TR-IT-0331

Cellular Phone Based ATR-MATRIX

グルーン ライナー
Rainer Gruhn

シンガー ハラルド
Harald Singer

塚田 元
Hajime Tsukada

内藤 正樹
Masaki Naito

中村 篤
Atsushi Nakamura

February 16, 2000

We describe the cellular phone based ATR-MATRIX. We focus on the problem how to reuse high quality speech databases to build a speech recognizer for low-bandwidth speech. This report is accessible for ITL members via /home/rgruhn/paper/TR-IT-PHS/.

Contents

1 **Introduction** 1

2 **Acoustic Modelling** 2
- 2.1 Training and Tuning ATRSPREC for English Cellular Speech . . . 2
  - (2.1.1) Baseline Training and Evaluation . . . 2
- 2.2 Training and Tuning ATRSPREC for Japanese Cellular Speech . . . 4
  - (2.2.1) Database Description . . . 4
  - (2.2.2) Training, Adaptation and Evaluation . . . 4
  - (2.2.3) Discussion of Results . . . 4
- 2.3 Speech Piping Experiments . . . 5
  - (2.3.1) Preparations . . . 5
  - (2.3.2) Piping . . . 6
  - (2.3.3) Data Refining . . . 6

3 **System Setup** 9

4 **Remaining Problems** 10
- 4.1 Environmental Noise . . . 10
- 4.2 Dropout . . . 10
- 4.3 Design of Human Interface . . . 10
- 4.4 Applicational Problems . . . 11

5 **Conclusion** 12

References 15

A **Telephone Filter** 17

B **Example Configuration File** 18

C **Automatic Cellular-Phone Answering System Manual** 20

D **Script for Concatenation and Splitting of Wavefiles** 21

E **Creating the Marking Signal** 23

1 Introduction

For more than a decade, ATR has been carrying out research in the field of speech-to-speech translation technology. Our most recent research prototype system, ATR-MATRIX, can translate Japanese speech into English, German, Korean and Mandarin, and English speech into Japanese [Takezawa et al., 1998].

Two key issues should especially be considered when we aim at a system which can be used in the real world. One is to make the system more robust against the variablity in human speech, environmental non-speech sounds and so on, that are typical problems in the case of using a spoken language system in real environment. The other issue is to make the system easily accessible from anywhere. Since conventional speech-to-speech translation systems, including ATR-MATRIX, have been available only on expensive and heavy high-end computers, only a limited number of people can use the system at limited places. One solution to this is to down-size the whole system by "pruning" the speech translation software, and by further progress in the integration technology of processor and memory devices. The other solution is to split the system into two parts, a computationally expensive part running on server computers in the network and a (minimal) speech input function part running on the user's laptop or hand-held terminal. This architectural solution allows the user's cost to be minimized without sacrificing performance compared to the standard ATR-MATRIX.

Several implementations to split the system are possible. In [Singer et al., 1999b], we proposed a design in which feature extraction for speech recognition and wave unit concatenation for speech synthesis are running on the user's laptop. In this paper, we discuss a design in which the user only needs a cellular-phone to access the speech translation system. In Japan, cellular-phones are already used by more than forty million people and they have a very vast access area. A system based on the cellular network enables the speech-to-speech translation system to be easily accessible anywhere. A hybrid of both approaches, i.e. a system using both laptop client and cellular phone, was demonstrated in [Singer et al., 1999a]

This technical report describes the setup of our cellular-phone based prototype, a procedure for training/adaptation of acoustic models for cellular phone speech by re-using previously collected databases, and remaining major problems in this design towards a real-world speech translation service.

- Section 2 describes our experiments to build a cellular phone acoustic model without sufficient cellular phone quality speech,
- Section 3 explains the technical setup for experiments,
- Section 4 lists the remaining problems and highlights future work items.
- The Appendix contains python scripts, configuration files and manuals for software made in this project. Details about the telephone filter are also given.

2 Acoustic Modelling

For any new acoustic condition and task domain, matched data has to be collected to train or adapt the statistical acoustic and language models. At ATR, we have been collecting huge databases for high-quality speech [Nakamura et al., 1996, Matsui et al., 1999]. However, we had no access to cellular-phone and not even telephone speech databases in similar task domains.

We thus tried out several methods to reuse our wide-bandwidth databases:

1. downsample to 8kHz and apply telephone FIR filter (gained from an ITU data sheet)

2. pipe speech data through a speaker – cellular-phone connection and re-record, similar to NTIMIT [Jankowski et al., 1990] and CTIMIT [Brown and George, 1995] as shown in Figure 3 in section 2.3 .

2.1 Training and Tuning ATRSPREC for English Cellular Speech

(2.1.1) Baseline Training and Evaluation

As outlined above, the fundamental problem we face is the insufficient amount of training data for sufficiently close conditions. An open research issue is a quantification of the terms "sufficient".

In fact, we collected a small cellular-phone database for 4 speakers (3 Germans, 1 British) consisting of 48 phonetically-balanced short sentences and 25 sentences of 16 digits. This amounts to about 500 seconds of speech (or about 350 seconds without initial and final 1 second of non-speech) per speaker. This is not enough for training a speaker-dependent cellular acoustic model (AM).

Our baseline cellular acoustic model is trained as outlined in Figure 1.

