TR-IT-0163

Building practical speech synthesis systems

Alan W. Black
アラン W ブラック

1996.03

This report discusses a history and philosophy of CHATR a generic speech synthesis system. Specific examples are given of various levels of use of the system. The currently released system is described and important aspects higlighted. Possible extensions to the system are discussed and current problems identified. This report is not intended as a user document for CHATR but a discussion of the overall system.

1 Introduction

This report is as a comment on my research in ATR over my three year stay (1993-1996). It concentrates on the development of CHATR a generic speech synthesis system. CHATR offers a practical system for research, development and general use of speech synthesis techniques. The current system allows text to speech for Japanese and English, in a large number of voices. It also allows automatic building of new voices from speech databases. It is flexible, written to be modified and documented. It is intended as the framework for future development of speech synthesis within department 2 of ITL.

Although the seeds of a general synthesis workbench/system were conceived before I came to ATR, the birth and christening happened not long after my arrival. The name CHATR was chosen (by Paul Taylor) after significant discussions about possible names for a system. The name CHATR is an recursive acronym standing for "CHATR at ATR". It is a deliberate play on the English word "chatter" (to talk quickly or casually) and the inclusion of the initials ATR.

After the choice of a good name, the system started to develop. It actually spoke for the first time on 28th April 1993 just under a month after I arrived, though the quality of the synthesis was virtually unintelligible.

Although many people have made significant contributions to the development and research behind CHATR, my own work has centered around the core system acting as software manager, as well as writing (and often re-writing) many of the modules within the system.

A revision history of the system is provided at the end of this document showing the release dates and major developments of the system.

Although often software development is not considered part of core research my own views are that without well engineered code, research is difficult and often unrepeatable. A well designed program, although perhaps taking longer to develop in the first place, will in the long run provide a better environment within which experiments may be carried out. Also due to there being a specific research goal within ITL, namely to develop speech translation systems, a practical implementation of our research is necessary if it is going to contribute to the overall project. Thus CHATR not only provides an environment for experimentation of new speech synthesis techniques but offers a conduit for the results of our research to be made available to people outside our immediate research group.

The main advantages of a system specifically designed to be modular are as follows

- individual researchers need not each develop their own full synthesizer but concentrate on their own work, benefiting from the modules already provided.
- individual modules once integrated within a central system have a chance of running after a researcher has left ATR, thus offering a continuity for work.
- A full working system, offers others outside the group a synthesis system, which can be upgraded as new research is completed.

- A full working system allows clear identification of overall problems that require further work.

To be fair there are disadvantages too, especially if the basic synthesizer system is not general enough, or too difficult or slow to use. A centralized system must offer interfaces and freedom such that it is suitable for individuals' research. However some constraints are necessary and good if they are to allow individual work to be integrated and used together.

From the start CHATR was specifically designed to be both a working synthesis system and also an environment within which new modules may be integrated and tested. Where ever possible no fixed decisions about the language to be synthesized, the phoneme set, the byte order, the audio hardware etc. have been made. The system was designed to be a general framework that would meet the needs of future researchers long after the original authors had left.

2 Philosophy

The general philosophy behind CHATR is that it can be used at three distinct levels, but of course there are often times when one person uses multiple levels.

The first level is the person who just wants a talking computer. They have little interest in the actual synthesis methods, though will be worried about quality and speed of synthesis. At this level CHATR may be used as a black box synthesis system. They can simply take text (in Japanese, English or mixed) and produce an audio rendering of the text. At this level CHATR simply acts as a component within a large system that happens to require speech output.

The second level is using CHATR as a toolkit, to develop existing sub-systems within it. In this mode users would most likely use the Lisp scripting Language offered by CHATR to investigate or train existing systems. Examples of this use are: the building of a new voice in CHATR from a speech database, training duration modules and intonation modules for supported or even new languages. Specifically, in this mode CHATR is used without any change in the C code. The installed version is used using Lisp programs or shell scripts as necessary.

The lowest level is the use of CHATR as a research system during the development of new synthesis modules. For example in the creation of a new duration module, intonation or waveform synthesizer. In this case a user has their own copy of the code and writes C (or C++) code and adds new functionality to the system. Of course all that should be necessary is the addition of their new module. As CHATR offers various display mechanisms for waveforms, F_0, segments etc as well as input at various levels, a user should be able to develop their own module and get the benefit of a whole existing system to aid their work.

To show CHATR's use more clearly we will give some specific examples of work that has already been done within CHATR at the various levels described above.

2.1 Telephone-based email reader

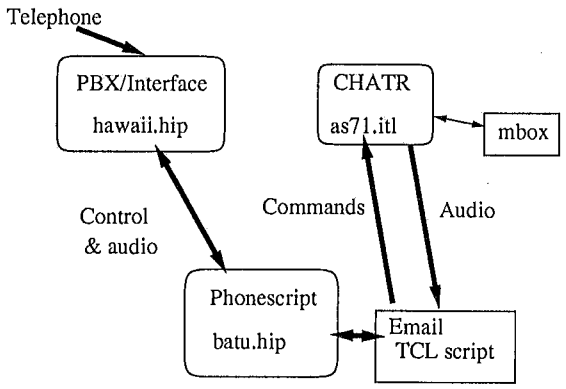
To show CHATR's capabilities as black-box synthesis system, a system that allows a user to hear their electronic mail over the telephone was written.

Imagine you wish to check your email but do not have your laptop with you. As your email is in textual form there is no way to read it when all you have is a telephone. With the help of Rick Woudenberg (HIP), a small demonstration system was built.

The system runs on three separate machines. One (hawaii.hip) offers the interface to the telephone system answering calls and routing them to the main phoneserver running on the second machine. The second machine (batu.hip) runs a server program that accepts calls and chooses the appropriate service. This mechanism on these two machines already existed, to implemented our email reader we had to add two things.

First a new service was added to batu.hip. This service, written in the Phonescript language (a tcl interface to lower level telephone commands) accepts button presses from the telephone (through the interface on hawaii.hip) and converts them to simple string commands. The second new requirement runs on the third machine (as71.itl) and is a CHATR server which both accesses the user's mail file and renders it as audio. The email service on batu.hip sends simple commands such as `(SM_Start)` and `(SM_Message N)` to the CHATR server. The CHATR server uses an external shell script to access the user's mail file. It extracts the headers or desired message as required. It then synthesizes the requested text, automatically dealing with Japanese and English in the same message and then sends the audio back to batu.hip, which in turn sends it to hawaii.hip and back out of the user's telephone.

The following diagram shows the major parts of the telephone email system.



