depending on the fundamental frequency, selection of fixed L would not have been feasible.

To find the fundamental lag, $E(\nu)$ must be minimized. To limit detection of false positives for small lag values matching many peaks, the search area is limited to be in the vicinity of detected peaks. Furthermore, lag values are limited to a minimum of 0.5 ms (2 kHz) and a maximum of 17 ms (60 Hz). These are reasonable limits for human voice scale, both when singing and whistling. The search has to be performed using interpolated values, since integer values do not give sufficient precision. The interpolation method used in this work is cubic interpolation.

Figure 4.7 shows the error function for the left-hand column of Figure 4.1. It can be seen that while the error varies greatly for small lag values, in the search region it is quite predictable and gives the correct result.

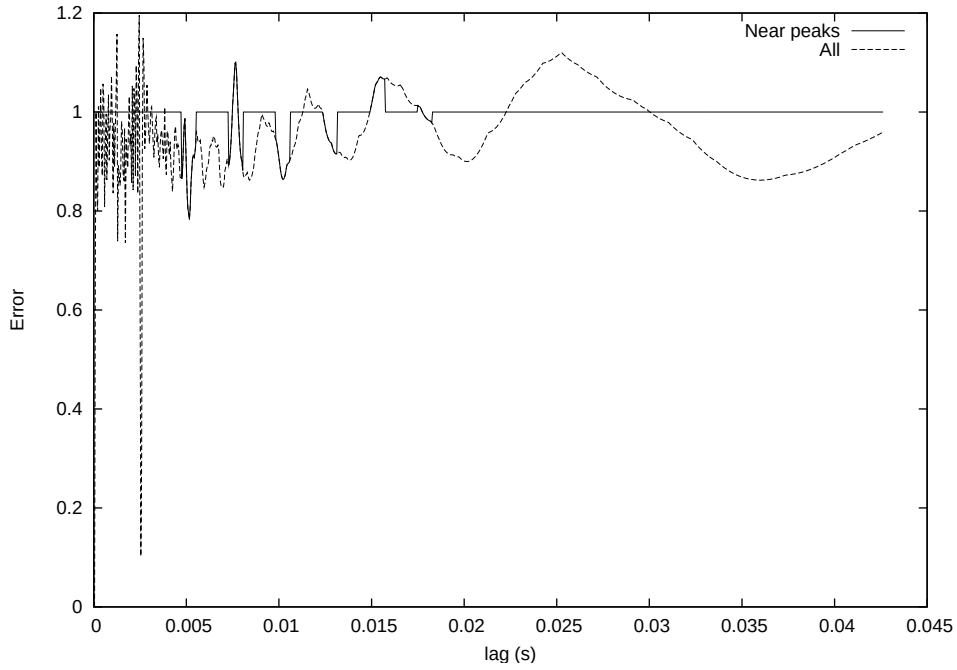


Figure 4.7: Error function for the whole frame and limited to vicinity of the peaks. Correct lag value is 5.1 ms, at which point there is a minimum in the near peaks -curve.

4.1.5 Voicedness detection

In addition to the fundamental frequency, some data regarding the reliability of the f_0 detection results is needed. In any recording, there exists ample amount of frames with no meaningful pitch. They might be simply frames with no sound energy, or frames with only inharmonic noise.

A simple method for assessing the strength of the harmonic components is to compare the highest autocorrelation peak to the value of autocorrelation at lag of 0:

$$v(k) = \frac{\frac{\phi_k(m_{max})}{\phi_k(0)}}{\frac{\phi_H(m_{max})}{\phi_H(0)}} \tag{4.3}$$

where $v(k)$ is the voicedness value of the frame, ϕ_k the short-time autocorrelation function given in (3.5), m_{max} is the location of the highest peak in that frame, and ϕ_H is the autocorrelation of a Hamming window.

The denominator part of Equation (4.3) is needed for normalizing out the effect of the Hamming window. Initially, the normalization was performed using computationally inexpensive triangle window approximation. While this worked well for most signals, it was noted that voiced frames with low pitch values would get artificially low voicedness values, thus hindering the performance of the pitch detector.

Figure 4.8 shows the voicedness values of a sine sweep ranging from 60 Hz to 2 kHz when using triangle window and Hamming window normalization. Although the curve

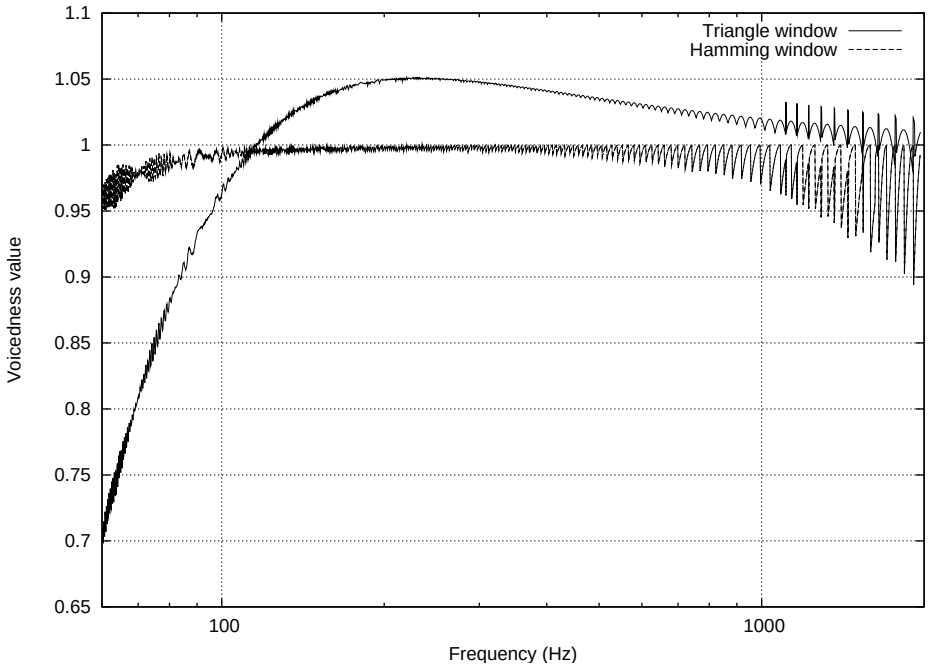


Figure 4.8: Effect of different window approximations on voicedness detection.

should have a constant level of 1, in triangle window normalization it ranges from 0.7 at 60 Hz to 1.05 at approximately 250 Hz, while in Hamming window normalization it stays considerably closer to the correct value. Such variations facilitate the need to use more precise normalization using autocorrelation of a Hamming window. The reason for the periodic variation of the value in high frequencies is not known.

The voicedness value is a decimal number with a value approximately between 0 and 1.05, depending on the window approximation used. A simple threshold, σ, is in the voicedness decision. A value of $\sigma = 0.8$ has worked well for the purposes of this project.

4.2 Event creation

As a bottom-up approach was selected for the transcription problem, the event creation is performed based solely on frame data. Event creation utilizes pitch and voicedness data in the process of forming note events from single frames.

This approach has its problems. Since no higher-level knowledge is used, all pitch values are allowed at all instances of time, leaving a lot of room for errors. Furthermore, as the tuning of singing of untrained people generally varies throughout the melody, no tuning correction can be performed. In some cases this results in disturbing one half-note errors.

First, it is determined, whether the frame has voiced content or not. This is done by comparing the voicedness value to a pre-defined threshold. The non-voiced frames are marked as pauses, while the MIDI pitch values of the voiced frames are calculated.

Consecutive frames with same pitch values are then concatenated to a single note.

4.2.1 Concatenation algorithm

Single frames are concatenated to note events as follows:

```
for frame in frames:
    if abs(note.pitch-frame.pitch) < maxdelta:
        note.pitch =
            (note.length*note.pitch+frame.length*frame.pitch)
            /
            (note.length+frame.length)
        note.length = note.length + frame.length
    else
        notes.append(note)
        note.pitch = frame.pitch
        note.length = frame.length
```

where `maxdelta` is the maximum allowed pitch difference of two note events, below which the events still are concatenated together. A value of 0.5 half-notes was observed to be suitable for `maxdelta`.

That is, as long as the pitch difference of the note being assembled and the next frame does not exceed `maxdelta` ($\frac{1}{2}$ half-notes), the frame is concatenated to the note so that the pitch value of the note is the weighted sum of the note and the last frame. Otherwise, assembly of a new note begins.

After the first concatenation phase, several heuristics are applied. First, a minimum length of notes and pauses is enforced. In our experiments, a minimum length of 43 ms (1024 samples at a sampling rate of 24 kHz) was found to be suitable. Shorter events are merged to preceding event without any pitch averaging. Then, events with equal rounded MIDI pitch are merged together. These two steps are used to remove short octave errors or unvoiced segments from the melody.

After this, one more merging pass is made, in which short notes with a pitch difference of less than one half-note are merged with adjacent notes. This is done to control the unstable attack and release periods of notes, which cause artifacts in the pitch trajectory.

4.3 Ringing tone conversion

After the event data is output from the event creation routine, it is converted both to an audible melody and to the Nokia Smart Messaging ringing tone format [Nok99].

The transformations were implemented in several steps. First, the event data is converted to MIDI file format [MID96] using a custom Perl script. Then, the MIDI file is converted to a Smart Messaging ringing tone file.

While the MIDI generation is a relatively simple format conversion, it also takes care of transposing the melody to pitch range allowed by Nokia mobile handsets. While not stated in [Nok99], it was found that the lowest note Nokia phones can play is C5, and the pitch range is three octaves. The specification [Nok99] defines a range of four octaves, beginning from C4, but it was noted that the phones transpose the lowest octave to the next-higher one. Furthermore, an optimal value for tempo is sought using least-square fitting. The optimization criterium is how well note lengths fit full-, half-, quarter- or one-eighth-notes at different tempos.

The ringing tone conversion is performed by another Perl program. The ringing tone specification imposes many restrictions on the possible tones:

- Note pitch values are limited to exact half-notes (as in MIDI).
- Tempo has to be explicitly defined.
- Note duration is limited to full, 1/2, 1/4, 1/8, 1/16 and 1/32 notes, with dotted and double-dotted variants.
- Pitch is a combination of scale (octave) and note-value, instead of a single value. Once a scale is defined, all further notes are in that scale, until the next scale definition.

If tempo detection fails, note duration restrictions may cause unnecessary quantization of note durations. This has been confirmed in tests, although it has not been a significant problem. The pitch encoding causes problems by making prediction of the length of the final melody difficult. For example, the sequence B-C-B-C, which crosses scale-boundaries three times, therefore takes nearly twice the space of sequence C-D-C-D, which stays on one scale only.

The pre-listening melody is generated using the event data as the source. Initially the melody was generated from the MIDI file with a separate MIDI wave-table synthesizer program, but to enhance portability a simple sine-tone synthesizer was implemented in Java. This synthesizer uses the event data directly, so that the MIDI conversion is at the moment only used as an intermediate step in the actual ringing tone conversion.

4.4 Computational efficiency of the ringing tone conversion

The bulk of the processing time in Rring is spent in the C language analyzer program. Table 4.1 shows the execution profile of the analyzer. It can be seen that nearly 80% of the execution time is spent in the FFT routines, so in practice they determine the total execution time. FFT is used in Rring to calculate the short-time autocorrelation for each window. Different methods for optimizing Rring processing power requirements would be to use some highly optimized FFT implementation or to increase the hop size to reduce the amount of analyzed frames.