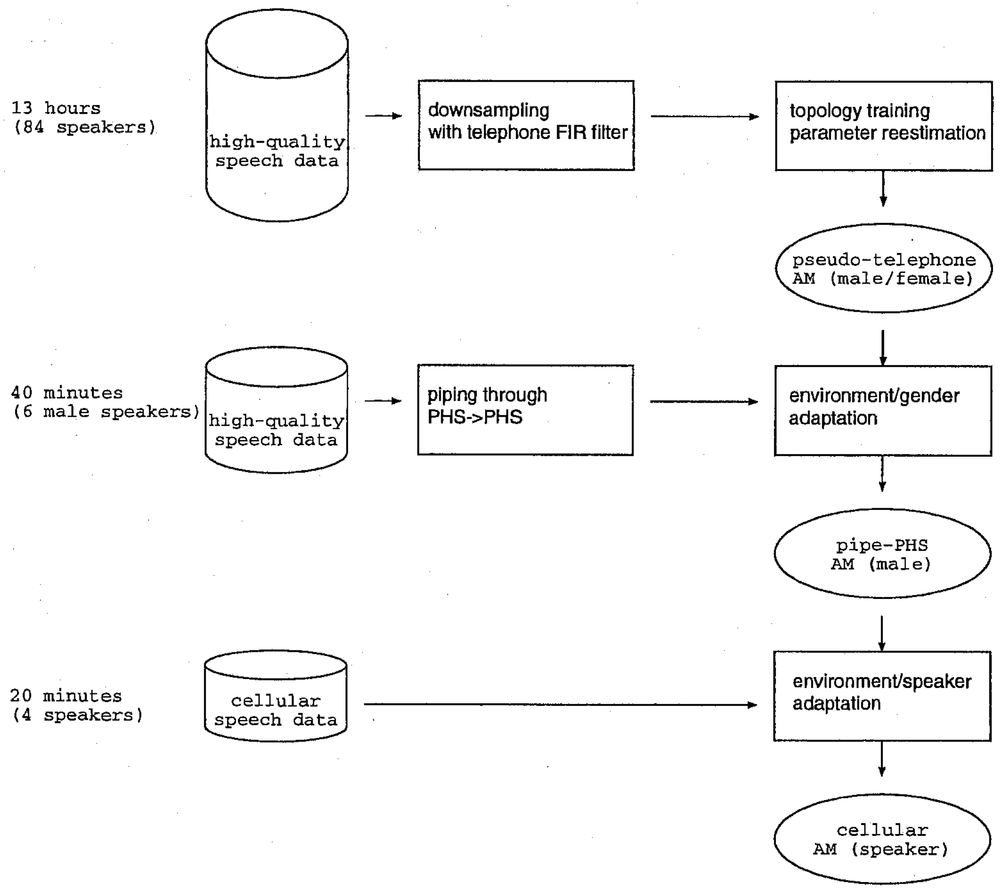

Figure 1: Baseline training for English Cellular speech

step 1

About 13 hours of speech by 84 male and female speakers (WSJ0) were were downsampled from 16 kHz to 8 kHz, additionally applying a telephone speech bandwidth FIR filter [Woudenberg, 1994] (see Appendix A). A 1000 state speaker-independent topology was trained and then gender dependent models were retrained using 5 Gaussian mixtures per state, first using phoneme time information and then with embedded reestimation (6.12 hours of speech by 42 male speakers, 6.10 hours by 42 female speakers). Finally, a gender independent 3 state, 10 mixture non-speech model was added.

step 2

Previously, we had recorded read-speech travel arrangement conversations, digit strings and phoneme-balanced sentences for 6 male speakers (1 American, 1 Australian, 4 Germans), a total of 2488 seconds (not considering non-silence segments). This data was piped through a loudspeaker and from there re-recorded with a PHS mobile phone as shown in Figure 3. Alignment between the orginal concatenated speech file and the re-recorded PHS speech file was hand-checked and we found only a total drift of about 200 msec/hour, which was fairly linear, i.e. after a linear compensation we only had a variation of less than 5 msec absolute.

Using this re-recorded data, MAP-VFS adaptation was done. Preliminary tests with real PHS speech data was satisfactory, but the performance for cellular speech data was not good. One of the reasons was a mismatch in the cepstral normalization, the other problem was a mismatch for the non-speech model.

step 3

We performed a rough alignment of the small cellular database and calculated a cepstral mean and retrained the 3 state, 10 mixture non-speech model. With this combined model we then aligned the cellular speech database to find pronunciation variants, especially insertions of non-speech into the digit strings. Silence segments shorter than 150 milliseconds were judged as wrongly labeled and those silence labels were discarded.

step 4

Based on this alignment, a final speaker speaker/environment adaptation was performed for each speaker.

Evaluation was performed for two sets: a typical scenario for information (TOS00001) like "when does the lecture start?" and a set of useful expressions (TZS00004) like "please repeat that."

Table 1: Baseline results for cellular English speech (word accuracy in % / real-time factor on Linux PII-300).

speaker/set	TOS00001 (26)	TZS00004 (25)	all (51)
M001	90.06 / 1.17	91.01 / 1.48	90.38 / 1.31
M006	97.66 / 1.33	85.39 / 1.03	93.46 / 1.19
M008	86.55 / 1.44	91.01 / 1.26	88.08 / 1.36
M009	93.57 / 1.08	86.52 / 0.95	91.15 / 1.02
average	91.96 / 1.25	88.48 / 1.17	90.77 / 1.22

2.2 Training and Tuning ATRSPREC for Japanese Cellular Speech

(2.2.1) Database Description

set1 : 166 male and 241 female speakers, each spontaneously uttering one conversation side in the travel arrangement task, a total of 6432 utterances, recorded at 16 kHz.

set2 : 10 male speakers reading 50 phoneme-balanced sentences each, collected simultaneously with 2 channels, one at 16 kHz with a Sennheiser headset and one at 8 kHz through a CellularPHS connection.

set3 : 5 male speakers reading 48 travel arrangement sentences each, collected similar to set2.

We also recorded **set4**, similar to **set3**, except that we used CellularCellular connection. The high-quality recording of **set2** was then piped through a CellularPHS (CellularCellular) connection, varying positions between playback speaker and cellular-phone microphone.

(2.2.2) Training, Adaptation and Evaluation

Set1 was downsampled to 8 kHz and and the FIR filter applied. The resulting data was used to train a gender-independent state-sharing HMM topology with ML-SSS[Ostendorf and Singer, 1997] for a 1000 state model. Gender-dependent models were then retrained using Baum-Welch algorithm and the number of mixtures at each state increased to 5 (Baseline AM).

We then performed MAP-VFS environment adaptation [Tonomura et al., 1995] on the Baseline AM using channel 2 of **set2** (trueCellularPHS AM). Alternatively, piped data of **set2** was also used with MAP-VFS (1cmCellularPHS and 20cmCellularPHS AM).

Recognition was performed with a 2-pass decoder with a vocabulary size of about 13,500 words using a multi-class composite n-gram[Yamamoto and Sagisaka, 1999]. For comparison, we also used a phoneme bigram trained on the 407 training conversations.

Table 2: Accuracy (%) and rtf (real-time factor for a PentiumII-300MHz) for acoustic models adapted with different data.

evaluation speech	model type	word accuracy	phoneme accuracy	rtf
Wideband	Wideband	89.0	79.3	0.45
CellularPHS	Baseline	87.3	69.1	0.85
	trueCellularPHS	88.8	71.1	0.78
	1cmCellularPHS	89.3	69.7	0.83
	20cmCellularPHS	84.1	66.0	1.02
CellularCellular	Baseline	83.1	68.8	1.50
	1cmCellularCellular	89.8	74.8	1.03

(2.2.3) Discussion of Results

The most surpring result was, that wideband speech does not outperform the cellular speech (89% vs. 89.3%). For cellular-phone speech, we had a considerable drop in phoneme accuracy, from 79.3% to about 70%, but the word accuracy remained at about the same level or even rose slightly. From this we can see the strong influence of the language model on recognition accuracy. The strongest impact the lower data quality has is on the real-time factor, which doubles compared to wideband speech.

The difference of CellularPHS and CellularCellular speech becomes visible in the performance of the Baseline acoustic model, which performed rather poor on CellularCellular speech (83.1%). Piping speech proved to be a cheap and fast way to generate data for an acoustic model that works very well in a new scenario. An important issue in piping is to set up the experiment properly, e.g. a long distance between speaker and keitai impairs the performance.

A different approach that we tried was piping several time-stretched pulse signals (TSP) and then filter the wide-bandwidth database with the averaged impulse response. Accuracy for acoustic models trained with that method was less than 50%, i.e. a huge mismatch.