The telephone email interface is purely a demonstration. Currently only one mail file is accessed (with no security). Long messages cannot be interrupted except by hanging up the phone. Although there is much that could be done to improve this interface, the problems are not due to the synthesis system itself. This system explicitly shows how CHATR is appropriate to be embedded within large systems.

2.2 Toolkit level

The second level is the use of CHATR as speech synthesis development system without adding any new C code to it. One of the important aspects of CHATR is that it offers a small Lisp-based scripting language to allow specification of parameters, flow of control and simple programs to be written in a standard text based form. Command line options are kept to the minimum.

Many of CHATR's existing sub-systems allow quite complex external parameter setting. In fact it is design philosophy of CHATR that as many parameters as possible should be settable at run time through Lisp variables and commands rather than require recompilation of the system.

A good example of using CHATR at toolkit level is using the system to train new intonation modules. One of the many intonation systems available in CHATR is an implementation of the ToBI system [Silverman *et al* 92]. For speech synthesis, an implementation of ToBI requires two parts. First the prediction of ToBI accents, tones and break indices, and second the realisation of an F_0 contour based on these labels (and perhaps other data).

The first stage is not discussed here, but the second stage, rendering an F_0 contour, will be. CHATR current supports two methods for this stage. The first, based on [Anderson *et al* 84], uses hand specified rules to predict target points based on the accents and tones on each syllable. The results are acceptable but parameters in the prediction require tuning which at present can only be done by hand. The second technique, which will be discussed more fully, uniformly predicts three target points on every syllable, using linear regression. The advantage of the second technique is that there are no rules, and no parameters to hand tune. The resulting F_0 has both a higher correlation, and a smaller RMS error than the Anderson method, and in simple listening tests, 70% of a test sample, were preferred to the older rule-driven method.

As stated this second method of rendering an F_0 from ToBI labels is fully trainable, in addition the method is trainable for new speakers without any change the C code. For training data, it necessary to extract three feature vectors for each syllable in the train set. Feature 0 should be the actual F_0 for the start, mid vowel and end point of the syllable respectively. Features 1 through n should be features of the syllable that are to be used to predict the F_0 value. Currently for English we use the following features:

current accent (and that of two preceding syllables and two following syllables)
break level after current syllable (and break level after two preceding and two following syllables)
end tone after current syllable (and end tone after two preceding and two following syllables)
number of (lexically) stressed syllables since the last major phrase boundary.
number of syllables since the last major phrase boundary.
number of (lexically) stressed syllables to the next major phrase boundary.
number of syllables to the next major phrase boundary.
number of accented syllables since late major phrase boundary

number of accented syllables to next major phrase boundary
lexical stress of current syllable (and of two preceding and two following syllables)
number of previous sub-phrases (break levels 2 or 3) since last major phrase break (level 4).

Linear regression models are trained to predict the three values for each syllable. The resulting targets are then smoothed. The following table shows results for a test set of 2778 syllables (not used in the training) showing the correlation of the predicted F_0 contour with the original, and the RMS error in Hz.

	APL	LR
Correlation	0.3991	0.6195
RMS	44.7	34.8

As we can see the correlation is significantly better, and the RMS error is smaller for the linear regression based model (LR) than for the Anderson et al. (APL) model.

This same technique has been used for Japanese producing similarly better results than a rule based approach as described in [Pierrehumbert & Beckman 88]. Again as shown in the table the correlation is better and the RMS is smaller.

	APL	LR
Correlation	0.5528	0.6996
RMS	25.6	20.9

There are problems with this method. Particularly if the database used for training does not contain all types of ToBI labels. For example the f2b news reader database only contains 3 examples of phrase final rises. It is not possible to train from such few examples. Thus the model created does not in fact generate final rises for syllables labelled with `H-H%`, while of course the APL model does. This would of course be solved by having a more appropriate database for the types of synthesis required.

This whole technique and the above problem is discussed more fully in [Black & Hunt 96].

This concludes an example of the second level of using CHATR as a toolkit: that of using an existing system to retrain new modules for predicting F_0.

2.3 New modules

The third and lowest level of using CHATR is the development of new synthesis modules themselves within the CHATR framework. The advantage of development within CHATR rather than externally, is that you can have immediately access to the existing sub-systems within CHATR. The display mechanisms allow a user to view, the waveform, the predicted F_0, segment durations, ToBI labels etc. Comparisons may be done at different levels, so

synthesized waveforms may be directly compared with originals. Original F_0 contours can be compared with predicted ones.

But often the most mundane items are the ones that offer the most convenience. Once CHATR is set up it may play waveforms of any sample rate so results can easily be played. New modules may be quickly slotted in to the whole existing system so they may be check with text to speech modules as well as particular tests.

Suppose a researcher was interested in the phenomenon of vowel reduction in English. A specific example of dealing with vowel reduction was recently added to CHATR. The phenomena causes certain vowels in certain contexts to be reduced to schwas. Roughly this occurs in unstressed syllables, but simply implementing that as a rule causes more vowels to be reduced than is really necessary.

The following new module was added to replace an older version that was very minimal. An important contributions to the ease of building such a module is the use of the PhonoForm utterances. For some databases we have automatically built detailed representations of each utterance in a database containing phonemes, duration, power, F_0, syllable, word, ToBI labels and phrasing. (This is the same format used to generate the F_0 prediction features discussed above.) Using this information we can easily access complex relationships within an utterance a extract feature vectors for some external training algorithm.

In the case of reduction we want to know two things. First what are the reduced/unreduced vowel pairs for the given phoneme set. And secondly which syllables contain reduced vowels when compared with the lexical entry for the word the syllable is contained within. The first is partly defined by the phoneme set itself but it would be interesting if such a relationship could be found automatically. The first stage was to add a function which given a syllable would look up the word which contained that syllable in the lexicon. Then looking at the position of the syllable in the actual utternace and the lexical entry determine the lexical entry vowel and the actual vowel. This is not always possible, some pronunciations have a different number of syllables from the lexical entry, but well over 95% of the cases this is easy. If the lexical vowel and actual vowel are same, then there is no reduction, if they are different there *may* be reduction. At first all the differing vowels were listed and it was decided which were reductions (most) and which were something else (different pronunciations). Once given a list of actual unreduced vowels to reduced vowels, we again checked all syllables in the database and could tell if they were reduced, unreduced or unknown. The unknown ones were less than 1% and hence removed from further training. We then collected other features about the syllables, accentedness, lexical stress (plus this information from surrounding syllables), position in phrase.