Figure 4.9 shows execution times of the melody detection program measured with two different hardware. In Figure 4.9 a) a modest 375 MHz Intel Celeron processor

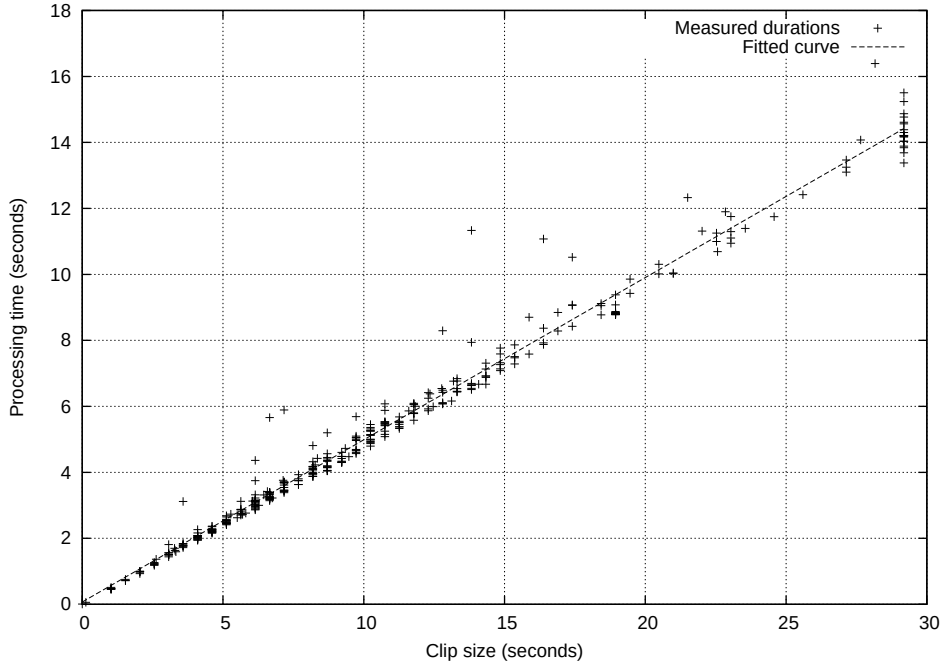
% time	cumulative seconds	self seconds	calls	self us/call	total us/call	name
41.33	1.86	1.86	3954	470.41	470.41	ifft
37.11	3.53	1.67	3954	422.36	422.36	fft
7.56	3.87	0.34	165137	2.06	2.06	sum_sq
6.44	4.16	0.29	3954	73.34	966.11	autocorr_win
4.22	4.35	0.19	3953	48.06	134.08	getMaxLag
2.00	4.44	0.09	5	18000.00	18000.00	biquad_section
0.67	4.47	0.03	1	30000.00	130000.00	getPCMData
0.44	4.49	0.02	1	20000.00	20000.00	highpass
0.22	4.50	0.01	168960	0.06	0.06	st_Alaw_to_linear
0.00	4.50	0.00	7905	0.00	0.00	freqToMidi
0.00	4.50	0.00	3953	0.00	966.11	autocorr
0.00	4.50	0.00	513	0.00	0.00	LLSnodePtr
0.00	4.50	0.00	420	0.00	0.00	LLSnodePtr2Next
0.00	4.50	0.00	192	0.00	0.00	LLSnodeAppend
0.00	4.50	0.00	192	0.00	0.00	LLSnodeAppendFrom
0.00	4.50	0.00	93	0.00	0.00	LLSnodeDelete
0.00	4.50	0.00	4	0.00	0.00	LLSnodePtr2First
0.00	4.50	0.00	2	0.00	0.00	ListInit
0.00	4.50	0.00	1	0.00	0.00	LLScreate
0.00	4.50	0.00	1	0.00	0.00	LLSsystemInit
0.00	4.50	0.00	1	0.00	530000.00	analyzeFrames
0.00	4.50	0.00	1	0.00	0.00	biquad_iir
0.00	4.50	0.00	1	0.00	0.00	detectEvents
0.00	4.50	0.00	1	0.00	0.00	fftInit
0.00	4.50	0.00	1	0.00	966.11	rringAnalysisInit
0.00	4.50	0.00	1	0.00	0.00	rringAnalysisRelease

Table 4.1: Execution profile of RTC pitch detector and event creator.

is used, while Figure 4.9 b) represents execution times of a more modern 1400 MHz AMD Athlon XP processor. The measurements utilize 379 different real-life recordings, recorded between December 2000 and June 2001. The length of the recordings varies from zero to 30 seconds, the average length being 11.1 seconds. Both computers were running some other services during the measurements, which may have induced some extra deviation in processing times. This, however, represents real-life operating conditions.

The processing speed of the Celeron processor is 16.3 kS/s (kilosamples per second), or 2 times real-time. While this performance is inadequate for proper scalability, the Athlon processor has a more acceptable processing speed of 63.2 kS/s, or approximately 8 times real-time.

a)



b)

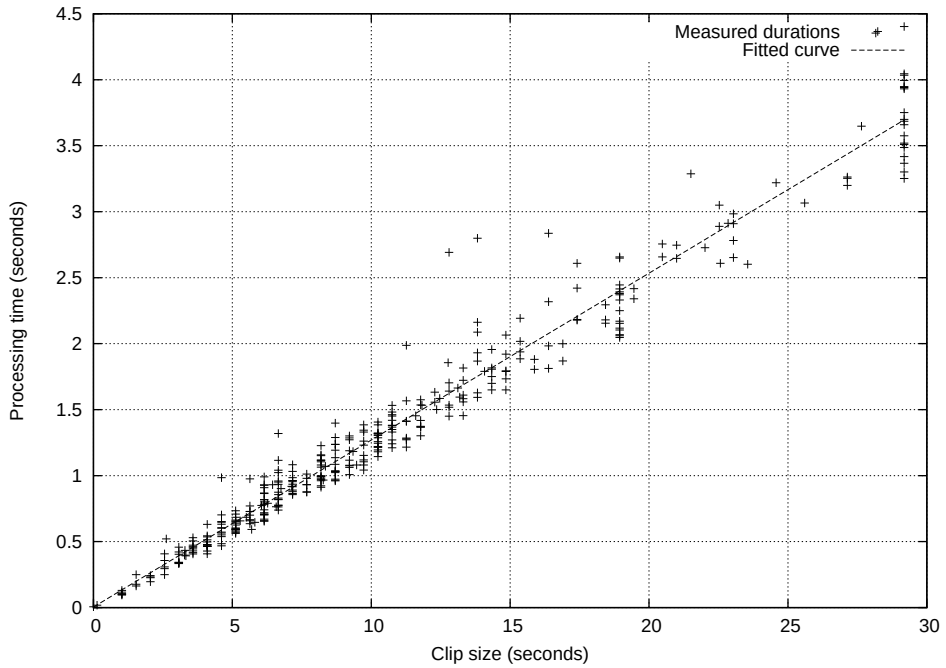


Figure 4.9: Execution times of the melody detection program as a function of recording length. Figure a) shows processing time taken by a Celeron 375 MHz processor, while Figure b) shows processing time required by a more modern Athlon XP 1400 MHz processor.

Chapter 5

Implementation of a ringing tone service

5.1 Design goals

The design goals for Rring service have been quality, performance, flexibility, reliability, scalability, usability and profitability. The quality requirement in this context refers to the quality of produced ringing tones. They should resemble the original piece wherever possible, without unnecessary omissions or undesired artifacts. Quality considerations are addressed in Chapter 4, so they are not addressed here anymore.

The performance requirement means that the service has to be able to produce ringing tones in a reasonable amount of time (say, 10–20 seconds for a non-interactive service, or less than 10 seconds for an interactive one), and it has to be able to handle several simultaneous service requests without dropping any calls or recordings at any point.

The flexibility requirement means that the service has to be easily adaptable and modifiable to both new operating system environments and use purposes. For example, even though the system currently runs on a Linux platform, it has to be able to be ported to Solaris when needed. Similarly, if the service environment changes (a different IVR platform is used, the ringing tone format changes, etc.), it must be possible to adapt the software to the new situation easily.

The requirement of reliability: The system should never go inoperable, nor should it drop any calls or ringing tones unintentionally. As the customer pays for the service, he should always receive some kind of feedback from the service. In a distributed system, all modules should work independently and handle communication breakdowns gracefully. For example, if the IVR system cannot reach the conversion module, it should queue the request and retry sending after a short while.

Scalability means that it must be possible to expand and still maintain the system as it grows beyond original usage expectations. For example, it should be possible to balance the load of different components by clustering, and there should be no fatal bottlenecks in the system design. That way, construction and maintenance of service handling 200 simultaneous users should not be significantly more difficult than implementation of a basic service handling only two simultaneous users.

Usability is defined as "the effectiveness, efficiency, and satisfaction with which specified users achieve specified goals in particular environments."[DFAB98] In the case of the Rring service, the users are the customers of the service and the specified goal is to successfully record and receive a ringing tone. If the users do not perceive the system as usable, they will not try it again, nor will they recommend it to their friends.

Profitability is the most straightforward goal: the service needs to make profit. The development, infrastructure and running costs have to be kept under control, and unless the system is perceived as a special novelty by operator, the service income has to exceed costs at some time-scale.

5.2 System architecture

Rring service in its basic form has three functionally separate modules:

1. Recording acquisition
2. Ringing-tone conversion
3. SMS sending

In the first part, a recording from the user is obtained. This is done with an interactive voice response (IVR) platform, which receives telephone calls, plays prompt recordings to the customer, records customer input and then sends the results forward to the ringing-tone conversion module. The ringing-tone conversion module receives the recording from the previous module and converts the recording to a ringing tone as described in Chapter 4. After the ringing tone is generated, it is sent to the SMS gateway, which then sends it to the SMS center of the mobile operator.

There are several ways to implement this functional chain. The most straightforward way is a monolithic system (depicted in Figure 5.1), in which all these modules are placed in the same computer, so they can closely interact. This may be desirable in development phase, as the structure is very simple and easily understandable. However, in a production system it is a somewhat simplistic approach, as it is impractical for clustering, and limits available hardware, software and business partners (in cases where some parts of the functionality are preferred to be externalized) too much.

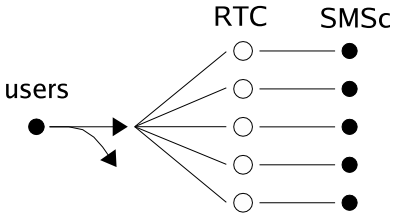


Figure 5.1: Monolithic service architecture. All service components are located in a single computer, and parallel requests are handled concurrently.

Since the three aforementioned modules are functionally quite separate, it is possible to construct a distributed system, where each module is located in a different computer system or even in a different company. In such an arrangement, some messaging system needs to convey the recordings and auxiliary data such as telephone numbers from the IVR to the ringing-tone conversion module and from the ringing-tone conversion module to the SMS sending module. While the messaging system increases system complexity, it makes the system a lot more flexible: the IVR can be an autonomous module, which transmits the recordings to a different location for ringing-tone conversion. Similarly the SMS gateway may be provided by a separate company. The obvious drawback is that all of the service chain may be no longer in control of a single company, potentially making updates and service development more cumbersome than necessary.

There are two different ways to implement a distributed system. The simpler of these implements non-queuing, or synchronous messaging. Such a system is shown in Figure 5.2. In this system, once the IVR system has finished recording, it sends the results forward and waits for a result. If the ringing-tone conversion module is not available for some reason, IVR fails immediately and reports the problem to the user. Depending on how the next module is designed, the result may come immediately or only after the ringing-tone conversion module has sent the resulting ringing-tone to the SMS-sending module. A synchronous system should be implemented only over reliable networks, e.g. local-area networks, as any disruption in the network will make the service drop messages.

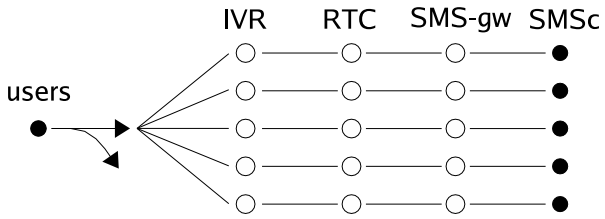


Figure 5.2: A synchronous distributed system. Different modules are located in separate computers, and the messaging between them is non-queuing.