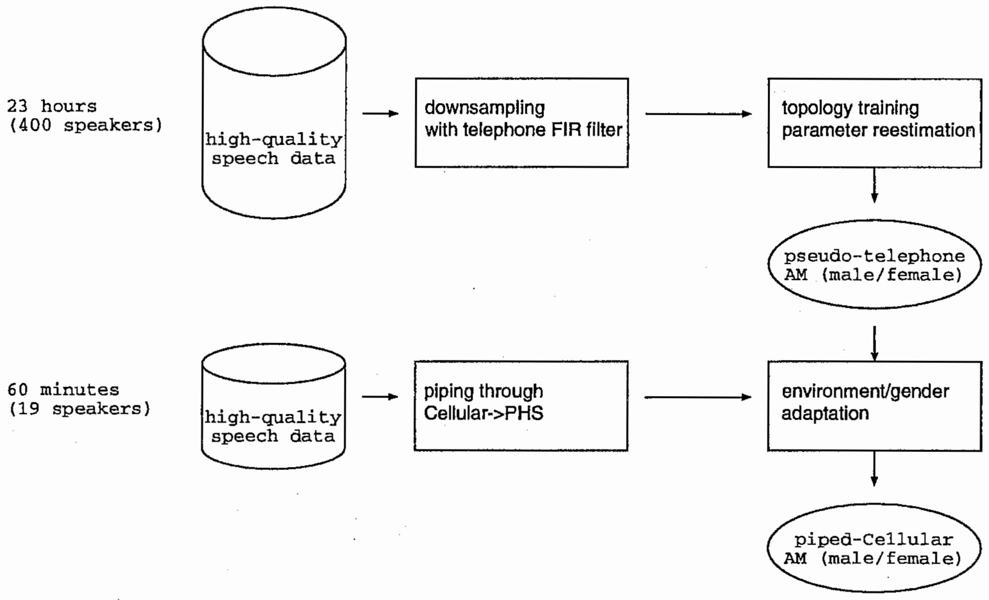
23 hours
(400 speakers)
high-quality speech data
downsampling with telephone FIR filter
topology training parameter reestimation
pseudo-telephone AM (male/female)
60 minutes
(19 speakers)
high-quality speech data
piping through Cellular->PHS
environment/gender adaptation
piped-Cellular AM (male/female)

Figure 2: Baseline training for Japanese Cellular speech

2.3 Speech Piping Experiments

To generate cellular phone quality speech using a high quality speech database, speech piping experiments were carried out. This chapter provides a comprehensive manual on how the experiments were made.

The same data generation method has also been used to make an acoustic model for a Toshiba Libretto laptop audio device as input system to ATR-MATRIX.

The experiment can be divided in four steps:

- preparation
- piping
- cutting of recorded speech file
- acoustic model training and evaluation

(2.3.1) Preparations

Preparations include two parts, the hardwaresetup and the speech database generation.

Figure 3 shows a possible experiment setup. A different idea would be to use a USB audio device on the right side in Figure 3 instead of the DAT.

Problems that occured during the speech conversion include:

- finding a way to place speaker and cellular phone so that the acoustic properties are close to those of a person speaking with someone through a cellular phone. The evaluation will later show that the cellular phone should be very close, e.g. in 1 cm distance, from the speaker.
- the transmitter power of a cellular phone is strong enough to induce disturbances in amplifiers which can be heard as noise in the speaker. A solution for this problem was to put the amplifier in a distant place from the speaker and the cellular phone. The noise induced by PHS is much smaller than the noise caused by a keitai.
- background noise such as the sound of the room's air condition is mixed into the speech.
- cellular phones cannot be used everywhere. PHS and keitai cellular phone systems have different access areas, i.e. at a place where a keitai cannot make a connection a PHS may work properly.

Speech databases consist of many single wave files. To ensure they are played continuously without unpredictable delays e.g. due to network problems, the files are concatenated and copied to a local disk. As later the resulting database must be split up precisely enough to be able to use existing phonetic labels, ways of marking borders between speech signals are needed. For marking, we chose a file consisting of 250 ms silence,

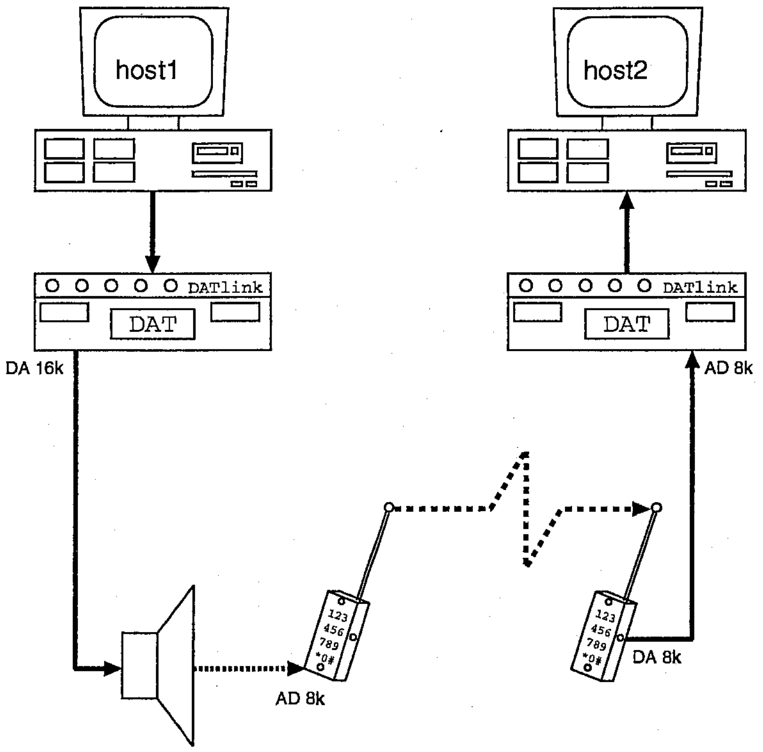

Figure 3: Piping speech data: high quality speech is played using a DAT (DA 16k), recorded by a cellular-phone microphone (AD 8k), transmitted to another cellular-phone and recorded in low bandwidth quality by a DAT (AD 8k).

250 ms frequency 1000 Hz sinus sound, 250ms frequency 1500 Hz sinus sound and another tailing 250 ms silence. For automatic detection, which was not needed here, we later found in the literature [Aoshima, 1981] that a TSP (time stretch pulse) signal or a chirp signal would have been more appropriate.

Figure 4 explains the marking signal distributions used for experiments. In a first implementation we sent the marking signal before each speech signal. We later realized that the drift, i.e. the difference between expected length and actual length of the rerecorded data is small and nearly linear. We changed our implementation and inserted a mark only before the first and after the last speech signal. For concatenation and splitting, the python script `ConcatSplitDB.py` (see Appendix D) is available. The header of the file provides a usage explanation.

(2.3.2) Piping

The concatenated file was played from local disc on one computer and recorded using a second computer, as shown in Figure 3. Typical play and record commands are

```
/usr/local/datlink/bin/naplay -s 16000 -o mono file.16k
/usr/local/datlink/bin/narecord -s 8000 -o left file.8k
```

(2.3.3) Data Refining

Following the steps described above results in a long file with concatenated speech files and marking signals. To be able to use the single utterances, the long concatenated file has to be cut into single wave files. To do this, it is necessary to know the offset, i.e. the position where the leading silence ends and the first speech signal begins, and the position where the last utterance starts or ends. This information is provided by the mark signals inserted during concatenation. The program `ConcatSplitDB.py` provides functions to split up a concatenated file. The beginning of the leading mark and the beginning of the beginning of the last mark must be measured manually, e.g. with `xwaves` or `SpeechEditor`. Depending on how marking signals were inserted (as described in section (2.3.1)), the last mark is either before or after the last speech signal. Using this information and the knowledge about the length of the original files, it is possible to split the long file up into the single files with sufficient precision.