We then used a CART regression tree system to learn the conditions for vowel reduction. The results were good and the tree generated was quite readable. Only unstressed, unaccented syllables are reduced, except in some certain cases near phrase boundaries. A new module was added to CHATR after lexical lookup which checked the features on each syllable and used the CART produced tree to predict if that syllable should be reduced. A lookup table for the appropriate phoneme set provides the mapping from unreduced vowel to reduced form.

This new module improves the quality of the English text to speech making it sound more fluent than before.

This new module was written using many of the sub-systems already built in to CHATR. The initial investigation of why segmental quality of text to speech was bad was made by displaying the original segments with those predicted by text to speech. It was then obvious that reduction was a problem. After the model was created the same comparison of predicted segments with natual segments showed a substantial reduction in differences. With full text to speech and synthesis the improvements could easily be heard.

This technique of comparing predicted segments with actual from naturally spoken utterances is easy to do in CHATR and useful in cases other than English vowel reduction. Japanese unvoiced vowels are a phenomenon which could be dealt with in a similar way. The vowel "u" and "i" are commonly reduced to unvoiced versions in fast speech, and at the end of phrases (`desu` is usually pronounced as `des`). The above method of finding the syllables (or mora) where actual pronunciation differs from lexical specification could be used and the appropriate features used to predict this.

This ends the example of building new modules within CHATR. However as many modules already exist that use existing tools, adding new modules can be very easy. For example, because there already are functions to interpret CART decision trees within CHATR, it is often just necessary to call the specified decision tree of all syllables (or whatever) and make some simple change based on the predicted decision to add new functionality.

As can be seen by the three examples above, CHATR offers multi-level use for different types of research. It has been specifically designed to address these needs. CHATR provides a clear route from experimental research to a stable system suitable for non-synthesis researchers to use.

3 Highlights

In this section some of the "highlights" of the CHATR system will be discussed. Particularly those parts which I feel make CHATR a useful system.

3.1 Scripting language

Although at first not thought of as a fundamental part of the system, the inclusion of a scripting language in CHATR is a major contribution to the usefulness of the system. A simple Lisp based language is integrated into CHATR allowing specification of various parameters to sub-systems, specifying flow of control, and general programming. This script language allows CHATR to be used for many tasks without having to change C code and recompile the system. The number of parameters available in CHATR is large. If they were to be set by command line options, calls to CHATR would be completely unstructured. A clear well-defined language which can deal with synthesis objects such as utterances and stream

cells, as well as lists, functions, integers and floats makes it possible to specify the desired behavior.

When the implementation of CHATR was first started, the idea of a scripting language was not as common as it is now. The ideas of including an underlying interpretive language in CHATR are mostly borrowed from EMACS, and it has proven to be a most useful part of the system.

For example there are various parameters that need to be set when choosing between different speakers, as well as selecting an appropriate database itself, lexicons, intonation and duration statistics, phoneme sets all need to be changed when a new speaker is selected. Using standard Lisp functions we have settled on a convention of defining a function `speaker_DBNAME` which sets up all the appropriate parameters for a speaker. We have even allowed for an auto-load version of this definition.

As Lisp allows functions to be *first class objects*, using functions as parameters has also allowed a well-defined way to increase the power of the system. Again borrowing from EMACS, CHATR supports the concept of `hooks`. A hook is a run-time definable list of functions which are to called at some particular time in the synthesis process. In CHATR, hooks are available at a number of points in the synthesis of an utterance. Most importantly at utterance creation time, and after waveform synthesis. With these hooks we can specify any functions which must also be evaluated on a utterance at that time. For example suppose we wish all waveforms to be power normalized to 0.7 of their maximum (so different voices have the same approximate volume). Without hooks we would need to call the Lisp function

```
(Regain_Wave UTT '0.7)
```

for all utterances after synthesis. Finding out where all calls to the `Synth` function is not easy, sometimes its called internally as in tts, or explicitly, or perhaps deep within some user-written function. If we define a function which takes a single utterance as an argument we can state that it should be called after all synthesis calls via the hook called `synth_hook`

```
(define pow_norm (utt)
   (Regain_Wave utt '0.7))
(set synth_hook (list pow_norm))
```

Now after every call to the waveform generator the function `pow_norm` will be called with the utterance as an argument, thus allowing the normalization of the waveform power.

Therefore we simply change one place and all calls to `Synth` will go through our function.

3.2 Unit selection

One sub-system that has attracted major development within CHATR is the sub-system for selecting units of speech from a speech database. This sub-system has been through many

versions with substantial rewrites, although the basic index format has remained constant (compiled indices from two and a half years ago are still compatible).

This sub-system was developed in response to a number of issues, its predecessor NUUTALK [Sagisaka *et al* 92] showed how the selection of non-uniform units from a speech database could produce high quality synthetic speech for Japanese. This work concentrated of using acoustic features in it selection [Iwahashi *et al* 93]. However [Campbell 92b], who exapnded NUUTALK for English, has long argued that prosodic features should be used in selection since even when units with good spectral characteristics are selected if they then must be modified by signal processing to correct pitch, duration and power differences, unnecessary distortion will be introduced.

Various attempts were made to attack this problem within CHATR both refining the original NUUTALK system, which was ported to CHATR, and introducing new selection systems which used prosodic features in selection. The original NUUTALK system was, unfortunately, written in a way far too specific to its task making modification for a different language and/or different selection features too difficult. In fact even the alternative unit selection systems were re-written a number of times until a general system was developed.

The current unit selection system is language independent. The features used in selection maybe prosodic or acoustic. Importantly the features used are no longer built in to the implementation and can be specified externally, allowing new features to tested without modifying C code—of course if that feature is to be predicted in synthesis some new code may be needed.

The current model used in unit selection describes units in a database as a collection of features. These features may be phoneme type, duration, pitch, syllable position etc. in fact any integer, float or discrete value. All units have a specific type, in all our current databases this is equivalent to a phoneme. When selecting a unit the target must be of the same type as the candidate unit from the database, thus phonemes may only be synthesized by selecting units of the same phoneme type. It should be added that as the definition of *phoneme* is user definable, these need not follow standard linguistic definitions. In fact some experiments with acoustically derived segments were made.

Given our representation of all units in the database as feature vectors, we similarly describe our target segments as a collection of features. We can then define a distance measure between a target segment and any candidate unit (of the same type) from the database. This distance is an estimation of the suitability of a candidate unit from the database being used to synthesize this target segment. The *target cost* is defined as a weighted sum of differences of each feature in the target segment and candidate unit.

$$C^t(t_i, u_i) = \sum_{j=1}^{p} w_j^t C_j^t(t_i, u_i)$$

The weights reflect the relative importance of the features (see later).

A second distance measure is defined. The *concatenation cost* is defined as a distance between two adjacent selected units. This reflects the cost of joining two selected units.