The other way to make a distributed system is to implement queuing, or asynchronous messaging, shown in Figure 5.3. This works so that the IVR puts the recorded message to a sending queue, and another process reads the queue and sends the recordings to the ringing-tone conversion module. In case of network problems, the messages stay in the queue, and are sent when the ringing-tone conversion module becomes available again. After placing the recordings to the queue, the IVR system can resume operation, so the customer may received feedback immediately after recording. Depending on the implementation, the ringing-tone conversion module may send an acknowledgement message after handling the recording. Also, the ringing-tone conversion module respectively places the generated ringing tones to a queue, from which the ringing tones are sent to the SMS gateway. This schema is most complicated to construct, but it also provides greatest reliability and flexibility for the system. Reporting results back to the IVR system also requires a separate upstream messaging channel, which also

complicates the modules when constructing interactive services. If the user interface design does not require immediate feedback, the upstream messaging channel is not required, as the system works reliably also without it.

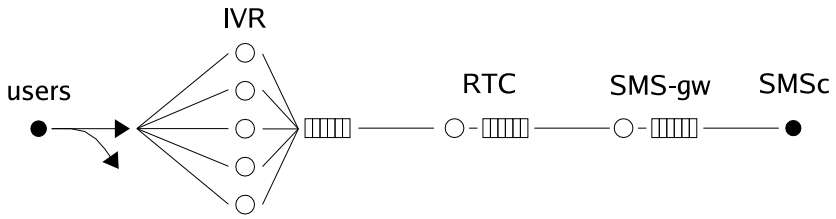


Figure 5.3: An asynchronous distributed system. Different modules are located in separate computers, and the messaging between them is queued.

It is possible to implement a distributed system as partially asynchronous. If, for example, the IVR service provider chooses to not implement queuing, it can still be used in other modules. That just creates a single weak link in the service chain. A partially asynchronous system in shown in Figure 5.4.

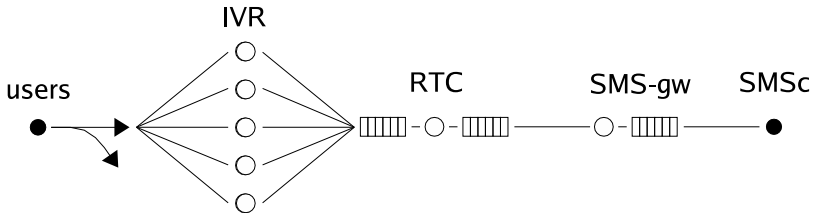


Figure 5.4: A partially asynchronous distributed system. The IVR operates synchronously, and the requests are queued in the ringing-tone conversion module.

Furthermore, the service can be also only partially distributed. In such an arrangement, for example the IVR platform and ringing-tone conversion service are embedded in a same system, and the SMS gateway is separated. Such arrangements may be worth considering in some special situations, such as when only low usage is expected, or in development phase, in which rapid changes are implemented both to IVR and ringing-tone conversion services.

5.3 Traffic analysis

In terms of capacity, the service has two bottlenecks. The first, and the most obvious one, is the number of incoming telephone lines in the IVR service. The second is the amount of processing power available for the DSP calculation. While these two can be analyzed separately, they are intricately linked to each other. If the processing backend cannot handle the load generated by the IVR server, it can lead to situations in which incoming calls are accepted and the customer is charged for the call, while the service is inoperable because of excessive backend server load, resulting in excessive waiting

times or service collapse.

5.3.1 IVR service quality

From a teletraffic theory point of view, the IVR service can be seen as a pure loss system [HP93]. It has n parallel servers, which accept calls. If all the servers are reserved, any additional calls are dropped. Note that while a queueing system would increase the service quality, it may not be possible to implement because of the planned revenue models. That is also the reason why queueing systems are not considered in this analysis.

For a pure $M/G/n/n$ loss system with calls arriving according to a Poisson process with a rate of λ and having an average holding time of h, the traffic intensity is defined as

$$a = \lambda h. \tag{5.1}$$

This is used in *Erlang's blocking formula*, which defines call blocking probability (B_c) in terms of number of lines and traffic intensity [HP93]. Call blocking is the probability that an arriving call finds all n channels blocked. Erlang's blocking formula is defined as:

$$B_c = \mathrm{Erl}(n, a) = \frac{\frac{a^n}{n!}}{\sum_{i=0}^{n} \frac{a^i}{i!}}. \tag{5.2}$$

The service quality $1 - B_c$ for a service with 10, 20 or 30 lines is shown in Figure 5.5. If service quality of 0.8 is deemed as satisfactory, it can be seen that a 10-line IVR platform can manage a service with a traffic intensity of 10 Erlang. Given a mean call duration of 1 minute, 10 Erlang equals 600 calls an hour. A 30-line service can then manage a traffic intensity of 34 Erlang, or 2040 calls an hour, which should be plenty for even big markets.

As the IVR platform telephone network connection typically is one or more ISDN E1 lines (30 simultaneous voice lines), the IVR capacity typically is not a problem in implementing the service.

5.3.2 Ringing tone conversion service quality

The RTC backend calculation process can be seen as a pure waiting system, in which there is one server and a service queue of infinite length. Theoretically it does not matter whether the system works as a "First in, first out" (FIFO) queueing system or as a processor sharing (PS) system [1]. Thus, it is an $M/M/1$ queueing system [HP93]. Requests arrive according to a Poisson process with a rate of λ, and the service rate has a mean of μ, and has an exponential distribution. It has to be noted that the actual service rate depends on the melody length, which is clipped to a certain period, and the rate distribution is probably not exactly exponential. However, the assumption of exponential distribution is kept because of its mathematical simplicity.

[1] These correspond to asynchronous, or serial, and synchronous, or parallel processing described in Section 5.2

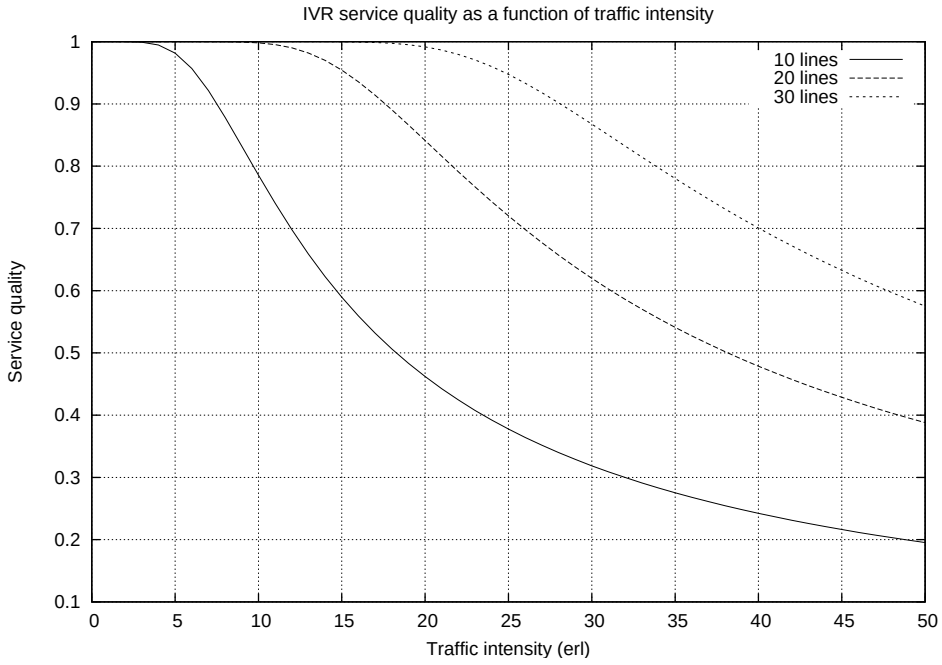


Figure 5.5: IVR service quality as a function of traffic intensity.

Traffic load is defined as a quotient of request arrival rate and the service rate:

$$\rho = \frac{\lambda}{\mu}. \tag{5.3}$$

The value represents the utilization of the server.

Using λ, μ and maximum allowed service wait time, z, a parameter describing the service quality of a waiting system can be defined:

$$P_z = \mathrm{Wait}(\lambda, \mu, z) = \begin{cases} \frac{\lambda}{\mu} \exp((\lambda - \mu)z) & \text{if } \rho < 1 \\ 1 & \text{if } \rho \geq 1 \end{cases}. \tag{5.4}$$

P_z is the probability for a service incident to take longer time than z. Assuming service time of 2 seconds ($\mu = 1/2$, assumption made from the results of the computational efficiency tests using modern hardware in Chapter 4.4), the service quality behaves as seen in Figure 5.6. A 1-processor server would begin to collapse after a request intensity of 20 requests per minute, while a 2-processor system could manage 50 before the service quality begins to deteriorate. In practice, if the service is designed to be interactive, even 10 seconds waiting time (as shown in Figure 5.6) is already very long.

One possibility would be to dispatch the service process to the IVR and meanwhile playback some music or advertisements to the customer.

Although previously stated that FIFO and PS systems are identical regarding traffic intensity aspects, this assumption holds only with systems having an unlimited amount of memory. In real systems, computer main memory is a limited resource, and running out of it induces swapping behavior, where less frequently used memory pages (processes or parts of processes) are moved to a swap file residing on a hard disk drive.

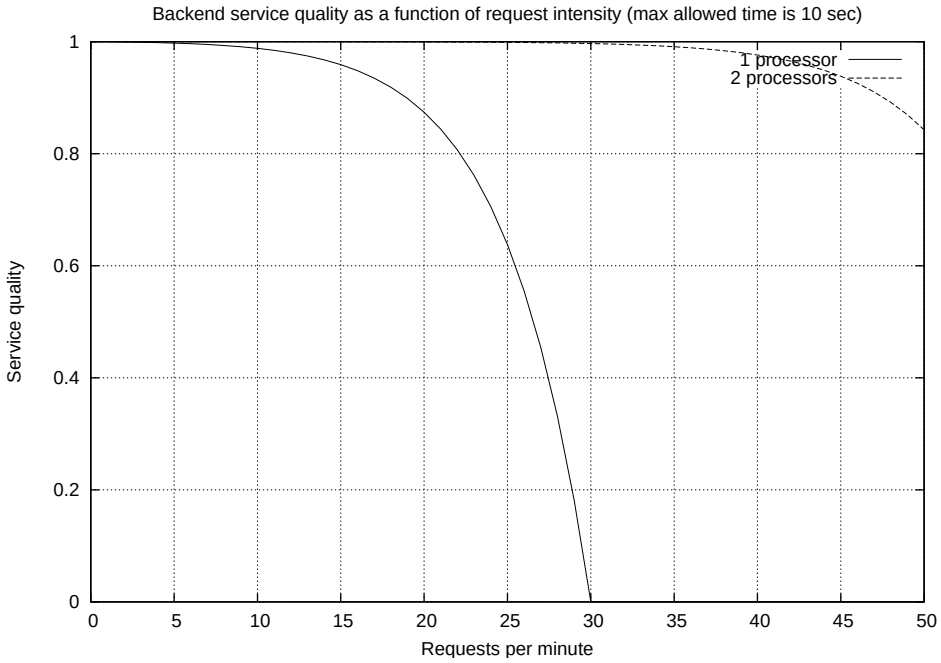


Figure 5.6: Backend service quality as a function of request intensity.