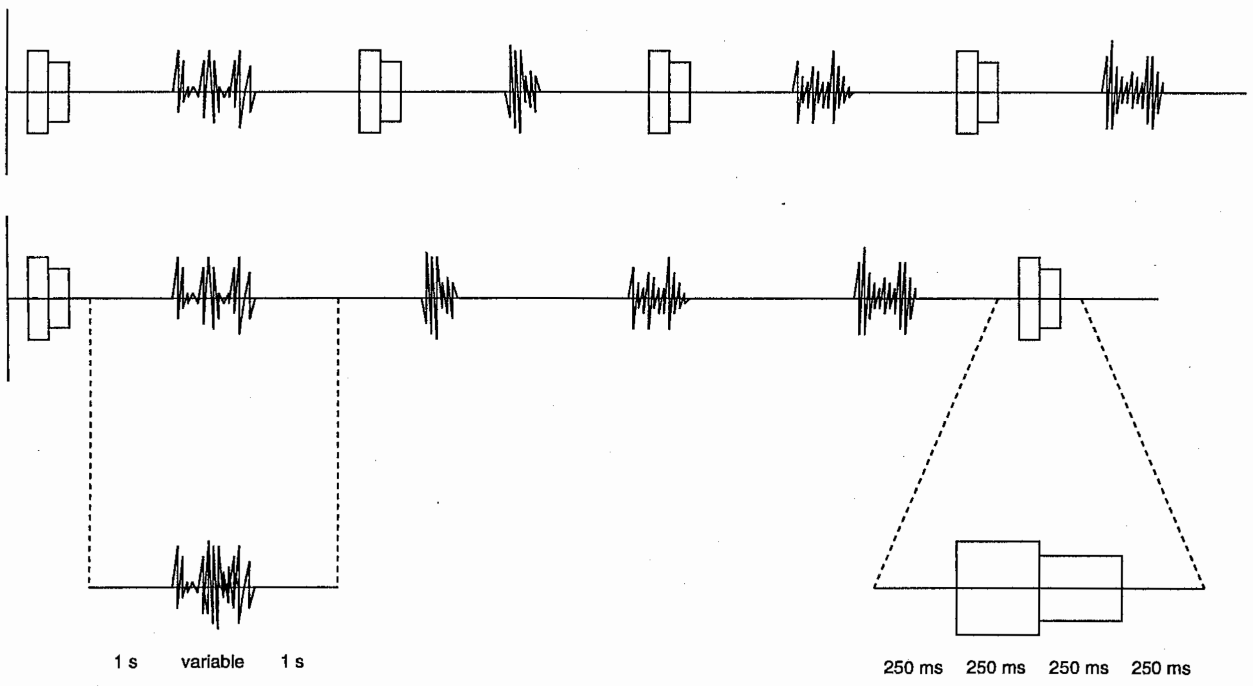

Figure 4: Two ways of marking concatenated speech databases: either put a signal before each speech signal or before the first and after the last. In the bottom, the timing of the speech signals and the marking signal is explained.

The mark signal shows the distortion of sounds transmitted through a cellular phone, which is especially strong if there are clear frequency changes. Figures 5 and 6 show a typical distortion. An improved marking signal would be a TSP, as it can be easily and precisely located by convolving it with the inverted TSP.

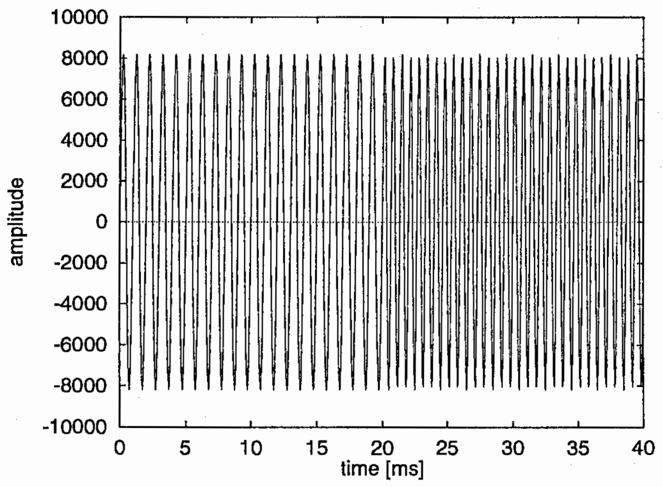

Figure 5: The original marking signal (zoomed to the position of the frequency change).

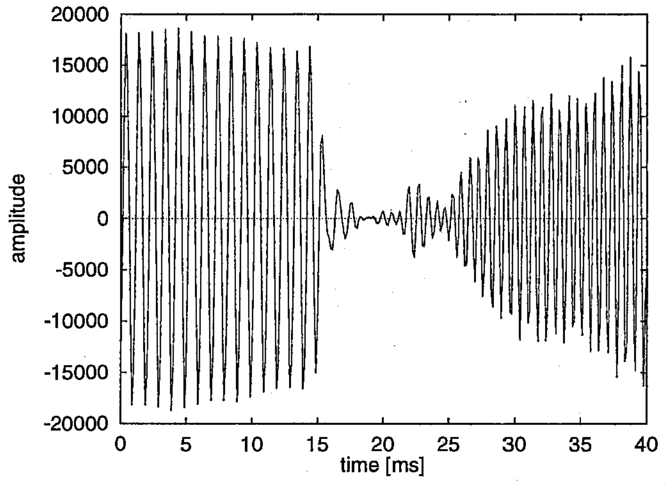

Figure 6: The marking signal piped through a keitai-keitai connection (zoomed to the position of the frequency change).

3 System Setup

This chapter describes how ATR-MATRIX was set up for use with cellular phones. Figure 7 gives an example of how the dialog was taking place, Figure 8 is a more technical view of the setup.

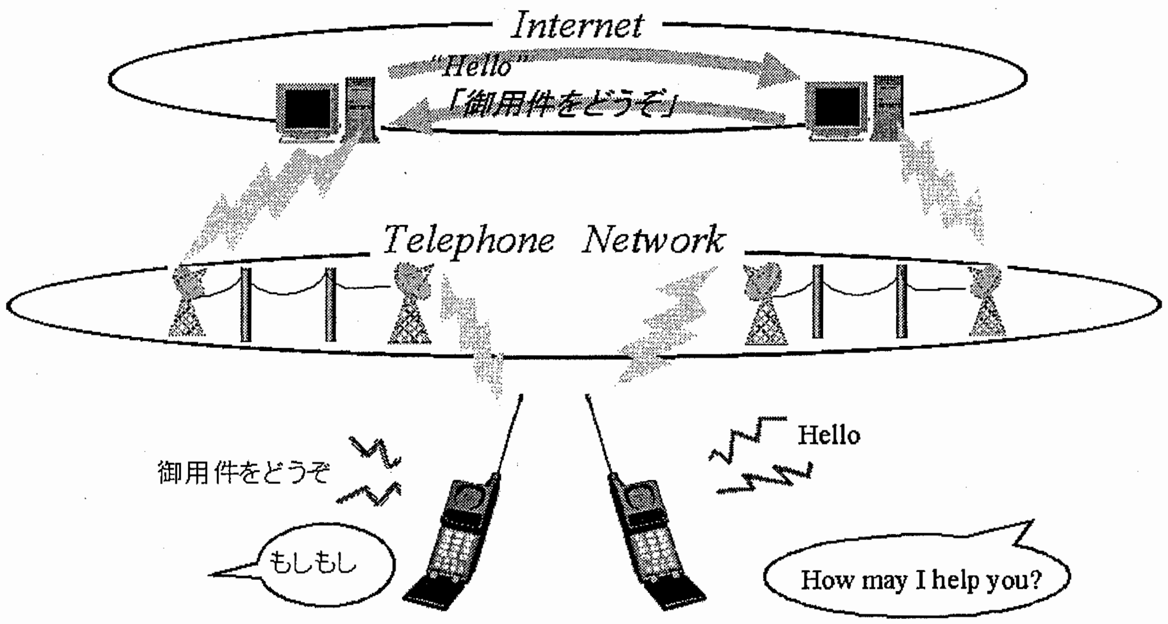

Figure 7: Overview for a Japanese-English cellular phone dialog system

The dialog system consists of two Compaq XP1000 workstations with Alpha 21264 500MHz processor and 368MB memory, running on Digital UNIX V4.0E operating system. Four DoCoMo Doccimo SH811 cellular phones are used.