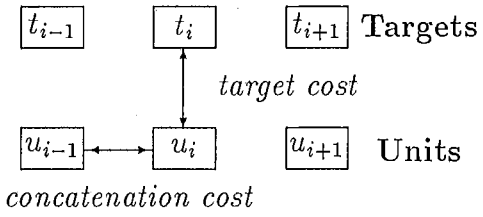
This is determined by the weighted sum of differences of each feature between the current candidate and a previously selected candidate.

$$C^c(u_{i-1}, u_i) = \sum_{j=1}^{q} w_j^c C_j^c(u_{i-1}, u_i)$$

In addition we have a further condition, if the previous selected candidate u_{i-1} and the current candidate u_i are actually adjacent in the database their concatenation cost is 0. This condition allows the selection of non-uniform units, in that selecting multi-unit strings from the database will be favored over disjoint ones (except of course when other factors overwhelm such selection).

Note that the feature set used in the target distance and concatenation distance need not be the same. In fact in all our databases we have settled on only three features in calculating the concatenation cost. These are local absolute F_0, local power, and a quantized MFCC vector to represent spectral features at the concatenation point.

The distinction between these two distance measures is illustrated in the following diagram



Given the above two distance measures we can define an overall distance for a string of selected units with respect to a string of target segments.

$$C(t_1^n, u_1^n) = \sum_{i=1}^{n} \sum_{j=1}^{p} w_j^t C_j^t(t_i, u_i) + \sum_{i=2}^{n} \sum_{j=1}^{q} w_j^c C_j^c(u_{i-1}, u_i) + C^c(S, u_1) + C^c(u_n, S)$$

The final two clauses define the end conditions of starting from silence and ending with silence, S represents a unit of silence.

The unit selection process can then simply be defined as finding the string of candidate units from the database that minimises the above expression.

Unit selection is done by a Viterbi search, which is restricted by a beam width to achieve real-time synthesis. [Campbell 94] first defined the above unit selection model while a more detailed description of the selection process described [Black & Campbell 95] but a later paper, primarily the work of Andrew Hunt, [Hunt & Black 96] gives a more formal account of this model and the training of weights.

The quality of synthesis produced by the above model, although potentially very effective, is very dependent on the weights for each feature distance in the two costs. Originally these

costs were hand tuned but it quickly became clear that optimal settings were not possible with only hand tuning. Also as we increased the number of features in selection, it became harder to tune the weights to take benefit from the new features.

[Black & Campbell 95] includes a description of our first method of automatic weight training. This method first made the assumption that a mean cepstrum distance between a string of selected units and a set of targets made from an original natural string would be smaller as the quality of the selection improved. At least in initial experiments this does seem to be true.

To find a set of weights, a range was selected for each weight and all possible combinations were used, and a string of units was selected. The weights and mean cepstrum distance from the target cepstrum vectors was recorded. This was done for a number of utterances. Then the best scores were collated to find the best weights. This method was to begin with reasonably successful. In that it would produce weights that on the whole were better than could be achieved by hand setting—though often hand tuning the trained weights could improve results. One major disadvantage of this technique was the amount of time required to train. For example testing three to five values for about 6 or 7 features require synthesizing the utterance around 100,000 times. Even on a fast workstation this took at least overnight. Normally around 10 different utterances were tested, requiring ten times the CPU time. More importantly as we increase the number of features the training time increases exponentially. An alternative method was sought.

Andrew Hunt devised a solution which is fully described in [Hunt & Black 96]. Again we use mean cepstrum distance in our measurement. Later after testing, this acoustic measure was augmented with F_0 information and a duration penalty. Instead of a "guess and test" strategy for weights we tried to predict the acoustic distance between each unit of the same type using linear regression from the feature based differences. The result offers a training mechanism which is more that one hundred times as fast and increasing the number of features only increases the training time linearly. The quality of the weights produced is about the same as that produced by the previous method. But it is now possible to train many databases fully automatically in reasonable time.

With this new method of training it became reasonable to try different features as retraining takes about one hour, even for a large database. We could also have different weights tailored to different phone types so for example allowing F_0 to have different weights for voiced and unvoiced phonemes. At this time we also reimplemented the distance function mechanism to allow for distance functions between arbitrary features allowing full training and testing of features without any change to the C code.

Thus our present unit selection code is language independent. It may use arbitrary features in unit selection and offers a training method for producing relative weights for those features. Speech synthesis voices may be built fully automatically from speech databases of waveform files and phoneme labels.

In a test of CHATR's flexibility a phonetically labelled Korean database was used. Within 12 hours, a Korean waveform synthesizer. No changes to the C code were necessary. The

quality of the synthesis was acceptable given the size of the database even though CHATR had dealt with no Korean synthesis before.

Note that the results are not quite as good as the above result suggests. The quality of the synthesis produced from a database does vary greatly, from near perfect to incomprehensible. The quality depends on a number of characteristics, some of which we have identified.

First the size of the database clearly affects the quality of synthesis, with the general maxim that larger is better. Secondly speaking style of the database is important. When we use a database built from isolated words, the resulting synthesis sounds like isolated words even though the units selected are sub-word units.

The quality of the labelling on a database is also crucial. Many databases have significant errors and it only takes one error in a phrase to cause the synthesis to be spoiled. For example our largest English database (f3a) comes from the Boston University FM Radio corpus [Ostendorf *et al* 95]. F3a consists of almost 100,000 units but its quality is much worse than f2b a database of only 40,000 units. The reason seems to be that f2b is much more carefully labelled that f3a.

Other databases have varying recording quality throughout so when concatenating speech from different parts, although joins are often good, the speech quality seems "muffled" in one phrase but clear in the next.

These issues point to a deeper problem with the current database building mechanism and unit selection in CHATR. Although for good clear databases we can build good synthesis voices for others we do not do so well. There is still much to do to investigate why this is the case and how we can automatically detect such poor quality, misaligned units, different recording quality and deal with it gracefully. The idea of pruning databases has been addressed [Campbell 94] but there is much in this area that would increase the overall quality of databases in general. This is related to the work on automatically labelling databases from orthography using aligning technology. Although we have attempted this the result still requires hand correction. Investigations are continuing in this area [Nishimura & Campbell 96], and we feel improvement will be gained if we can automatically detect (not necessarily correct) errors.

4 Current

The current released version of CHATR is 0.8. This version is significantly more stable than its predecessors as well as offering the widest range of voices, and highest quality of synthesis.