Hard disks are generally over 100 times slower than main memory, and because of the rotational delays seek latency differences are even larger. Some swapping does not radically affect system performance, since modern multi-tasking operating systems are able to allocate processing time to in-memory processes while waiting for disk reads to complete. However, when memory load still increases, the ratio of in-memory processes and swapped-out processes decreases until swapping occurs at most process contexts switches. At this point process throughput collapses, and the computer enters a state commonly known as "thrashing", where time is spent mostly waiting for disk activity. Since the RTC service can use up to 24 MiB of main memory for the analysis, it can easily be seen that on a computer with 128 MiB of main memory, only five concurrent processes fit into memory at one time, and this does not take even the memory required kernel or any other services into account. Even with 512 MiB of main memory, the amount of concurrent processes fitting to memory is only about 20, not counting the memory use of the kernel and other processes.

In FIFO systems, there exists only one process continuously processing queued service requests. Since the process count is constant, memory requirements vary only little, and swapping never occurs. On the other hand, in process sharing systems, a new process is started immediately when service request arrives, so there may be multiple parallel processes at a given time. If the traffic intensity rises sufficiently, swapping and trashing may occur. On modern hardware, this happens only with comparatively high traffic intensities, but it is still quite possible to reach such behavior.

Some simulations to reveal the differences of PS and FIFO system performances were performed. In the simulation, service requests appear at random intervals with a traffic intensity of 0.8 Erlang. Holding time is defined to be constant 5 seconds. FIFO

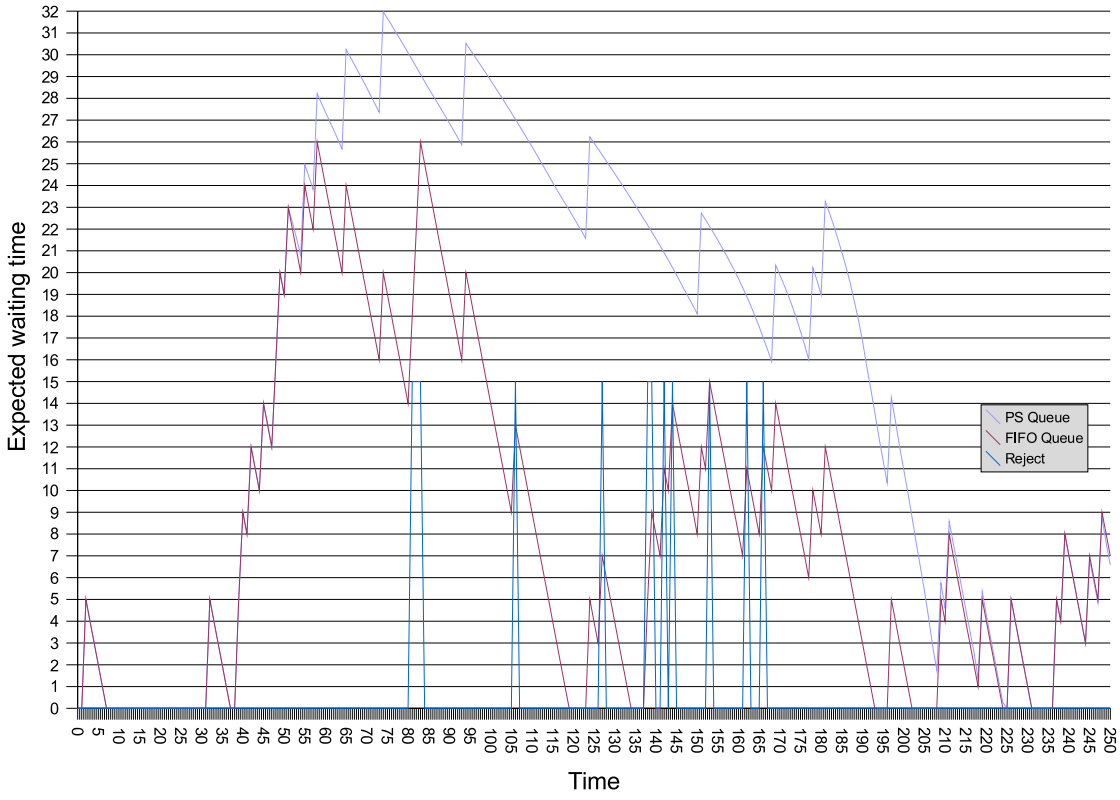


Figure 5.7: Backend service load simulation graph.

and PS systems are simulated separately. In FIFO, there is an infinite queue and a default processing time. In PS, all requests are accepted immediately, but throughput proportionally decreases when there are more than six concurrent processes. This behavior simulates swapping and thrashing. Furthermore, there is a maximum limit of ten simultaneous processes, which simulates the effect of finite main memory. In real systems the process and memory limits may be higher, and these values should be seen only as representative.

Figure 5.7 shows results of simulation of backend load. The x-axis depicts time, while the y-axis shows the expected waiting time (sum of unhandled request durations). It has to be noted that in PS system, the actual waiting time may be longer since high process counts induce swapping, which decreases throughput. There is a burst of requests beginning at about $t = 30$ s, during which the expected waiting time increases considerably. It can be seen that the FIFO system is able to consume queued requests without problems, while the performance of PS systems strongly deteriorates after the load had risen sufficiently. The FIFO system is able to clear the request burst in 70 seconds, while the effects last significantly longer in the process sharing system. What is even worse, the PS system is forced to reject requests because of insufficient memory, and the rejects are spread over a long duration of time. Depending on the thrashing behavior and the virtual memory management capabilities of the operating system, it is even possible to reach a state where the system is not able to recover from thrashing.

5.4 Software architecture

5.4.1 Scalability

While the development of Rring has occurred in a small one-computer environment, it is clear that commercial deployment has very much different requirements for operation. The most obvious requirement is sufficient capacity to handle the expected amount of connections. While a limited system of 6–8 simultaneous users might be implemented using a PC with several ISDN controllers, practical implementations require a commercial interactive voice response (IVR) system with a 30-line ISDN PRI or similar connection. Section 5.3 outlines some traffic estimates for the service.

Scalability is another key requirement for a commercial installation. The service has to scale in terms of user amounts, so that both the number of incoming lines and the available processing power have to be easily increased when necessary, without proportionally increasing administrative or maintenance workload.

There are several possible models to scale the service. Outlines of a few different models are shown in Figure 5.8. Figure 5.8 (a) shows the monolithic configuration, in which both the IVR service and ringing-tone creation are integrated to the same computer. This arrangement is very simple to set up and easy to operate. This method, however, allows only for 30 simultaneous users (limited by the capacity of ISDN E1 connection). Furthermore, the amount of incoming phone lines and the processing power required for handling the service are difficult to balance in model shown in Figure 5.8 (a), since handling 30 simultaneous users requires considerable amount of

processing power, which requires server hardware with support of multiple central processing units. The model can be scaled up either by acquiring an IVR system with multiple ISDN E1 connections, as in Figure 5.8 (b) or by replicating a single server (Figure 5.9 (a)). While E1 spans are an easy and straightforward way to scale ordinary IVR services, the still increased processing power requirements make it an even more impractical solution for Rring. On the other hand, server replication necessitate the use of a separate database server for any kind of data storage purposes, Figure 5.9 (b).

Probably the most practical models to implement the service separate the IVR platform from the backend processing as shown in Figure 5.10 (a). In such a model, the IVR platform handles the call logic and sends processing requests to the backend as remote procedure calls (RPC). Different RPC protocols are discussed in the next section.

High availability can be achieved by replicating any critical components of the service. This can be performed in any configuration, but models shown in Figure 5.9 or 5.10 (b) are especially suitable for replication. The IVR platform can be configured to have a hot spare, that is, an idle backup IVR server that takes over the service in case of a primary server crash. Similar replication or clustering can be performed to all other components (IVR system and SMS gateway) as well, although in distributed systems the network quickly becomes the weakest link. While it is possible to provide redundant service by using two separate Internet service providers, it is neither straightforward nor simple to do.

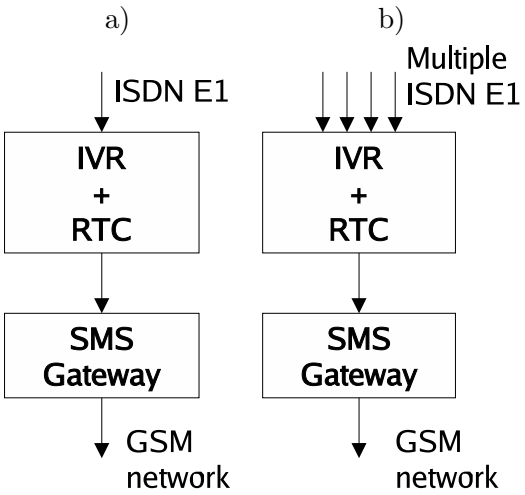


Figure 5.8: Basic IVR line bundling.

5.4.2 Remote procedure call protocols

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. RPC uses the client/server model. The requesting program is a client and the service-providing program is the server. Like a regular or local procedure call (often known as a *function call* or a *subroutine call*), an RPC is

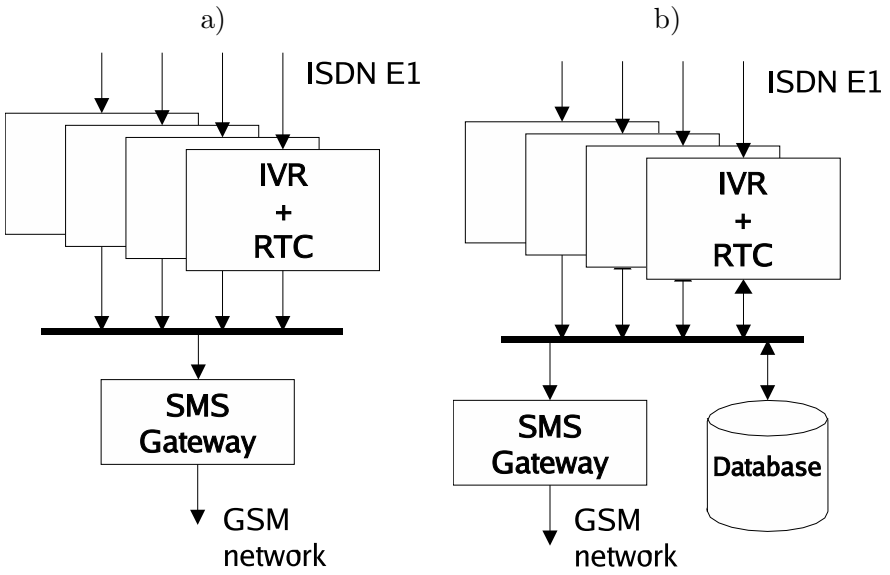


Figure 5.9: Multiple IVR servers.

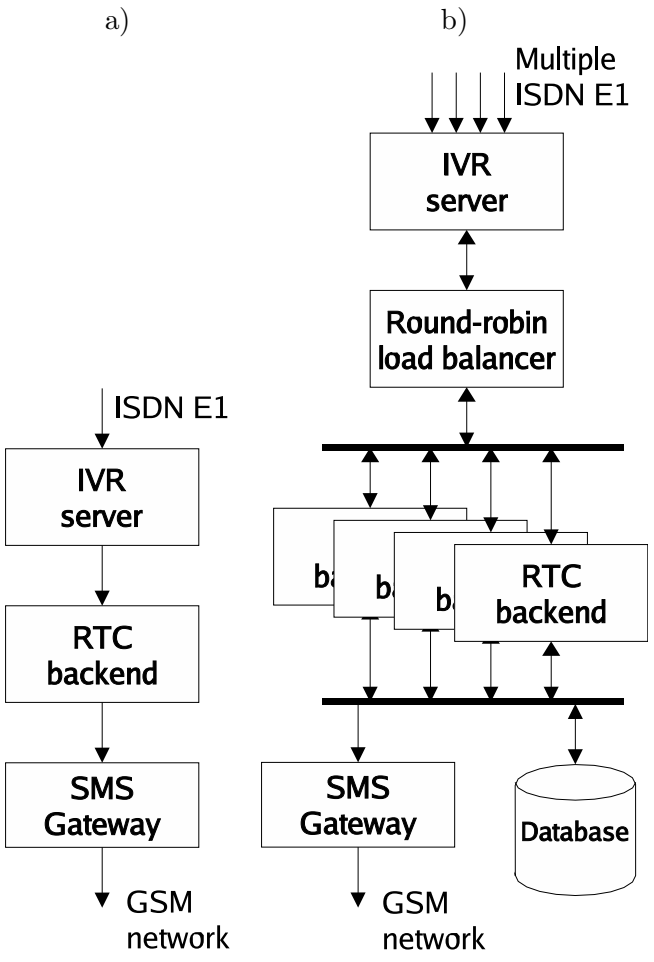


Figure 5.10: Separate processing backend.

a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure are returned.