The headset plug on the side of the doccimo was used to connect to the computer. Using a one stereo microjack to two stereo minijack switch, input- and output lines were separated. The cellular phone output (i.e. the speech from the client phone) was connected to the line-in port of a Sony DAT walkman. The input plug was connected with the soundcard's speaker port on the computer. To ensure continuous power supply the server-side cellular phone was permanently plugged in to the electricity network. Typical level settings are 8 (of 10) on the DAT and maximum on the cellular phone.

The connection between the two ATR-MATRIX servers follows the TIOP protocol [Alshawi, 1999] as described in [Gruhn and Nakamura, 1999]. The transmitted data is translated text. The receiving system generates synthesized speech.

A major problem for the system are dropouts. During a cellular phone conversation, occasionally the transmission is interrupted for a short moment. Utterances spoken during a dropout are discarded. Human listeners can often guess the missing part from the context, but a speech recognition system will fail in such a case. Figure 9 shows an image of a wave signal that was received during a demonstration. The original utterance was "hai, wakarimashita". But only beginning and end were transmitted, in the middle only silence was recorded. The received utterance sounded like "hai ... shita".

Experiments show that dropouts depend on the position of the user, they are more likely close to a strong power cable or in a windowless room, i.e. a place with bad transmission properties and disturbances. Another theory is that busy cellular phone network nodes may be a cause. Experiments also showed that transmission quality and dropout likelyhood for PHS and "keitai" cellular phone systems are not highly related, a PHS connection can be very stable where a "keitai" connection is lost.

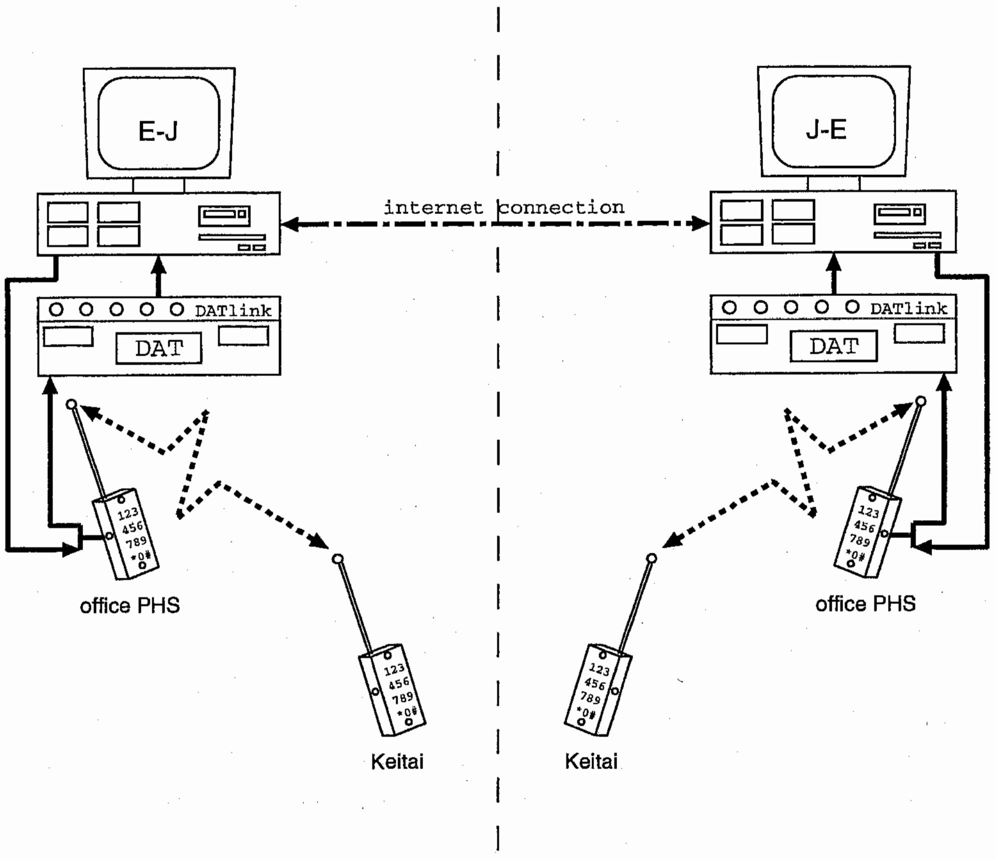

Figure 8: Setup for the Japanese-English cellular phone dialog system used for the ATR Openhouse 1999 demonstration

4 Remaining Problems

4.1 Environmental Noise

How to overcome environmental noise is one of the most important research and development topics for our system. In the real world, cellular-phones are often used in noisy environment, but our speech recognition system has difficulties with speech with background noise. Microphones of the current cellular-phones pick up a lot of environmental noise. To improve the input device of cellular-phones is indispensable for a speech recognition service application. Environmental noise causes start-and-end point detection (EPD) errors on the input speech as well as acoustic model mismatch. It is necessary to develop noise-robust methods for both EPD and acoustic modeling, e.g. as described in [Yamamoto and Singer, 2000].

4.2 Dropout

In current cellular-phone services, sometimes short transmission interruptions occur, e.g. as shown in Figure 9. Even for a human it is difficult to understand utterances with dropouts. More reliable cellular-phone services are desirable for speech recognition applications.

4.3 Design of Human Interface

In our prototype implementation, users are not informed

a) if the system is currently processing,

b) if the recognition subsystem has rejected the input speech, and

c) if the recognition/translation results are reliable, e.g. a confidence score.

As for (a) and (b), the system can give such information to users by playing music etc., whereas (c) is a fundamental problem for a speech-to-speech translation system. Our current design strategy of the system

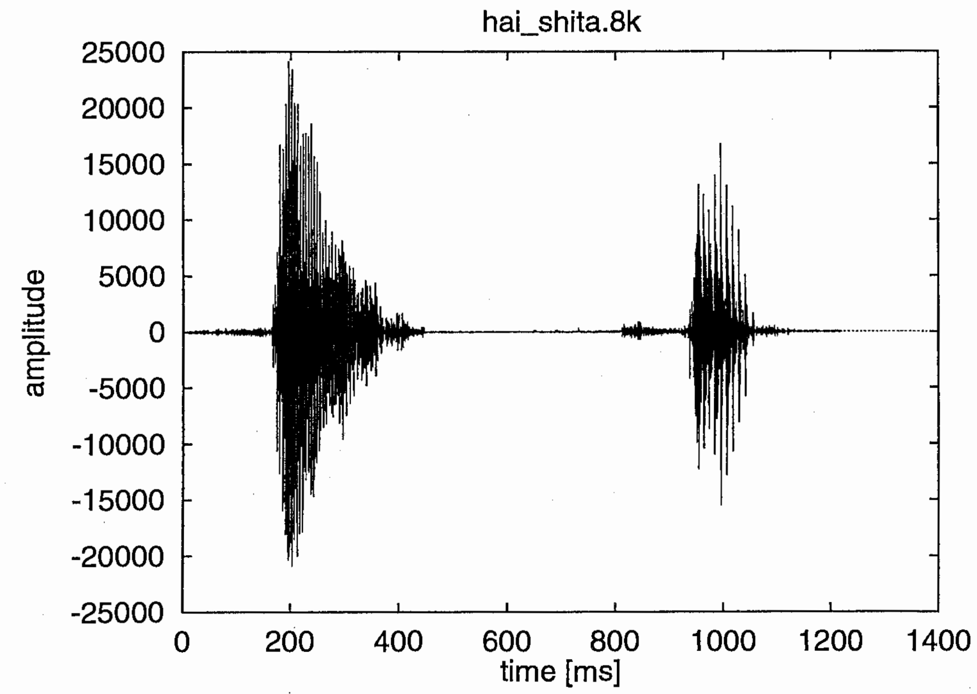

Figure 9: Example for a dropout for the original utterance "hai, wakarimashita": the "i" of hai ends at 440 ms, then 370 ms silence follow until with the "sh" at 810 ms the transmission continued.

is to do nothing about (c) since users probably know it through conversation. However, it would be worth developing methods to know (c), for example using paraphrasing and etc.