As it is easy to add new voices, the number of speech synthesis databases available to CHATR increases often however the standard system currently supports 21 databases for full text to speech:

9 English voices (3 male, 6 female)
13 Japanese voices (8 male, 5 female)

In addition a Korean database and some German databases are available, but do not yet support full text to speech.

Version 0.8 includes a completely newly written suite of programs to build and train new databases. From a set of waveform files and phoneme labels a synthesis database may be built and trained fully automatically in anything between 1 and 6 hours, depending on the size of the database.

In this version a completely new distance function mechanism was implemented offering a much more generic use of features in units. Also significant work was done on the higher level aspects of the synthesis process. English text to speech changed from barely acceptable to very acceptable for our core voices (particularly f2b from which many models were trained).

In this version for the first time we include generic trainable modules for duration and the prediction of F_0 contours ([Black & Hunt 96]) which have been used, with good results, for multiple languages. This adds to our claim of CHATR being both generic and multi-lingual.

Over the past three months significant effort has been put into making the CHATR system more stable. Many memory leaks were fixed and work was done to optimize the speed, removing unnecessary code from the system. Although the system is written in ANSI C the implementation of this language in actual compilers is unfortunately not as precise as the standard itself. Testing on multiple machines with different ANSI C compilers has both fixed problems in the code and made it more likely that porting to other architectures will be trivial. An important addition to the list of supported machines is the DEC Alpha which offers a 64bit architecture. CHATR has been fully tested on this machine such that as other vendors offer 64bit machines (i.e. Sun) no further changes to the system need be done.

The installed version for 0.8 in ITL currently supports the following architectures

— SunOS 4.1.[34]
— Solaris 5.5
— DEC Alpha OSF1 V3.2
— HPUX A.09.05 for hppa
— SGI IRIX5.3 for mips
— FreeBSD 2.1 for iX86
— Linux for iX86

Although not all these machines currently exist within ITL, they are the machines that are standardly used in research throughout ATR as a whole.

If CHATR is truly to exist as a standard tool for speech synthesis, both as a black box text to speech system and as a programming environment in which new synthesis modules can be developed, it must be stable on multiple architectures.

5 Future

The current version is just a step of CHATR's full life. There is much that can be improved, and the system although stable, should not be considered in anyway complete. It must be continually developed if it is to remain useful.

For many years, Nick Campbell has argued for the creation of synthesis systems using databases of natural speech [Campbell 92a], [Campbell 95]. CHATR is a major step to realising that goal.

Can CHATR create a synthesized voice for any speaker? The answer to this seems to be yes, but we still have more work to do. The waveform synthesizer using unit selection from speech databases successfully creates a voice in many cases, though sometimes the quality can be poor. The problem seems both in the labelling of the database, the quality of the recording, and still in the unit selection itself. Given a well-recorded, well-labelled, reasonably sized database the quality seems reasonable. But work is still required to be able to take databases that are not well-recorded (varying acoustic quality), and bad labels and be able to prune them (automatically) successfully. But the waveform synthesizer is only part of the synthesis system.

We already have on trainable duration and intonation modules in CHATR. But they require prosodically labelled databases. Work on automatically labelling databases has started but has not yet reached the stability of other parts of the system. However given, prosodic labels (i.e. ToBI labels), word, syllable (with lexical stress) , phoneme labels, CHATR can automatically extract information from which duration and F_0 prediction modules may be created. This has successfully been done for the f2b database using linear regression as the method (other methods may be more appropriate, but getting the data in a uniform automatic way is the difficult part).

Other aspects of a speaker's characteristics can be parameterized within CHATR including the lexicon (for different dialects), vowel reduction, pause prediction, accent placement and phrasing. However these could not yet be considered as truly general or automatically derivable from a database. These aspects do however contain speaker characteristics and it would be good to be able to parameterize these modules automatically as part of the standard database building process.

Can CHATR build a synthetic voice for any language? This is of course much harder but of course a much more interesting question. To some extent we have already tried to reach this goal. The waveform synthesizer from unit selection from a speech database is language independent. The test with a completely new language (Korean) was successful, further tests with a German database also confirm this. However, there are still some issues though about which features are the most appropriate to use in unit selection. For example tonal languages like Chinese may require better representation of F_0 in unit descriptions but it does seem that the basic framework of unit selection would in general be adequate.

The higher levels would of course require work. We already have mutli-lingual duration and F_0 prediction modules (given a ToBI like intonation labelling). Though different features

would of course be required for different languages. It does seem that being able to build any synthesizer for a new language without writing new C modules is probably outwith the scope of the current CHATR system. But it does seem feasible that CHATR could be developed further such that it does offer an environment where synthesizers for new languages may be created without much new work.

6 Problems

It is important to realise that CHATR is not finished. It has been developed so that it has provided a working system at any time but also much of it has been replaced as better techniques and better understanding of the problems it is addressing come to light. Hopefully the core architecture of the system is relatively stable but much of rest should be considered for reimplementation, at some time in future. Good code evolves and should rarely be considered complete.

Having said that there are specific points within CHATR that are now ripe for reconsideration. The reasons are varied, partly that in order to get something to work, a quick incomplete implementation is all that there is time for, and partly due to the growth of the system, a module becomes too complex and should be replaced or restructured. This section unashamedly admits the areas in CHATR which will require specific attention in the near future.

CHATR allows the definition of arbitrary phoneme sets. Each phoneme in a set is currently identified through a number of features: vowel/consonant, vowel height, length, frontednes, roundness, consonant type, place of articulation, voiced/unvoiced. These features are used in various places within the system and are useful in clustering groups of phonemes together (and offering some degree of phoneme independence). The problem is that when new languages are imported into CHATR the current set of features and/or their values is not general enough. A hacky solution to this is to edit the C code and add new values but this is not in the spirit of the CHATR system. A better solution is to allow the specification of phoneme features and their values so that new features and values may be specified externally. There is already a structure within CHATR for the representation of discrete values which is a good basis for this new system.

CHATR's Lisp was written specifically for CHATR but it is not a complete implementation. Specifically it does not include a full garbage collector. Utterances do contain reference counts and are automatically garbage collected when there are no longer any pointers to them. Utterances are by far the largest objects and care has been taken in the system to ensure that no memory is leaked. However Lisp commands themselves can leak. It is not possible in general to know when a returned List object is no longer referenced. There are number of well-known garbage collection techniques. I feel the most suitable for CHATR is a reference count of cons cells. However implementing this is not easy. In order for references to work it is necessary that changes in pointers referencing cons cells increment and decrement a cell's reference count as required. Currently most references are done through simple

C assignment and would not allow this check to occur. If all references and dereferences went through a special set C function or macro the reference counts could be kept up to date and garbage collection could happen, but that would require a lot of small changes throughout the code. Another solution is to move to C++ where the assignment operator can be overloaded, if the List C structure became a C++ object. Then reference counts would be easier and require fewer changes throughout the whole code.