There are several alternative RPC protocols, including (but not limited to) CORBA, XML-RPC, SOAP, and Java RMI [OMG02, XML02, BEK+00, Jav02]. Although HTTP by itself is usually not regarded as remote procedure call protocol, it can be used as such, and so it is included in this presentation.

Each RPC protocol has its strengths and weaknesses. HTTP requests are most simple, and they are easily implemented in VoiceXML, the standard IVR service definition language. HTTP based services are also very easy to scale up, so that the processing power and IVR capacity can be replicated individually. Figure 5.10 (b) shows such an arrangement. This provides increased redundancy as well, providing an easy way to make the service highly available. On the other hand, using raw HTTP calls provides no standardized RPC parameter passing method (other than HTTP GET and POST parameters). HTTP interfaces are quite commonly used to provide simple service interfaces such as short-message sending and receiving services.

CORBA is a stable and widely used RPC protocol, and suitable for IVR-backend communication as well. In CORBA, the interfaces are defined in interface definition language (IDL), which provides accurate and unambiguous description of the call interface. The IDL files are then used to create *proxies* and *stubs* in the target programming languages. Proxies are function libraries, which make calling the interface methods look like calling native, local functions. Stubs are function templates, which are implemented to provide server side implementation of the said interface. IDL interface definitions make it fairly straightforward to implement interfaces, but the process of defining and implementing a CORBA interface still requires many steps and has quite a steep learning curve. The actual network traffic is performed in binary form, which makes CORBA quite efficient in that manner. CORBA is widely used both as desktop environment object access protocol and as enterprise-level object access protocol of choice.

SOAP (Simple Object Access Protocol) and XML-RPC are both XML-based RPC protocols. Both normally use HTTP as a transport layer protocol, although SOAP can be adapted to use a multitude of common protocols, including FTP, SMTP, and Jabber. Both have data typing and fault handling properties. Because its simplicity, XML-RPC does not have an explicit interface definition language, instead the interfaces are defined by external means. SOAP, on the other hand, has WSDL (Web Service Description Language), which is analogous to CORBA's IDL. The WSDL files are generated either by hand, or from an already implemented interface. The client proxies can then be generated from the interface definition. Some SOAP implementations also allow dynamic method calling, so that the actual calls are generated on-the-fly from the WSDL definition. Despite of not having any interface definition language, XML-RPC interfaces are fairly simple to implement. SOAP interoperability, on the other hand, suffers from the complexity of specification, and getting different implementations to communicate reliably can be a difficult task. Both SOAP and XML-RPC are stateless protocols, and so they scale identically to HTTP servers. As they are based on XML, transmitting large amounts of binary data is very inefficient for both of them.

There is a separate MIME attachment specification for SOAP though, so that large binary objects such as audio recordings in Rring can be transmitted efficiently. SOAP has become a common protocol for many remote call tasks. For example, Microsoft .NET architecture uses SOAP as its remote call protocol. Also XML-RPC is widely used, especially in constructing distributed web applications. Many new high-level programming languages provide excellent XML-RPC interfaces.

Java RMI (Remote Method Invocation) is an RPC interface for the Java language. It is limited to Java language only, but the protocol is very efficient and easy to implement. The interfaces are defined as Java language interfaces with no further steps. The server and the client can be easily implemented using the defined interfaces. While very straightforward to use, its usefulness is limited because of dependency on Java. When creating interfaces dealing with custom hardware and systems such as IVR systems, and when the systems are spread across several different companies, such language assumptions usually cannot be made. Java RMI is extensively used in J2EE (Java2 Enterprise Edition) environments.

In Rring, raw HTTP interfaces were used so that the implementation could be kept simple and the different components easily deployed across different companies.

5.4.3 Interface between IVR system and processing backend

Communication between the IVR system and the processing backend is done using HTTP calls, which the IVR system makes. Parameters and data to be processed are transmitted using HTTP GET and POST methods, and success status is conveyed using HTTP status codes and supplementary messages.

There is support both for session-based requests, which support pre-listening of the ringing tones and only optional sending of ringing tones, and sessionless requests, which take care of both converting the ringing tone and sending it during a single request.

The following describes different CGI scripts that define the interface between the IVR system and the processing backend.

get_session.cgi

Purpose *get_session.cgi* begins a new session and returns a session cookie to the client.

Receives Nothing.

Returns Session cookie named “sesid”. The cookie does not have a defined lifetime, so it is supposed to last only that session.

Remarks Calling of *get_session.cgi* is not strictly necessary, but it ensures that the server is available at the beginning of the call.

pitch_detect.cgi

Purpose *pitch_detect.cgi* performs the pitch detection and creates a pre-listening sound clip.

Receives *pitch_detect.cgi* receives the following arguments sent via HTTP POST request:

sesid (Optional.) Session id given by *get_session.cgi*.

data PCM data as a file upload. The MIME type of the data should be either `audio/x-wav` or `audio/x-pcm`. In the first case, the file should be a monophonic wave file with sample rate of 8 kHz. In the latter case, the file should be raw A-law-compressed audio data with 8 kHz sample rate.

nosave (Optional.) If this parameter is set, saving of the recording in the backend is prevented even if it otherwise would have been done.

Returns

- Session cookie named "sesid".
- Output with the same MIME type as the input data. The output audio clip holds a pre-listening sample of the generated ringing tone.
- In case of an error, returns with HTTP Status 500 together with an associated error message.

Remarks If a session cookie is not given as an argument, one is generated in *get_session.cgi*.

send_sms.cgi

Purpose *send_sms.cgi* sends a previously generated ringing tone to the client via a SMS gateway.

Receives **sesid** Session id given by *get_session.cgi* or *pitch_detect.cgi*.

telnum The telephone number to which the ringing tone should be sent.

name (Optional.) The name of the ringing tone to be sent. If one is not given, a random name will be generated.

dummy (Optional.) If set, the ringing tone will not be sent actually, but instead an artificial delay will be inserted. This is useful for benchmarking the backend functionality.

Returns "OK" if sending succeeded. Otherwise returns HTTP Status 500 together with an error message.

end_session.cgi

Purpose *end_session.cgi* ends a session.

Receives Session id "sesid" given by *get_session.cgi* or *pitch_detect.cgi*.

Returns "OK" if sending succeeded. Otherwise returns HTTP Status 500 together with an error message.

Remarks Calling of *end_session.cgi* is not required, as sessions expire after a set time. It is provided as a courtesy method to clean the session immediately.

detect_and_send.cgi

Purpose Convert a recording to a ringing tone and send it in a single, sessionless request.

Receives telnum The telephone number to which the ringing tone should be sent.

name (Optional.) The name of the ringing tone to be sent. If one is not given, a random name will be generated.

dummy (Optional.) If set, the ringing tone will not be sent actually, but instead an artificial delay will be inserted. This is useful for benchmarking the backend functionality.

data PCM data as a file upload. The MIME type of the data should be either `audio/x-wav` or `audio/x-pcm`. In the first case, the file should be a monophonic wave file with sample rate of 8 kHz. In the latter case, the file should be raw A-law-compressed audio data with 8 kHz sample rate.

nosave (Optional.) If this parameter is set, saving of the recording in the backend is prevented even if it otherwise would have been done.

Returns "OK" if sending succeeded. Otherwise returns HTTP Status 500 together with an error message.

5.4.4 Interface between processing backend and SMS gateway

The interface between the processing backend and SMS gateway is also based on HTTP. The requests are sent as HTTP GET calls with parameters defining the data format, telephone number and header information. Since the interface is defined by the SMS gateway service provider, it is not described here in further detail.

5.4.5 Programming language considerations

Rring currently has been implemented using several different programming languages. In the early planning and prototyping phases, C was deemed to be only feasible language in the implementation of the digital signal processing functionality. Likewise, the auxiliary scripting, including different conversion tools and the IVR scripts themselves were programmed in Perl. Perl is flexible and rich language, and well suited for such purposes. Perl cgi-binaries can easily be executed within the Apache process using the mod_perl Apache module [Fou01].

It has been noted that there would be several benefits if the service was implemented with Java technologies. First, a Java implementation would be fairly platform-independent and easily portable to different IVR systems. Second, all of the application logic, including DSP algorithms, could be componentized and implemented in only one language. This would ease the maintenance of the software and encourage component and technology reuse. Third, installation and upgrades of Java Servlet packages is standardized and automated, further easing service maintenance. Fourth, Java Servlet and Java Servlet Page (JSP) platforms are well supported and deemed as standard platforms by several different value-added service providers, facilitating easier service integration.

Java platform, however, is not without its problems. The primary concern in this case is the execution performance. Traditionally programs written in Java have been deemed as being slow, because the program binaries have to be translated to native machine code prior to execution. While no tests regarding Java performance has been performed at Elmorex, several tests available on the web indicate that with modern Java just-in-time (JIT) compilers, execution speed of Java programs is comparable to programs written in C or C++ [Rij00, Shu01]. In some tests, the Java JIT compilers even outperformed some C compilers. Execution speed similarity also applied to different tests involving Fast Fourier Transform (FFT) calculation, such as SciMark benchmarks performed by Roldan Pozo [Poz00, PM00]. FFT benchmarks are particularly important regarding Rring, since the bulk of the calculation time is spent in FFT calculation. The importance of FFT in Rring is illustrated in program execution profile shown in table 4.1.

Converting the backend service to Java cannot be efficiently done in steps. The whole backend architecture has to be changed in one sweep to servlets, as stand alone Java programs executed as cgi-binaries are extremely inefficient. The process invocation overhead includes time spent in the just-in-time compiler, and can easily exceed 1.5 seconds for even relatively simple programs.

5.4.6 Implementation platforms

IVR systems are traditionally implemented with proprietary systems using specialized command languages. Re-implementation of applications using a system of different vendor is costly, effectively locking the service provider to a single service platform. The proprietary systems are generally built on Solaris, AIX or some other commercial Unix platform, and they utilize special hardware for telephone network connectivity. Lately also Linux and Windows are used as IVR platforms.

VoiceXML has become a promising new standard for IVR platforms. VoiceXML is a standardized description language for IVR services. It may be implemented on same platforms as traditional IVR systems, although usually it is provided as a specialized turn-key solution. A closer description of VoiceXML may be found a little further below.

The ringing tone conversion and SMS gateway modules are not as limited to any particular platform, since they are basically regular network servers. Therefore viable op-

erating systems include commercial Unixes, Linux and Windows. Commercial Unixes, such as Solaris, HP-UX and AIX are well supported, reliable and established operating systems with good scalability and proven track record. On the other hand, they (with exception to Solaris/x86) only run on proprietary hardware, require knowledge of that particular brand of Unix and are generally quite costly. Linux servers are now considered in practice as capable as the commercial Unixes, at least as reliable, and a lot more cost-effective than the commercial Unixes. Windows servers are tempting because of their deep desktop market penetration and apparent ease of administration. In practice, however, they require a lot more administration than the different Linux/Unix variants, are not as reliable, and require costly licenses.

Rring was developed on a Linux platform. While the ringing tone conversion module currently runs only on Linux, it may easily be ported to commercial Unixes. The IVR platform and the SMS gateway are currently handled by third parties, which use Sun Solaris and Linux operating systems, respectively.