4.4 Applicational Problems

Before a cellular-phone based service can be opened, some server construction and support problems must be solved. In the first prototype, which was shown at the ATR Openhouse 1999 and following demos, operators had to answer a customers call manually, i.e. to press the "accept call" button on the server cellular phone. Our most recent system is able to answer the phone automatically, so that an unsupervised demo and long-term test becomes possible. But this requires a keitai-keitai type connection, which has a higher probability of dropouts than the keitai-officePHS connection of the first prototype. The reason lies in currently available modem cards which support giving commands to a cellular phone in speech mode only if it is a keitai. For a usable service we must advance to a computer telephony (CT) board that can handle several users at the same time. Probably, such a system would be based on a keitai-ISDN connection. If such a service is a face-to-face translation system, we must think of a way to connect the right pairs of users: if for example two people in Osaka want to use the system and at the same time two people in Tokyo call the system, how can we ensure not to mix the dialog partners ?

5 Conclusion

We have implemented a cellular-phone based speech translation system and shown several approaches to re-use previously collected high-quality data. For a real-world service, however, human-interface and noise robustness issues still require a major research effort.

Acknowledgement

The authors would like to thank all members and the management of ATR ITL to support us in this research. In particular, we would like to thank Michael Paul, Atsushi Nishino, Takeshi Matsuda, Koji Takashima and Takanori Matsui for their help and assistance in preparing and performing the ATR Openhouse 1999 demonstration.

References

[Alshawi, 1999] Alshawi, H. (1999). Translation inter-operating protocol. specification by AT&T Labs. Copyright AT&T Corp.

[Aoshima, 1981] Aoshima, N. (1981). Computer-generated pulse signal applied for sound measurement. *J. Acoust. Soc. Am.*, 69:1484–1488.

[Brown and George, 1995] Brown, K. and George, E. (1995). CTIMIT: a speech corpus for the cellular environment with applications to automatic speech recognition. In *Proc. ICASSP*, pages 105–108.

[Gruhn and Nakamura, 1999] Gruhn, R. and Nakamura, A. (1999). Tiop protocol connection controller for atr-matrix. Technical Report TR-IT-0302, ATR.

[Jankowski et al., 1990] Jankowski, C., Kalyanswamy, A., Basson, S., and Spitz, J. (1990). NTIMIT: A phonetically balanced, continuous speech, telephone bandwidth speech database. In *Proc. ICASSP*, pages 109–112.

[Matsui et al., 1999] Matsui, T., Naito, M., Singer, H., Nakamura, A., and Sagisaka, Y. (1999). Japanese spontaneous speech database with wide regional and age distribution. In *Proc. EuroSpeech*, pages 2251–2254, Budapest.

[Nakamura et al., 1996] Nakamura, A., Matsunaga, S., Shimizu, T., Tonomura, M., and Sagisaka, Y. (1996). Japanese speech database for robust speech recognition. In *Proc. ICSLP*, pages 137–140, Philadelphia.

[Ostendorf and Singer, 1997] Ostendorf, M. and Singer, H. (1997). HMM topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, 11(1):17–41.

[Singer et al., 1999a] Singer, H., Gruhn, R., Naito, M., Tsukada, H., Nishino, A., Nakamura, A., and Sagisaka, Y. (1999). Speech translation anywhere: Client-server based ATR-MATRIX. Technical Report SP99-121, IEICE.

[Singer et al., 1999b] Singer, H., Gruhn, R., and Sagisaka, Y. (Fall 1999). Speech translation anywhere: Client-server based ATR-MATRIX. In *Proc. Acoust. Soc. Jap.*, pages 165–166.

[Singer et al., 2000] Singer, H., Gruhn, R., Tsukada, H., Naito, M., Nishino, A., Nakamura, A., and Sagisaka, Y. (Spring 2000). Cellular-phone based speech translation system ATR-MATRIX. In *Proc. Acoust. Soc. Jap.* (to appear).

[Takezawa et al., 1998] Takezawa, T., T.Morimoto, Sagisaka, Y., Campbell, N., Iida, H., Sugaya, F., Yokoo, A., and Yamamoto, S. (1998). A Japanese-to-English speech translation system: ATR-MATRIX. In *Proc. ICSLP*, pages 957–960.

[Tonomura et al., 1995] Tonomura, M., Kosaka, T., Matsunaga, S., and Monden, A. (1995). Speaker adaptation fitting training data size and contents. In *Proc. EuroSpeech*, pages 1147–1150, Madrid.

[Woudenberg, 1994] Woudenberg, E. (1994). Telephone band conversion of studio quality audio data. Technical Report TR-H-109, ATR.

[Yamamoto and Sagisaka, 1999] Yamamoto, H. and Sagisaka, Y. (1999). Multi-class composite n-gram based on connection direction. In *Proc. ICASSP*, pages 533–536.

[Yamamoto and Singer, 2000] Yamamoto, H. and Singer, H. (2000). Speech-start-point detection using vowel and non-speech HMMs. In *Proc. Acoust. Soc. Jap.* (to appear, in Japanese).

A Telephone Filter

The FIR filter used in this project is based on the specifications in Table 3.

It was created with `$ATRSPREC/sample/ATRsrconv/CreateFilter.py` for use with `ATRsrconv`.

Table 3: The telephone filter used to generate the data for the baseline acoustic model [Woudenberg, 1994].

frequency	dB
100	-45.8
125	-36.1
160	-25.6
200	-19.2
250	-14.3
300	-11.3
315	-10.8
400	-8.4
500	-6.9
600	-6.3
630	-6.1
800	-4.9
1000	-3.7
1250	-2.3
1600	-0.6
2000	0.3
2500	1.8
3000	1.5
3150	1.8
3500	-7.3
4000	-37.2