It should be noted that during standard text to speech (including switching between speakers) no memory leaks occur. So the garbage collection problem is not very serious. Even in the interpreter when leaks occur they are mostly small so often will not be serious. However, leaks do exist and that is a bug.

The HLP module which concerns itself with translating "syntactic" trees to prosodic phrase trees and predicts accent placement, pauses, prosodic boundaries is in somewhat of a mess. Although originally designed to be language independent its basic structure is now rather old and it would greatly benefit from a complete rewrite. Accent position is, for example, assigned in a number of places and depending of the intonation method selected may even be done more than once (the later one overwriting the previous). This is bad programming.

A complete rewrite of the HLP module to fully accommodate Japanese (and other languages) would be worth while. The module should take a syntactic structure tree as input with each word labelled optionally with part of speech (in a standard defined way) and include standard defined ways to specify IFT (speech act), focus, prominence (and other as yet unspecified markings). It should predict prosodic phrase boundaries, and accent position in an intonation method neutral way. The later intonation module should realise this information as actual accent (and tone) types. This is partly true at the moment but it is not at all clear (too many interacting subsystems). A full rewrite is worth while, and at the same time the inclusion of Japanese phrasal prediction (and pauses) would be advantageous.

At a lower level all of an utterance's information is held in a set of streams, for words, syllables, phonemes etc. Various relations exist between cells of different levels allowing access to the syllables in a word and the word a syllable is in. This structure is powerful and in the most part good. The main problem is that it it difficult to know at what time in the synthesis process which relations exist and if values within stream cells are set. There is not an easy solution to this. One way to go is to completely redesign the stream cell so this information is more explicit, another it to require modules to explicitly state what information they require and what they provide so that when in a module you can guarantee that specific information exists. This is not a problem if you are willing to look at the structure itself and trace the code to find out what is set when, but it is already the case that developers add new fields to cells to hold information which actually already exists elsewhere.

Many have complained about the speed of CHATR both in its start up time and its synthesis time. CHATR can synthesize speech in about 60% of the time it takes to say it on a fast machine (i.e. SPARCStaion 20). However if it is to say a ten second utterance you must wait 6 seconds before you will hear the first noise. That is a long time. However we have

to make sure we know the reasons for this delay. First it is now known that the DATLINK (a common output device) adds at least 2 seconds of silence in front of any waveform being played. This is a fundamental aspect of the controller device, using alternative hardware would solve this problem. As for speed of synthesis itself, in tests, 60% of the synthesis time is in accessing the files containing the waveform segments. The limit here is *not* the CPU speed of the machine but the speed of the disk access and (in most cases) file access across the network. Note that our Ethernet, only allows transfers of about 800K bytes per seconds, and that is when no one else is using the net. Local disk access (on Suns) is about 1.8M bytes per second. But add to the read time, the time to open a file (which takes longer than to read it) and the fact the automounter may have unmounted a disk, then the synthesis time can sometimes be very slow.

Note that on an Alpha the synthesis time reduces to less than one fifth real time. If phrase by phrase synthesis is used complaints about time are simply unfair. But on a SPARCStation 20 accessing remote disk over a heavily used network, and playing through a slow audio device, speed is a problem.

To some extent this problem will be solved through time as machines get faster. A faster network (100MBit Ethernet) is a start and better disk access will help.

The start up time for CHATR (and when it has to load in a new speaker) may also be significant. When a new speaker is accessed it must load in the index which contains the basic index, acoustic information for each (10ms) frame of the database and pitch marks. This can easily be a few megabytes for our normal databases. Given that our best Ethernet throughput is only 800K and for example f2b's index is 5.6M, it means it cannot take less than 7 seconds to load this database.

Possible solutions to this problem have been suggested such as preloading database indices and using "unexec" to save versions of CHATR. Also shared libraries have been suggested. All these will do is save the time required to unpack the information in an index (mostly trivial) but CHATR will still need to read that data. There is the possibility that not all of an index need be accessed at once and hence "unexec" or a shared library would be quicker than actually explicitly loading it. But I feel that we access most of our databases in near random fashion (at least we should) and hence there seems little point in making databases pre-loaded, as the bottle neck is still the reading of the information.

Two solutions are possible, either increase the speed of the access or reduce the amount of information that has to be read. The first solution is easiest as it requires no program changes, but would restrict CHATR's use to fast network machines. The second requires more work but in the long run should be considered. The question is how much of a database do we use? And more importantly how much must we use? Although some investigation into pruning a database has been made it still would benefit from more work. It appears we can prune out almost half of f2b's units without making a difference to the synthesis quality. Work has started in optimising the number of units in a database. Halving the size of a database (in the right way) could reduce the load time by half.

CHATR's Lisp system does not allow a partitioning of the name space. It would be useful

if we could have a notion of local variables for each speaker. That is, when a speaker is selected we move into a local name space so variables may have local names (but still access global ones). This would be similar to the way "buffer-local" variables work in EMACS. If this were the case, speaker variables could be set without the possibility of altering the operation of other speakers currently loaded into the system. It would be fairly easy to add this to the current Lisp system and then change all the speaker definitions to conform to this new system.

Debugging a synthesis method without getting deep into the C code is still a little tricky. For some methods resorting to gdb and the C source is the only way to do this. To be fair this problem is partly due to CHATR's own success. As it is possible to do investigations and experiments in the CHATR's Lisp system, it would be useful if this could be done with debugging aids. At least a backtrace of where the error occurred would be useful. Adding a backtrace (or even a simple trace option like other Lisps) would be possible. But there is the problem of tracing internally called functions as well as those that are called through the interpreter. A trace function could be written in Lisp as is, but minor changes would be required to allow the shadowing of Lisp functions which are builtin C functions. Tracing internally called functions is harder though maybe not impossible as tracing only key utterance modules is probably sufficient.

A better text to speech front end system would be good within CHATR. Although we have always said that CHATR is (primarily) not a text to speech system. Realistically text to speech will be how most people actually use the system. There are many simple text to speech problems which could be fixed without much work and they would significantly improve the quality of the speech output. This should be done for both English and Japanese (and any other language CHATR supports).

It is also important to stress that having many users of a system can significantly improve the quality of a program. Listening to their needs often (though not always) improves a system. Currently CHATR really has very few users, even for text to speech. Making it easier for people to use is one thing e.g. having a program that automatically figures out what audio device a person has will help but also the speed and quality of the generated synthesis is important. Also the system that it is embedded in will help make it more interesting to use. In addition giving users the feeling that they are contributing to CHATR when they report bugs, and that their bug reports are dealt with promptly can quickly create a sound body of users that will help to improve CHATR as a generic speech synthesis system.