5.4.7 VoiceXML

VoiceXML is an XML-based language designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed-initiative conversations [Voi00].

Voice applications are nowadays abundant, ranging from simple information retrieval systems, e.g. weather services, to automated mail order and ticket reservation systems or entertainment services such as ringing tone services. While widely used, the development systems are proprietary and re-implementation of applications using a system of different vendor is costly. Even worse, the interaction model of the service may not match the established commodity systems well.

The goal of VoiceXML is to make the well-established client-server architecture of WWW applications available to interactive voice response applications. This will make it easier to treat telephony applications as an alternative user interface for ordinary WWW or mobile services, or to use web application technologies in creation of IVR services. By defining the data and interaction with XML, standard tools can be used, reducing product development time and simplifying development.

The VoiceXML architecture is modeled to closely match that of a web service client-server model. See Figure 5.11 for an overview of the architectural model. A document server, e.g. a web server acts as an architectural backbone, handling requests from the VoiceXML interpreter and providing it documents. The VoiceXML interpreter handles the user inputs together with VoiceXML interpreter context. The VoiceXML interpreter parses the actual VoiceXML documents and the state information encoded within, while the VoiceXML interpreter context responds to certain system-wide predefined events, e.g. special escapes for reaching human operator, adjusting output volume or speech synthesis characteristics.

The VoiceXML interpreter controls the implementation platform together with VoiceXML interpreter context. The implementation platform is responsible for the actual hardware interaction, text-to-speech synthesis and speech recognition. The platform

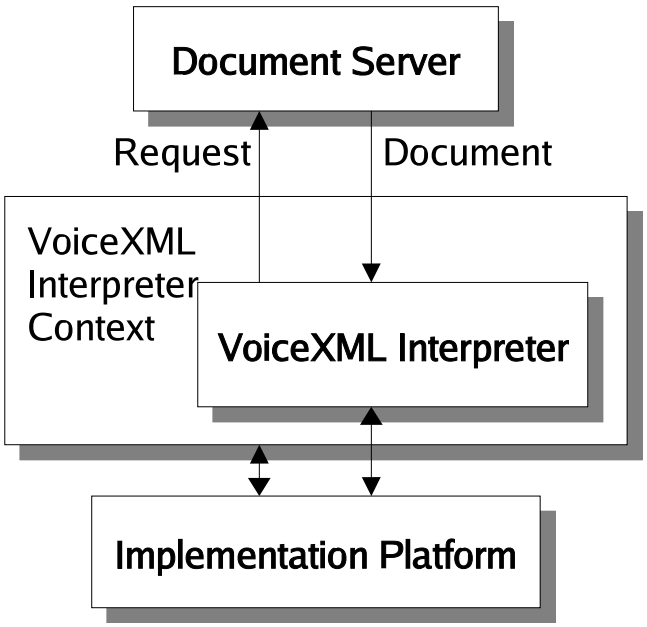


Figure 5.11: VoiceXML Architecture.

generates events according to user actions like spoken or DTMF input, or disconnect. The events are then handled as defined in the VoiceXML document by the VoiceXML interpreter, or if the interaction is not defined in the VoiceXML interpreter, by the VoiceXML interpreter context.

The main drawback of VoiceXML at the moment is its incompleteness. Many central aspects of the language, like the speech and DTMF grammars, are completely unspecified. Furthermore, specifications for such things as supported audio formats are quite lacking. The language also supports many different platform-specific extensions. The shortcomings may lead to a situation similar to different SQL dialects, in which the applications in principle are written in a common language, but the porting costs between different platforms are still considerable.

5.5 User interface issues

Usability of the service is important for the perception of service quality and for the overall image of the service. For the end-user, several criteria have to be fulfilled:

- The IVR service has to be easy to use.
- The service has to give always feedback.
- The response time has to be short.

The IVR service has to be designed so that the prompt speeches are short and clear, that the user will not have to make extra key-presses, and that the user may not get

lost in the service. Ideally the user only hears a short prompt speech, gets to record the message, and then finishes the recording by hanging up the phone. The ringing tone is then automatically sent to his mobile phone. That way, no user-interface interaction with the IVR service is required. Another possibility is to provide the user a choice to listen to the generated ringing tone and to decide whether it is sent to the mobile phone. However, this is not desirable for two reasons. First, it has been noted that the male singing voice often gets misinterpreted as DTMF signals, causing preliminary ending of the recordings. Second, even if the end-user has a flat-rate charge, the IVR system providers usually charge by service use time, so that the shorter the call is, the less the IVR service provider charges.

The customer always has to get feedback from the service. If no ringing tone is generated or if there are network problems, the customer has to be informed of this by some means. This may be a notification during the phone call, or a short message, depending on the IVR structure chosen.

The delay between recording and receiving the ringing tone has to be short, preferably less than 20 seconds. Fast operation improves the perception of quality of service, and encourages the customer or his friends to use the service again.

5.6 Development framework

5.6.1 Description of the pilot platform

The pilot system was constructed as a process sharing system, since it is a lot easier to implement than a FIFO system, especially when using a HTTP server to receive the requests. To construct a PS system, the HTTP server receiving the request can simply execute the programs handling the input data itself. The drawback is that the HTTP server process has to wait for the processing to complete, and that consumes extra memory.

An IVR platform has been implemented locally for development and demonstration purposes. In the beginning of the project, purchase of a commercial development platform was not deemed necessary or viable, so the development has occurred on a custom system built from commodity components and free software. The architecture of the system is shown in Figure 5.12.

The development platform is a generic x86 computer which is connected to an ISDN line with a passive ISDN card. The operating system used is Red Hat Linux 6.2 [red01].

The SMS sending facilities are provided by connecting a Nokia 6150 handset to the computer with a serial cable and by communicating with the handset via Gnokii software package [gno01]. Small changes to the Gnokii software were required in order to facilitate raw SMS sending.

The custom IVR service implementation has been done using Vgetty modem software package [Ebe98], which is controlled by a state machine implemented in Perl. The IVR implementation takes care of playing prompts back to the user, reading user input and recording the singing of user. Slight bug-fixes to Vgetty and to Linux kernel ISDN

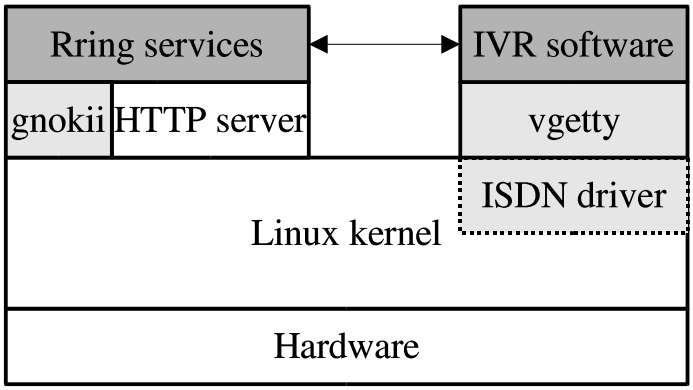


Figure 5.12: Rring development framework architecture. Light gray portions have been modified or enhanced during the project, dark gray portions have been implemented from scratch.

driver were required for reliable operation.

Vgetty could not recognize user hangups, instead it waited until timeout. In the development system this was considered highly undesirable, since the IVR would first wait until timeout of the recording, then play next prompt, then wait until timeout of the prompt answer, and finally exit the system. The whole process would take up to two minutes time, during which the line would return available instead of busy tone to the next caller. Vgetty was modified (and the patches sent back to Vgetty development team) so that it would recognize user hangups immediately and return the information back to the IVR software.

Linux ISDN driver often misinterpreted singing as dual tone modulated frequency (DTMF) signals induced by phone key-presses. The phenomenon occurred especially on male voices. There exists no simple fix for the problem, since it also occurs in commercial IVR platforms, but the problem was alleviated by implementing a detection buffer to Linux kernel ISDN driver, so that several consecutive positive detections are needed for triggering the DTMF detection routine.

The service architecture has been implemented in a model as compatible with VoiceXML as possible. Although the IVR functionality is defined in the IVR script, communication with different backend modules is performed solely by using a standard web server and services implemented on that server.

5.6.2 Testing methodology

Since a flawless operation of the RTC backend is critical for reliable service operation, a test bench was built for finding any deficiencies in the backend server operation.

The test bench program was designed to simulate the IVR service as closely as possible. It is run in another computer (so that the existence of the test program itself would not skew the test results), from which it sends service requests to the backend server by HTTP. It simulates multiple clients accessing an IVR service simultaneously, by

initiating calls with delays falling on an exponential distribution, as in a traffic theory Poisson model [Aal01]. The maximum number of simultaneous connections is limited to 30, as in an IVR system with ISDN PRI connection.

Each simulated call proceeds exactly like a real call would do. The list below outlines the steps taken by each simulated call.

1. A random real-life recording is selected from a pool. The pool consists of 379 different recordings, the length of which varies from one second to 30 seconds, the mean being approximately 10.5 seconds.
2. Simulation is paused for a duration of 15 seconds plus the length of the selected recording, to mimic the instruction prompt and the user recording.
3. The selected recording is sent to `detect_and_send.cgi`, which then transforms it to the event data by executing the pitch detection program at the server, and simulates sending the results to the SMS gateway by pausing for 2 seconds.

Each remote call made by the test bench together with its results are also logged in a log file, so that the results can be analyzed later. The log includes precise times of script executions, time taken by the script and information on whether the call succeeded or not.

The RTC backend was tested on two different computer configurations. The first one was a low-end computer with a 375 MHz Intel Celeron CPU and 128 megabytes of main memory, while the second one was a more modern 1.4 GHz AMD Athlon XP CPU and 512 megabytes of main memory. The computers were selected so that the meager performance of the Celeron system would induce any problems inherent in the design, while the Athlon system would give a reasonable insight of the performance of a modern production system.

Figures 5.13 and 5.14 show the backend processing times during the load tests. The processing times have been recorded with one-second granularity, which explains the visible step size in the graphs. Especially in the case of the Celeron CPU it can be seen clearly that request bursts cause strong peaks in processing times, which last long after the recordings have been processed. While average processing times are approximately 10 seconds, the highest peaks reach several times higher, and in Figure 5.13 c) even to 180 s, at which point the client timeouts. Once the load reaches levels where timeouts occur, the server stays congested for several hundred seconds, and during that time only very few processing requests get through.

By comparing Figures 5.13 and 5.14, it can be seen that the great amount of memory of the tested Athlon configuration allows the computer to recover immediately from the momentary high-traffic situations. Even in Figure 5.14 d), in which the mean delay between calls is only 2 s, the longest processing times only slightly excess 50 s, and the performance recovers quickly from the congestion. This may be accounted to the generous amount of main memory of the computer–swapping never occurs, even though the number of processes is nearly 20. However, on these load levels processing times already have increased enough to be unacceptable with regard to service quality.