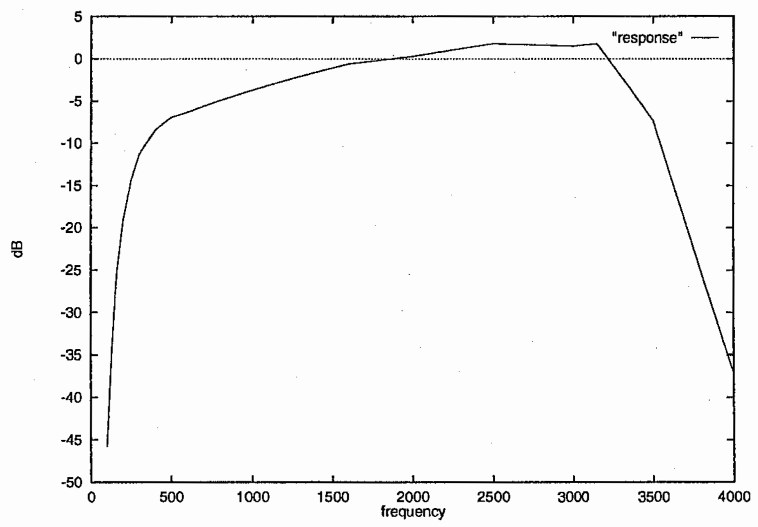

Figure 10: CCITT Telephone Band Response Curve

B Example Configuration File

This chapter provides an example for a ATRSPREC configuration file as used for the English-to-Japanese side in the 1999 ATR openhouse demonstration. It was used with ATRSPREC version r06r05.

```
# English Cellular speech   Harald Singer
# last update: 15OCT99
#
# debugging with ATRall:
#   $ATRSPREC/bin/ATRall -config=config.e.keitai
#       -ATRepd:sendNonspeech=0 -I/Ocontrol:inputEOFexit=ON
#       -inputFd=/DB/MDB/EDB1/rick/OH99NB/WAV/M001_TOS00001_1/M001_TOS00001.0010.A.8k

I/Ocontrol:inputFd=process("$ATRSPREC/sample/ATRcollect/atrCollectDoubleSocket.py -portr=24042 \
                -stdout=OK -portws=24442")
I/Ocontrol:inputFormat=NoHeader
I/Ocontrol:inputParamSize=80
I/Ocontrol:inputParamType=short
I/Ocontrol:inputEOFexit=OFF
I/Ocontrol:inputByteorder=LittleEndian
I/Ocontrol:outputFormat=NULL
I/Ocontrol:outputFd=stdout
I/Ocontrol:outputByteorder=BigEndian
I/Ocontrol:rpcNumber=5
I/Ocontrol:inputDecompress=OFF
I/Ocontrol:outputCompress=OFF

ATRwavecut:pause_symbol=-
ATRwavecut:PausePeriod=NOT
ATRwavecut:SamplingFrequency=8000.0

ATRepd:SamplingFrequency=8000
ATRepd:energyThreshold=100
ATRepd:upperDispersionThreshold=70
ATRepd:lowerDispersionThreshold=15
ATRepd:orderInMs=300
ATRepd:alpha=1.05
ATRepd:skewInMs=300
ATRepd:s1TimerLimitInMs=200
ATRepd:epdFramesInMs=2000
ATRepd:framePointsInMs=10
ATRepd:sendNonspeech=0
ATRepd:track=0

## this requires at least SPREC r06r04j
ATRwave2cep:Preemphasis=0.95
ATRwave2cep:FrameLength=25
ATRwave2cep:FrameShift=10
ATRwave2cep:SamplingFrequency=8000
ATRwave2cep:TimeWindow=hamming
ATRwave2cep:LagWindowFactor=0.01
ATRwave2cep:LpcOrder=16
ATRwave2cep:CepstrumOrder=12
ATRwave2cep:FrequencyWarping=mel
ATRwave2cep:FilterBankOrder=20
ATRwave2cep:CutoffLowFrequency=200
ATRwave2cep:CutoffHighFrequency=3400
ATRwave2cep:AnalysisType=fft
ATRwave2cep:DebuggingLevel=0
ATRwave2cep:HTKV2CompatEnergy=ON
ATRwave2cep:MeanInFile=$MATRIX_RESOURCES/ccs-models/emodel/V3.0/amodel/all1.mean
ATRwave2cep:Subtract=logpow+cep

ATRdisplaypow:meterCmd="$ATRSPREC/bin/socket -sl 50000"
ATRdisplaypow:indicatorCmd="$ATRSPREC/bin/socket -sl 50001"
ATRdisplaypow:refresh=50
ATRdisplaypow:powerlevel=-12.5,-2.5

ATRcep2para:OutputParameter=cep(12)+dpow+dcep(12)
ATRcep2para:rho=1.0
ATRcep2para:DDCepstrumPadding=zero
ATRcep2para:deltaCepstrumPadding=zero
ATRcep2para:DeltaCepstrumWindow=5
ATRcep2para:CepstrumOrder=12
ATRcep2para:WindowType=rectangular
ATRcep2para:DebuggingLevel=10

ATRlattice:lexicon=$MATRIX_RESOURCES/ccs-models/emodel/19990406/lmodel/hrt.1000.lex.open99.v3
ATRlattice:ngram=$MATRIX_RESOURCES/ccs-models/emodel/V3.0/lmodel/hrt.1000.bin
ATRlattice:amname=$MATRIX_RESOURCES/ccs-models/emodel/V3.0/amodel/AM.M.HS1.bin
ATRlattice:active_model=all
ATRlattice:lmscale=6,8
```

```
ATRlattice:beam=65,50
ATRlattice:work_area=500,50
ATRlattice:wdpenalty=0,0
ATRlattice:UTT_END_delay=-1
ATRlattice:word_boundary_skip=2
ATRlattice:frame_shift=10
ATRlattice:pause_symbol=SIL
ATRlattice:dimension=25
ATRlattice:max_allophone=20000
ATRlattice:phone_boundary=ON
ATRlattice:word_merge=all
ATRlattice:UTT_START=5
ATRlattice:UTT_END=6
ATRlattice:backward_frame=-1
ATRlattice:amscale=1.000000
ATRlattice:FSA=
ATRlattice:null_trans=OFF
ATRlattice:state_skip=OFF
ATRlattice:active_lmodel=1

sprec97:startWithPitchOn=0
sprec97:startWithMicOn=0
sprec97:startWithBunkatsuOn=0
sprec97:n=1
sprec97:track=0
sprec97:epdShutoff=0
sprec97:minimumDuration=0.4
sprec97:saveWaveFiles=0
sprec97:language=e
#EOF
```

C Automatic Cellular-Phone Answering System Manual

RINGwatcher.exe

[概要]
シリアルポートに接続されているモデムから"RING"という文字列を取得すると、
（電話に着信すると、）
ユーザーが指定した文字列をソケットを介して他のマシンに送信する事が出来ます。

[免責の確認]
このプログラムの実行による如何なる損害も作者は一切責任を負えません。

[実行条件]
mscomm32.ocx がインストールされているマシンでないと実行出来ません。

[RINGwatcher.exe のインストール方法]
任意のディレクトリに RINGwatcher.exe をコピーして下さい。

[mscomm32.ocx のインストール方法]
---Windows 95/98 の場合
1) mscomm32.ocx を、c:\windows\system にコピーする。
2) MS-DOS プロンプトで、c:\windows\system まで cd する。
3) regsvr32 mscomm32.ocx と入力してリターンキー押下。
4) "DllRegisterServer in mscomm32.ocx succeeded."
　というメッセージボックスが表示されたらインストール成功。

---Windows NT の場合
1) mscomm32.ocx を、c:\winnt\system32 にコピーする。
2) MS-DOS プロンプトで、c:\winnt\system32 まで cd する。
3) regsvr32 mscomm32.ocx と入力してリターンキー押下。
4) success のメッセージボックスが表示されたらインストール成功。

[mscomm32.ocx のアンインストール]
1) 上記の 3) で regsvr32 /u mscomm32.ocx と入力してリターン
2) "DllUnregisterServer in mscomm32.ocx succeeded."
　というメッセージボックスが表示されたらレジストリ情報のクリア成功。
3) mscomm32.ocx を削除して下さい。
　(別にそのまま置いておいても特に問題は無いとは思いますが。)