7 Conclusions

In many ways CHATR is a success, it has shown that non-uniform unit selection for multi-languages is a realistic way to do speech synthesis. Of course many believed this before but CHATR actually shows this. CHATR also offers a practical text to speech system which is already being used outside the group for synthesis in other projects.

For the most part, the work I have contributed is the implementation of existing theories

of speech synthesis. It can be argued that there is no new research in CHATR. And I think that is a valid statement, but with some caveats. The implementation of ideas always leads to highlighting shortcomings in the original idea. It is almost impossible to fully specify a concept without producing something that is as finely specified as a program—not to say that all programs are necessary correct implementations of ideas. Likewise in the implementation of ideas which have been experimented with before, those ideas have typically had to be filled out and made more explicit. For example the concept of unit selection existed before, even that of using prosody in selection, but when these were brought together in CHATR it became necessary to formalise this concept further. The concepts of *target cost* and *concatenation cost* were not as explicit even in previous implementations of these ideas. Likewise the work on training of weights for features was not such an issue before as previous implementations were not as versatile and survived with hand tuned weights.

Implementation, in a research field such as speech technology, should not be simply regarded as simple job that unskilled programmers can carry out in the last part of a project. Implementation is inherently part of the research cycle. It provides a realisation for ideas. Even one-of, personal implementations of research should be regarded only as a first stage in the use of ideas in systems. Further, more robust implementations, will often show problems, often solvable ones, but without the full stable implementation problems with ideas will not be fully discovered, and the full benefits of ideas will not be available to anyone else.

It is attributed to Isaac Newton that he once said that he could see further because he stood on the shoulders of others. CHATR is not itself a system that can see further but it offers, I hope, a stable pair of shoulders on which others may stand.

References

[Anderson *et al* 84] M. Anderson, J. Pierrehumbert, and M. Liberman. Synthesis by rule of English intonation patterns. In *Proceedings of ICASSP 84*, pages 2.8.1–2.8.4, 1984.

[Black & Campbell 95] A. W. Black and N. Campbell. Optimising selection of units from speech databases for concatenative synthesis. In *Eurospeech95*, volume 1, pages 581–584, Madrid, Spain, 1995.

[Black & Hunt 96] A. Black and A. Hunt. Generating f0 contours from tobi labels using linear regression. submitted to ICSLP96, 1996.

[Campbell 92a] N. Campbell. Syllable-based segmental duration. In G. Bailly and C. Benoit, editors, *Talking Machines*, pages 211–225. North-Holland, 1992.

[Campbell 92b] W. N. Campbell. Synthesis units for natural English speech. Technical Report SP 91-129, IEICE, 1992.

[Campbell 94] N. Campbell. Prosody and the selection of units for concatenative synthesis. In *Proc. ESCA Workshop on Speech Synthesis*, pages 61–64, Mohonk, NY., 1994.

[Campbell 95] N. Campbell. Mapping from read speech to real speech. In *ATR Workshop on Computational modeling of prosody for spontaneous speech processing*, pages 3–20–3–25, ATR, Japan, April, 1995.

[Hunt & Black 96] A. Hunt and A. Black. Unit selection in a concatenative speech synthesis system using a large speech database. to appear in ICASSP96, 1996.

[Iwahashi *et al* 93] N. Iwahashi, N. Kaiki, and Y. Sagisaka. speech segment selection for concatenative synthesis based on spectral distortion minimization. *IEICE Transaction Fundamentals*, E76-A:1942–1948, 1993.

[Nishimura & Campbell 96] K. Nishimura and N. Campbell. A method for auto-labelling speech databases for speech synthesis. Technical Report TR-IT-0154, ATR Interpreting Telecommunications Research Laboratories, 1996.

[Ostendorf *et al* 95] M. Ostendorf, P. Price, and S. Shattuck-Hufnagel. The Boston University Radio News Corpus. Technical Report ECS-95-001, Electrical, Computer and Systems Engineering Department, Boston University, Boston, MA, 1995.

[Pierrehumbert & Beckman 88] J. Pierrehumbert and M. Beckman. *Japanese Tone Structure*. The MIT Press, Cambridge, Mass., 1988.

[Sagisaka *et al* 92] Y. Sagisaka, N. Kaiki, N. Iwahashi, and K. Mimura. ATR – ν-TALK speech synthesis system. In *Proceedings of ICSLP 92*, volume 1, pages 483–486, 1992.

[Silverman *et al* 92] K. Silverman, M. Beckman, J. Pitrelli, M. Ostendorf, C. Wightman, P. Price, J. Pierrehumbert, and J. Hirschberg. ToBI: a standard for labelling English prosody. In *Proceedings of ICSLP92*, volume 2, pages 867–870, 1992.

8 Publications

The following is a list of all publications I was involved with during my stay at ATR.

Black, A. and Hunt, A. 1996. "Generating F0 contours from ToBI labels using linear regression" Submitted to ICSLP96, Philadelphia.

Hunt, A. and Black, A. 1996. "Unit selection in a concatenative speech synthesis system using a large speech database" Proceedings of ICASSP 96, Atlanta, Georgia.

Campbell, N. and Black A, 1996, "Prosody and the selection of source units for concatenative synthesis", in "Progress in speech synthesis" eds. van Santen, J., Sproat, R., Olive, J. and Hirschberg, J. Springer Verlag.

Black, A. and Campbell, N. 1995. "Optimising selection of units from speech databases for concatenative synthesis" Proceedings of Eurospeech 95, vol 1, pp 581-584, Madrid, Spain.

Black, A. and Campbell, N. 1995 "Predicting the intonation of discourse segments from examples in dialogue speech", Proceedings of the ESCA workshop on Spoken Dialogue Systems. Denmark.

Black, A. 1995 "Predicting the intonation of discourse segments from examples in dialogue speech", ATR Workshop on Computational modeling of prosody for spontaneous speech processing. ATR, Japan. Also to be published in "Computing Prosody: approaches to a computational analysis and modelling the prosody of spontaneous speech" eds Sagisaka, Y., Campbell, N., and Higuchi, N. Springer Verlag 1996.

Black, A 1995 "Comparison of algorithms for predicting accent placement in English speech synthesis", Spring meeting of the Acoustical Society of Japan.

Black, A and Taylor, P. 1994 "Assigning intonation elements and prosodic phrasing for English speech synthesis from high level linguistic input", ICSLP94, Yokohama, Japan.