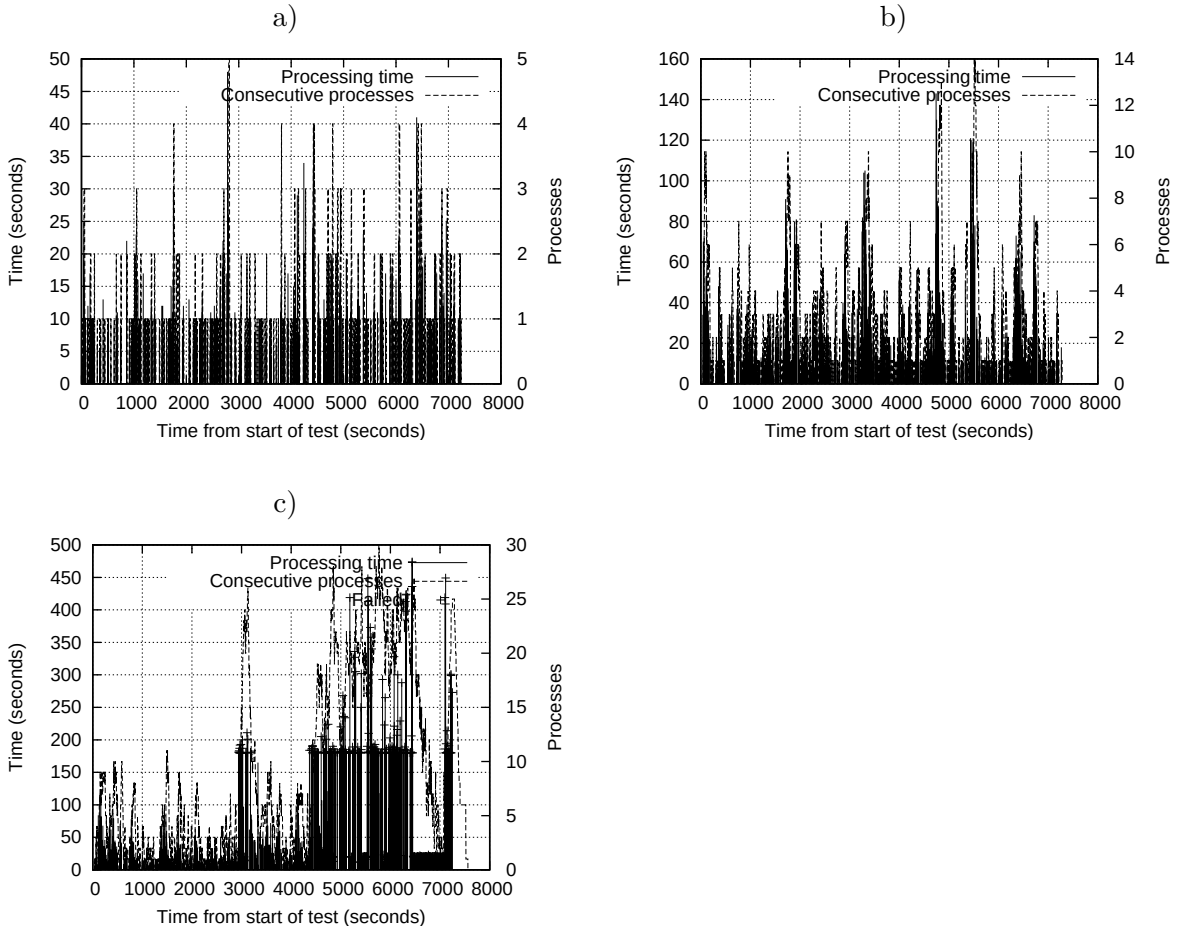


Figure 5.13: Sample processing times in test runs on a Celeron CPU. Figure a) represents a mean delay of 20 seconds between calls (traffic intensity of 1.3 Erl at the IVR), Figure b) represents a mean delay of 10 s (traffic intensity 2.6 Erl), Figure c) represents a mean delay of 7 s (traffic intensity 3.6 Erl).

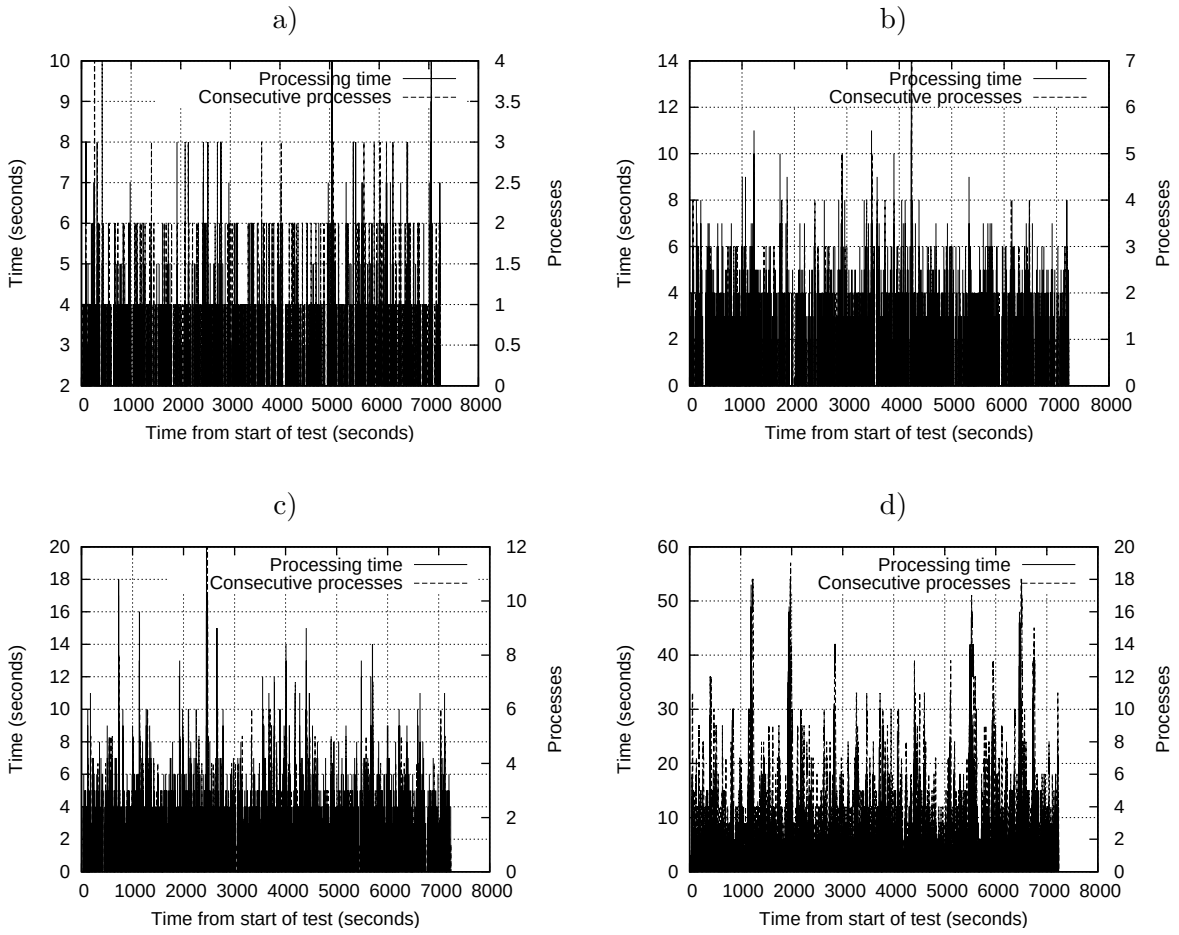


Figure 5.14: Sample processing times in test runs on a Athlon XP CPU. Figure a) represents a mean delay of 10 seconds between calls (traffic intensity of 2.6 Erl at the IVR), Figure b) represents a mean delay of 7 s (traffic intensity 3.6 Erl), Figure c) represents a mean delay of 4 s (traffic intensity 6.4 Erl), and Figure d) represents a mean delay of 2 s (traffic intensity 12.8 Erl).

While the tests clearly confirmed the simulation results in section 5.3.2 and further emphasize the need for serializing the backend processing, they also showed that given enough processing power and main memory, a modern computer can acceptably handle light and moderate loads even with parallelized backend processing.

Chapter 6

Future technologies

6.1 Ringing tone formats

So far practically all ringing tone providers have been offering only Nokia-compatible ringing tones. Nokia supports ringing tones in their Smart Messaging Specification, introduced already in 1997. Smart messaging not only defines sending of ringing tones, but e.g. sending of business cards, calendar items, dynamic menu items, Internet access configuration data, and so on. Currently all Nokia phones on the market support Smart Messaging. Nokia has opened Smart Messaging Specification for royalty-free licensing in December 2000, although so far Samsung is the only major mobile phone manufacturer to have licensed Smart Messaging technologies [Nok00, Nok01].

Multimedia Messaging Services (MMS) is the next-generation messaging format standardized by 3rd Generation Partnership Project (3GPP) [Nok02]. It is supported by every major mobile manufacturer, and also the operators expect it to be a new revenue source, so it is expected to gain rapid market acceptance. The first MMS-enabled phone, Ericsson T68, was released in Q1 2002, and the actual breakthrough is expected to happen in 2003 or 2004 [Mob02]. MMS supports several different message formats, including text-based messages, picture messaging, image messaging, and even audio and video messages. The standard is not bound to any specific carrier methods, allowing both GSM and UMTS as well as CDMA networks to be used. On a GSM network, the messages are sent as a combination of a SMS notification and WAP over GPRS transmission [Nok02].

6.2 Polyphonic ringing tones

As the demand for phone personalization features has increased, handset manufacturers have been adopting support for polyphonic ringing tones. In Japan, i-Mode devices generally support polyphonic ringing tones, although different phone manufacturers have their own ringing tone formats [KPN02].

In Europe, adoption of polyphonic ringing tone technologies has been slower than in Asia Pacific. However, several mobile phone manufacturers have introduced models supporting polyphonic ringing tones in 2002. Phones made by Nokia and Sony Ericsson

have support for ringing tones in MIDI format [MID96]. On Nokia phones, the ringing tones may be transmitted either using SMS Smart messaging [Nok99] or WAP. Sony Ericsson phones support WAP on-the-air transmission only.

6.3 Pitch detection based mobile entertainment services

Of course, ringing tone services are not the only possible mobile services to utilize pitch detection technologies. Several other concepts have been depicted at Elmorex. Pitch detection may easily be adapted to creation of melody messages. One could compose a message consisting of a personalized greeting melody together with a text passage. Such service could work for example so that the user calls a number and sings a melody, after which he or she sends an SMS message to a SMS service number. The service then combines these separate messages to a single multimedia message, which is sent to the intended receiver.

Even more intriguing possibilities arise in real-time pitch detection applications. One could for example have a real-time melody karaoke in an Internet-based virtual chat-room like Habbo Hotel [Hab01]. In such a service, one would queue up for a performance in the virtual room. Then, on his/her turn, the user would call a service number and make a singing performance. The result could be then streamed to all listeners in real-time using MIDI or some other high-level music description language. So, the performer could sing in relative anonymity, while the audience would still get a reasonable image of the performer's skills. Other real-time applications proposed at Elmorex are for example different interactive entertainment applications for Digital TV. There, people would call service numbers, perform on-line and see the results directly on the TV screen. It is also possible to create applications utilizing no other devices than regular telephone or mobile handsets. One such service would be automatic accompaniment generator for singing. The customer would call the service, then defining a phone number to be called. After the connection is made, one could sing a song to the recipient, while the accompaniment generator created accompaniment to the singing. Such services could easily be marketed for Christmas, valentines or birthday greetings.

6.4 Operating systems for mobile client-based sound applications

6.4.1 Java software environment

Java2 Micro Edition (J2ME) is a version of Java designed for consumer devices, such as smart cards, pagers, mobile handsets, or even set-top boxes. The specification is currently at version 1.0a, which was released on December 15th, 2000.

For each J2ME device group, there exists a special API for generating applications in that genre. The mobile API is dubbed Mobile Information Device Profile (MIDP). It does provide rudimentary means for networking, persistent data storage and user interface API, but sound support is limited to playing one of a few pre-defined alert

sounds. Therefore development of sound-based entertainment software or services is not possible on J2ME.

As of October 2002, Java-enabled mobile devices are already widely available in Japan and Europe.

PalmOS, the operating system of Palm handheld devices, has had support for MIDI sounds since version 3.0. However, at least the older devices have been very limited in sound support, featuring only a simple piezo-electric buzzer, so only simple sound applications may be implemented.

PalmOS version 5 is the port of PalmOS from the previous Motorola Thunderball platform to the ARM processor platform. Simultaneously with the platform change, a new sampled sound API has been included. This includes support for playback and recording of 16-bit sampled stereo and mono sounds in different formats [Pal02]. While the connectivity of Palm devices won't be on the same level as that of most mobile phones, these inclusions still make Palm a viable platform for client-based mobile sound applications.