[設定値について]
アプリケーションの設定値は、
HKEY_CURRENT_USER\Software\ATR\ITL\TSG\RINGwatcher
に保存されます。

何かありましたら以下まで連絡お願い致します。
t_matsui@itl.atr.co.jp

D Script for Concatenation and Splitting of Wavefiles

```
#!/usr/bin/env python
# ATRSPREC Copyright 1997,1998 ATR Interpreting Telecommunications Research Laboratories
"""
H. Singer, August 99
concatenating/splitting up wavefiles for transmission over noisy channels

   ConcatSplitDB.py

example:
   # for concatenating
   python -c "import ConcatSplitDB; ConcatSplitDB.test16()"

   # for splitting
   python -c "import ConcatSplitDB; ConcatSplitDB.test17()"

how to find start/end via SpeechEditor
----------------------------------------
setenv LD_LIBRARY_PATH "$LD_LIBRARY_PATH":/home/singer/work/lesstif/lib
/home/singer/bin/Linux/SpeechEditor
 [Open...]
 select ByteSwap  (data is BigEndian, Linux is LittleEndian)
 select file (have patience with loading)

 - zoom in on beginning of first beep and beginning of last beep
 - enter offset and lastuttstart values in testXX()
 - run it
 - select a file in the middle, e.g.
   /DB/MDB/EDB1/rick/OH99WB/WAVA/M006_sx/M006_sx.0010.B.8k
   and enter the last value into SpeechEditor "offset", e.g. 2583268

"""
import os, string, re
os.sys.path.append(os.environ['ATRSPREC']+'/script/python/lib')
import atrdb

def ConcatenateFile2beep(opts):

    if type(opts['db'])==type(''): opts['db']=eval(opts['db'])
    if type(opts['badids']) == type(''): opts['badids']=eval(opts['badids'])
    if type(opts['AsciiFile'])==type('') and opts['AsciiFile']!='None':
        opts['AsciiFile']=string.split(opts['AsciiFile'],',')
    if opts['idsFile']=='None': opts['idsFile']=None
    if opts['ansSpecial']=='None': opts['ansSpecial']=None

    e = atrdb.RunDB(opts=opts)

    fd = open(opts['longfname'],'wb')
    fl = open(opts['listfname'],'w')
    print 'writing to', opts['longfname'], opts['listfname']

    ids = e.db.GetIDs()
    for id in ids:
        wavfname = e.db.GetWAV(id)
        line = '%s %d\n'%(wavfname,os.stat(wavfname)[6])
        print line,
        fl.write(line)

        if id == ids[0]:
            # start with mark wave file
            fd.write(open(opts['markfname'],'rb').read())
        fd.write(open(wavfname,'rb').read())

    # finish with mark wave file
    fd.write(open(opts['markfname'],'rb').read())
    fd.close()
    fl.close()

def SplitFile2beep(
        SrcSamplingRate,
        DstSamplingRate,
        listfname,
        phsfname,
        SrcWavDir,
        DstWavDir,
        offset,              # the beginning of the leading beep
        lastuttend,          # the beginning of the final beep
        ):

    buf = map(lambda x: string.split(x), open(listfname).readlines())
    # real start of first wave file
    offset = offset + 750
    # real end of last wave file
    lastuttend = lastuttend - 250

    # calculated estimated duration in msec for all files
    accdurationPrev = 0.0
    for fname,bytes in buf:
        # assuming 2 byte per sample
        accdurationPrev=accdurationPrev+string.atoi(bytes)/(SrcSamplingRate*2.0)

    # difference between real duration and expected duration
    drift  = (lastuttend - offset) - accdurationPrev

    fp = open(phsfname,'rb')

    accduration = 0.0   # accumulated duration
    start = offset
    for fname,bytes in buf:
        duration =  string.atoi(bytes) / (SrcSamplingRate * 2.0)

        accduration = accduration + duration

        adjustment = drift * accduration / accdurationPrev
        end = offset + accduration + adjustment

        print fname, duration, accduration, start, end, adjustment

        fname = re.sub('16k$','%dk'%(DstSamplingRate),fname)
        fname = re.sub(SrcWavDir,DstWavDir,fname)

        # in byte, please note that duration is the expected duration,
        # i.e. we cannot really handle large divergences
        startb = (int)(start * DstSamplingRate) * 2
        durb = (int)(duration * DstSamplingRate) * 2
        print fname, startb, durb, accduration, start

        if 1:
            fp.seek(startb)
            os.system('mkdir -p %s'%os.path.split(fname)[0])
            open(fname, 'wb').write(fp.read(durb))

        start = end

    fp.close()
    print 'drift=',drift, 'accduration=',accduration

### 2 beep file
def test16():
    opts={
        'idsFile':'None',
        'AsciiFile': [
            '/DB/MDB/EDB1/rick/eval991118/info.ascii',
            ],
        'db':{
            'wavDir':'/DB/MDB/MDB12/SPH/WAV/JAPANESE',
            'wavExt':'16k',
            'trsDir':'/DB/MDB/MDB12/SPH/TRSwithALLp/JAPANESE',
            'trsExt':'TRS',
            'ansDir':'/RR/Recognition/MatrixE/ldata/19990406/ANS',
            'ansExt':'ANS',
            },
        'ansSpecial':'None',            # EDB, MDB, OTHERS, None
        'badids':[],
        'longfname':'/DB/MDB/EDB1/rick/eval991118/eval991118.16k',
        'listfname':'/DB/MDB/EDB1/rick/eval991118/eval991118.list',
        'markfname':'mark.16k',
        }
    ConcatenateFile2beep(opts)

def test17():
    SplitFile2beep(
        SrcSamplingRate=16,
        DstSamplingRate=8,
        listfname='/DB/MDB/EDB1/rick/eval991118/eval991118.list',
        phsfname='/DB/MDB/EDB1/rick/eval991118/rightkeitaikeitai.8k',
        SrcWavDir='/DB/MDB/MDB12/SPH/WAV/JAPANESE',
        DstWavDir='/DB/MDB/EDB1/rick/eval991118/rightkeitaikeitai',
        offset=24193.0,
        lastuttend=3171572.0,
        )

## EOF
```

E Creating the Marking Signal

This section contains a python script showing how the marking signal used in the piping experiments was created. The script is an excerpt from `$ATRSPREC/sample/ATRsrconv/CreateFilter.py`

```
#!/usr/bin/env python
import Numeric,struct

def CreateSignal(f=1000.0, sr=16000.0,dur=1.0,func=Numeric.sin, amp=1.0):
    return func(Numeric.arange(0,f*2*Numeric.pi*dur,f*2*Numeric.pi/sr)) * amp

def CreateCompositeSignal(fname='mark.16k'):
    x = Numeric.concatenate((
        CreateSignal(f=1000, dur=0.25, amp=0),
        CreateSignal(f=1000, dur=0.25, amp=2**13),
        CreateSignal(f=1500, dur=0.25, amp=2**13),
        CreateSignal(f=1000, dur=0.25, amp=0),
        ))
    MachineByteOrder = ((struct.pack('l',1)[0]=='\001' and ['LittleEndian']) or ['BigEndian'])[0]
    if MachineByteOrder == 'BigEndian':
        open(fname,'wb').write(x.astype(Numeric.Int16).tostring())
    else:
        open(fname,'wb').write(x.astype(Numeric.Int16).byteswapped().tostring())
# EOF
```