Taylor, P. and Black, A. 1994 "Synthesizing Conversational Intonation from a Linguistically Rich Input", Proc. ESCA Workshop on Speech Synthesis, Mohonk, NY.

Black, A and Taylor, P. 1994 "CHATR: a generic speech synthesis system", COLING94, Kyoto, Japan.

Black, A and Taylor, P. 1994 "A framework for generating prosody from high level linguistic descriptions", Spring meeting of the Acoustical Society of Japan.

Black, A. 1993, "Some different approaches to DRT", DYANA-II deliverable, R3.2.

Black A, 1993, "Using Situation Theory in a computational language for natural language processing", 4th Natural Language Understanding and Logic Programming Conference, Nara, Japan.

9 CHATR revision history

chatr-0.8 Mar 08 1996

Ports to SunOS, Solaris, HPPA, SGI, Linux, FreeBSD and DEC Alpha checked substantial ansification done
Language independent duration model using linear regression examples for Japanese and English
Manual completely overhauled (200 pages)
Substantial checks to all files for copyright problems and tidy up
Fixed another round of memory leaks and optimizations
Made NUUTALK and formant synthesizers optional subsystems
F_0 prediction by linear regression from (J)ToBI (plus alpha) same method for Japanese and English
New database building software, fully automatic and does training using new distance mechanism
New distance function definition mechanism, allows external specification of arbitrary distance functions.
Pause prediction by decision tree
Boundaries (in English) tts predicted by decision tree
Linear regression model for duration
Pruning of databases based on distance measures
Syllable reduction predicted from decision tree
ToBI prediction by decision tree
Save supports XUnits, XSegs, XBreaks, F0, Tones, Words
floats are held as floats in Lisp, no longer as strings.
Tidy up of xwaves interface, now supports display of F_0.
Start at phrase by phrase synthesis and alternative silence generation techniques
Better tuned Beckman JToBI module
Linear regression training method for weights
11 new Japanese voices
Built new duration nnets
Added support for sylpos feature in udbs
Redid Lexicon module: supports multiple lexicons (BEEP and CMU)
Method for dumping udb selection stats (for beam width tuning)
Restructured unit selection distance functions
Introduced PhonoForm, more detailed linguistic specification of utts, for better extraction of information from a database.
Removed taylor and isard diphone sets as standard (they are optional though, but distribution does not include them by default)

chatr-0.7 Aug 31 1995

A working psola

chatr_pipe client program, text in/waveform out.
Can use cepstrum params in targets to find "best" units for natural utterance (and also for joins)
Korean speaker (*NO* changes to C or Lisp system required !!!)
New databases, mlp350, fmp559 (bilingual)
Documentation updates (htmlable)
Substantial bug fixing and tuning of unit selection
An implementation of the Fujisaki model for Japanese intonation
Multiformat cepstrum support
new PSOLA from Christian
New databases (with tuned weights: wnc600, sab600, gsw200)
Start at scripts for building new databases
Better join measures for unit selection (vq, pros and phone)
ToBI input method and F_0 generation (and emmi-based prediction of labels)
Updated dependent libraries, nist-sphere-2.5, readline-2.0 and nas-1.2p1
Allow pitch marks to be preloaded (and increased accuracy)
Various renaming in lib directory (separate code from dbs)
f2b based nnet duration module
Cepstrum measures for tuning unit selection
Neural Net support (for duration)
usable sally (200) generic udb synthesizer
Yet another generic udb module (with MHT Bset db), as good as nus.
Feature functions
Separate out Japanese modules better (Kaiki duration, JLTS)
Beckman intonation module (JToBI)
Documentation updates
BSD socket server mode (telephone email reader demo)
Documentation strings for user functions, and variables
Wave_filters, low pass, high pass, chorus, backwards etc.
Tilt accent type predicted by decision trees (Radio data)

chatr-0.6 Nov 17 1994

restructuring of CHATR lisp code library files
F2b FM Radio database
Japanese female database(s) (NUUTALK)
Improved EMACS interface
Substantial bug-fixing and addition to tilt labelling code
TTS modes through escape commands
cepstrum resynthesis for standard units
require and provide functions
Reintegrated Nick's superunit selection strategy as a generic database selection stratgey (NUS)

chatr-0.5 Aug 15 1994

Vector quantization for NUUTALK unit selection (much faster)
Initial Japanese TTS support (romaji and using kakasi/KDD_kan2rom)
Added Discrimination Tree support, can be specified in lisp and changed at run time
Another pass to remove memory leaks in the Lisp part of the system
Added syllable structure to letter-to-sound sub-system
Added basic arithmetic functions
Monaghan prosodic prediction (can choose strategy)
Moving unit boundaries (vis cepstrum distance measures) available as lisp function
Choice of waveform resynth method in NUUTALK (PSOLA/NUUCEP)
much faster NUUTALK (with binary dumps of DBs)
Stream type: list, tree, (and some generic functions to traverse them)
UDB code: selective exclusion of units
vowel reduction in destressed words, word contractions

chatr-0.4 May 11 1994

NUUTALK loadable low-level databases
new (faster) lisp reader
NUUTALK access from segment stream
updated documentation
X windows utterance inspector
CSTR diphone synthesizer
tts stdin input mode
Persistent history
NUUTALK initial port
cepstrum parameter cost function, and support
Faster PSOLA
fixed many memory leaks (many major paths have *no* leaks)
utt_modules from Lisp (and Lisp language enhancements)
builtin sample rate converter
Hirschberg accent assignment algorithm
the Tilt theory of intonation
multiple speakers for intonation system, stats collected on cmu data.
syllable input mode(s)
audio spooler (can synthesize next utterance while playing previous waveform)
Added print and load structure code (some streams defined)
Many more core Lisp functions added (Lisp interface now allows functions, loops, map, apply, eval etc.)
Made lisp structures honest objs (streams, utterances, functions)
Bachenko et al style prosodic phrasing
many malloc bugs fixed (now can compile with -0)
lts and text (file) input method
pisel udb selection strategy
multi-unit segments, (selection and PSOLA)

chatr-0.3 Dec 28 1993

slightly improved PSOLA, fast udb code, phomeme structure rehash, made system compile with g++.

chatr-0.2 Dec 9 1993

Major code restructuring, HLP module, demos, udb code, PSOLA, multiple phoneme sets, intonation features parameterized.

chatr-0.1 Oct 22 1993

Basic isard diphone synth and first unit selection code.

chatr-0.0 Sept 3 1993

First version change, formant synth and basic architecture

CHATR first spoke (incomprehensibly) April 28 1993

"Her friends considered her beautiful."

CHATR work started Apr 10 1993