Symbian's EPOC, which is the operating system of most new advanced mobile phone devices, does have support for recording and playing sampled audio clips. It also has a telephony API, which provides application access to the phone's telephony functionality, and a framework for writing drivers for particular telephony hardware [Sym02].

The most advanced Windows CE devices, namely Compaq iPaq, have fairly advanced audio capabilities. They support stereophonic playback through headphone speakers, as well as audio recording with an integrated microphone. With their fast StrongARM processors they have enough processing power for MPEG 1 Layer 3 and even compressed video playback.

There already exists a multitude of music playback and generation software for the iPaq. The main problem with Windows CE based products is their weak mobile connectivity. None of them are currently integrated with mobile handsets, and while for example iPaq can utilize wireless local area network (WLAN) cards, they do not offer communication capabilities similar to for example already existing Nokia Communicator devices. So, although the audio technology of Windows CE devices is most mature of all, it is not very attractive as platform for mobile musical entertainment services, until their connectivity is enhanced.

Chapter 7

Discussion and conclusions

7.1 Ringing tone quality

Systematic interviews would have given valuable insight of the perceived quality and performance of the system, but such effort could not be feasibly done in the time frame and scope of this project. However, lots of informal feedback was gathered by discussing with testers and following the usage patterns and listening to the recordings and the created ringing tones.

Unfortunately the use experience of the pilot system was skewed because of a bug which caused full-length ringing tones to not get transmitted correctly. This bug was corrected only late in the pilot phase, and so no conclusive results from the pilot period could be obtained.

Good singers were often able to create ringing tones of very good quality, although even then there would almost always be some impurities in the ringing tone. The impurities would range from missing short notes to a few half-note errors and short high-pitched notes. However, often the users sang very badly, and in these cases the ringing tones were completely unrecognizable, more or less random note sequences. This seemed to often come as a surprise to the users, who might have expected to receive a corrected version of their performance.

One observation was that the users with little musical skills often intentionally spoke or just made noises. While ringing tones created this way were not melodic, they still sounded very unique and were suitable as alarm ringing tones.

Most of the test users were pleased with the service. It was considered to be a novel concept, and while the created ringing tones were not perfect, their quality was considered acceptable. It was also noted, maybe not surprisingly, that the users with at least some musical background, be that choir singing or piano lessons in the childhood, were most eager to use the service, while the ones with no musical background mostly had a neutral attitude towards it. Also, professional musicians often had negative attitudes towards the service, not tolerating the errors and impurities in the produced ringing tones.

The ringing tone quality is quite dependent on the type of the input recordings. Monophonic recordings with a moderate tempo and long, clear notes gave the best results.

Short adjacent notes tend to get bound together, but there seemed not to be any simple way to get rid of that behavior. Also background noise generated extra notes in the ringing tones.

It was also noted that the GSM transmission path frequently causes different clicks and pauses to the sound. While they are short enough to not degrade legibility of speech, they do propagate to generated ringing tones. Since the user is not aware of transmission path artifacts, the service itself is blamed for these artifacts. Unfortunately there are no solutions to this when using the IVR approach. If the transmission path was lossless, the problem would be automatically solved. One way to achieve this would be to use recorded messages and send them via MMS.

7.2 Contributions made by the author

The initial idea for this project came from Tero Tolonen, who was employed at Elmorex Ltd. at that time. He also developed the initial version of the f_0 detection and event creation routines. Many fundamental design choices, such as the use of fast autocorrelation and the bottom-up event creation process were made at that time by him. All other parts of the system were developed by the author. Also the fundamental frequency detection and event creation routines, except the autocorrelation and associated FFT routines were extensively refined by the author, so that the routines described in this thesis are the handwriting of the author.

During the pilot phase of the project, the interactive voice response platform and the SMS gateway were provided by third parties, and so only the interfaces to these systems were designed by the author.

7.3 Evaluation of project success

The objective of this thesis was to develop a ringing tone generation service, in which the user can sing or hum himself a new ringing tone. This objective was fulfilled. A functional service was created, first as a development system, and then as an operational commercial pilot service. While the quality of the created ringing tones is not perfect, it is considered adequate for a commercial service.

Although the proposed architecture changes proposed in section 5.4 were not implemented, the original parallel processing architecture was proved viable in light use. During a period of several months, there were no performance problems. Even though the arrival of the ringing tones was intermittently delayed, it turned out that the greatest delays were caused by the SMS centre, and were thus unavoidable.

The implementation of the system might be cleaner if done using the Java language and application platforms, as suggested in Section 5.4.5.

Given the market conditions outlined in Chapter 1, a commercial service would probably be viable still with the quality limitations described in this Chapter. Marketing of the service would just have to be adjusted accordingly. Given the current state of

the application, it would take only little work to turn it into a fully-fledged commercial service.

Bibliography

[Aal01] Samuli Aalto. S-38.145 introduction to teletraffic theory. Helsinki University of Technology Course Material, January 2001.

[Alk99] Paavo Alku. S-89.126 puheensiirtotekniikka (speech transmission technology). Helsinki University of Technology course material, 1999.

[BEK+00] Don Box, David Ehnebuske, Gobal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol 1.1. Web page, May 2000. http://www.w3.org/-TR/SOAP/.

[BMS00] Juan Pablo Bello, Giuliano Monti, and Mark Sandler. Techniques for automatic music transcription. In *International Symposium on Music Information Retrieval*, Plymouth, MA, October 2000.

[BP89] Judy Brown and M. Puckette. Calculation of a narrowed autocorrelation function. *Journal of Acoustical Society of America*, 85:1595–1601, 1989.

[Bre90] Albert Bregman. *Auditory Scene Analysis*. MIT Press, Cambridge, MA, 1990.

[Bro91] Judy Brown. Musical frequency tracking using the methods of conventional and narrowed autocorrelation. *Journal of Acoustical Society of America*, 89:2346–2354, 1991.

[Dav52] W. B. Davenport, Jr. An experimental study of speech-wave probability distributions. *Journal of the Acoustical Society of America*, 24:390–399, 1952.

[DFAB98] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human Computer Interaction*. Prentice Hall Europe, 1998.

[DR91] Boris Doval and Xavier Rodet. Estimation of fundamental frequency of musical sound signals. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1991.

[Ebe98] Marc Eberhard. Vgetty documentation center. Web page, 1998. http://alpha.greenie.net/vgetty/.

[Ell95] Daniel Ellis. Mid-level representations for computational auditory scene analysis. In *Proceedings of the International Joint Conference on AI, Workshop on Computational Auditory Scene Analysis*, August 1995.

[Ell96] Daniel P. W. Ellis. *Prediction-driven computational auditory scene analysis.* PhD thesis, Massachusetts Institute of Technology, 1996.

[Fla65] James L. Flanagan. *Speech Analysis, Synthesis and Perception.* Springer-Verlag, 1965.

[Fou01] The Apache Software Foundation. Apache/perl integration project. Web page, July 2001. http://perl.apache.org/.

[gno01] Gnokii homepage. Web page, April 2001. http://www.gnokii.org/.

[Hab01] Habbo Hotel. Web page, May 2001. http://www.habbohotel.com/.

[HAHN93] Risto Hämeen-Anttila, Pertti Hölttä, and Seppo Niinioja. *Tietoliikennejärjestelmät.* Painatuskeskus, 1993.

[Har78] Fredric J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, January 1978.

[HP93] Peter G. Harrison and Naresh Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Longman Group, January 1993.

[itu96] International Telecommunication Union. *Methods for subjective determination of transmission quality. Recommendation P.800*, August 1996.

[Jav02] Java remote method invocation. Web page, October 2002. http://java.sun.com/products/jdk/rmi/.

[Jeh97] Tristan Jehan. Musical signal parameter estimation. Master's thesis, CNMAT, University of California, Berkeley, 1997.

[Kla97] Anssi Klapuri. Automatic transcription of music. Master's thesis, Tampere University of Technology, 1997.

[KPN02] How to create an i-mode site. PDF Document, July 2002. http://www.imode.nl/.

[MID96] MIDI Manufacturers Association. *The Complete MIDI 1.0 Detailed Specification*, 1996.

[Min01] Tekstiviestimarkkinat 1999-2002. PDF publication, April 2001. http://www.mintc.fi/www/sivut/dokumentit/julkaisu/julkaisusarja/2001/a20.pdf.

[Min02] Tekstiviestimarkkinat 2000-2003. PDF publication, April 2002. http://www.mintc.fi/www/sivut/dokumentit/julkaisu/julkaisusarja/2002/a192002.pdf.

[Mob02] Mobilemms.com. Web page, October 2002. http://www.mobilemms.com/.

[Nok99] Nokia Mobile Phones Ltd. *Smart Messaging Specification, version 2.0*, 1999.

[Nok00] Nokia opens the specification for ring tones and mobile phone logos for free licensing. Press release, December 2000.

[Nok01] Samsung and Nokia announce licensing agreement on mobile browser and smart messaging technology. Press release, November 2001.

[Nok02] How to create MMS services. PDF Document, September 2002. http://www.forum.nokia.com/ndsCookieBuilder?fileParamID=2411.

[Obj02] Unified modeling language. Web page, October 2002. http://www.omg.org/uml/.

[OMG02] CORBA FAQ. Web page, October 2002. http://www.omg.org/getting-started/corbafaq.htm.

[Owe82] Frank Owen. *PCM and Digital Transmission Systems*. McGraw-Hill, 1982.

[Pal02] PalmOS 5 overview. Web page, October 2002. http://www.palmos.com/-dev/support/docs/palmos5/os5overview.html.

[PB52] G. E. Peterson and H. L. Barney. Control methods used in a study of the vowels. *Journal of Acoustical Society of America*, 24:175–184, 1952.

[PM00] Roldan Pozo and Bruce Miller. SciMark 2.0. Web page, 2000. http://math.nist.gov/scimark2/.

[Poz00] Roldan Pozo. Java performance analysis for scientific computing. seminar presentation slides, November 2000. http://www.ukhec.ac.uk/events/java-hec/pozo.pdf.

[Rab77] L. R. Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 25(1):24–33, 1977.

[RCRM76] Rabiner, Cheng, Rosenberg, and McGonegal. A comparative performance study of several pitch detection algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-24(5), October 1976.

[red01] Red Hat Inc. Web page, 2001. http://www.redhat.com/.

[Rij00] Chris Rijk. Binaries vs byte-codes. Web page, June 2000. http://www.aces-hardware.com/Spades/read.php?article_id=153.

[Roa98] Curtis Roads. *The computer music tutorial*. The MIT Press, 1998.

[Ros90] Thomas D. Rossing. *The Science of Sound*. Addison-Wesley, 1990.

[Sch96] Eric Scheirer. Bregman's chimerae: Music perception as auditory scene analysis. In *Proceedings of 1996 International Conference on Music Perception and Cognition*, 1996.

[Sch01] Henning Schulzrinne. Advanced internet services, audio compression. Lecture slides, October 2001.

[Shu01] Kazuyuki Shudo. Performance comparison of JITs. Web page, February 2001. http://www.shudo.net/jit/perf/.

[Son68] M. M. Sondhi. New methods of pitch extraction. *IEEE Transactions on Audio and Electroacoustics*, pages 262–266, June 1968.

[Sym02] *Symbian Development Library v7*, July 2002.

[Tex86] Texas Instruments. *TCM29C13 Data Sheet*, April 1986.

[Tol00] Tero Tolonen. *Object-based sound source modeling*. PhD thesis, Helsinki University of Technology, 2000.

[Voi00] VoiceXML forum home page. Web page, October 2000. http://www.voicexml.org/.

[XML02] XML-RPC home page. Web page, October 2002. http://www.xmlrpc.com.