第 9 部

マルチメディア (音声インタフェースの取り扱い)

第 1 章

序論

従来人間とコンピュータの間のインタフェースは、文字表示と打鍵によって成り立っていた。しかしコンピュータの速度の向上や技術の進歩のおかげで、もっと副次的であるグラフィックスや音声といったインタフェースも利用することができるようになった。こうした文字だけでない様々な形態によるインタフェースのことをマルチメディアと言う。その中でもグラフィックスによるインタフェースは古くから研究されており、現在では利用方法もほぼ統一されてその資源も管理されているために充分実用に耐えるものとなっている。しかし音声によるインターフェースは、まだ利用できるようになったばかりでその利用方法や管理などには様々な問題が存在している。この論文ではこの音声によるインタフェースについて考察する。

音声によるインタフェースを考えた場合、まず音声を意味するデータと音声とを相互変換することが考えられる。また人間が読むことのできる形態でコンピュータ上に蓄えられている情報を音声に変換することと、その逆変換といったものも考えられる。これらのインタフェースのうち前者を音声入出力と呼び、後者を音声合成、音声認識と呼ぶ。ただしこの論文では、音声認識については論じていない。

音声入出力を用いることによって新たな利用形態である、ユーザ間での音声による交信や保存しておいた音声を再生するといったこと、また入力した音声を変換／編集して再生するといったことが可能になる。特に音声を使用する価値があると思われるのが、メイルが届いた場合やエラーが発生した場合などという突発的な出来事が発生した場合に、保存しておいた音声メッセージを再生して注意を引くという利用形態である。

また音声合成を用いることによって、音声出力で使用するための音声データをコンピュータ内部の情報から生成できる。つまり全てのメッセージを音声化して出力することができるようになる。そのため音声合成と音声出力を組み合わせて使うことにより、音声によるインタフェースが飛躍的に便利になる。利用方法としては、メイルを読み上げるといったものから、複雑なエラーメッセージを読み上げるといったものまで様々なものが考えられる。

しかしこうした音声によるインタフェースにはまだまだ問題点があり、有効に利用するためには、それらの問題点を解決しなければならない。問題となるのは、まず音声入出力を行うための装置や音声合成を行うための装置が管理されておらず、利用方法にも数々の方法があり統一されていないという点である。また音声をコンピュータ上やネットワーク上で取り扱うための形式が統一されていないという点である。

実際に用意されている音声入出力を行うための装置や音声合成を行うための装置、またそれらの装置における問題点としては以下に挙げるようなものがある。

NEWS や Sun をはじめとして多くのワークステーションで音声入出力デバイスを装備している。さらに、音声出力デバイスを用意している X ウィンドウ端末まで登場してきている。これらの音声入出力デバイスを利用することによって、ワークステーション外部のアナログデータである音声と、内部のデジタルデータである音声データと相互に変換し利用することが可能となる。

ただしこれらのワークステーションに音声入出力デバイスが装備されているといっても、いずれの場合も単にハードウェアとデバイスドライバが用意されているにすぎない。また音声入出力デバイスに対するアクセス方法もワークステーション間で統一されていない。

例えば、NEWS では`/dev/sb0` または`/dev/sbe0`[89] を、Sun では`/dev/sound`[90] という名前のデバイスファイルを `open` し `read/write` システムコールによって読み書きすることで、録音/再生が行えるようになっている。これらの機能を直接使うことによって、取り敢えず音声入出力を行うことができる。しかし機械によってデバイスの構成が違うため、プログラムごとにそれぞれのデバイスに対応した変更が必要になる。またこれらのデバイスに、同時に複数のプロセスから読み書きを行った場合には、様々な混乱が生じてしまう。[1]。

また、これらの音声入出力デバイスは A/D-D/A 変換器で構成され録音と再生を行なう単純なものであるが、漢字仮名交じり文を入力するとそれを音声に変換する音声合成装置も存在する。この装置は、音声データとして保持されていない文章を音声として出力したい場合などに有効に利用できる。しかし現状では、このような装置は高価であるためすべてのワークステーションに用意することが困難な状況にある。

しかし現状でもネットワーク上に音声合成装置を用意しておき、それを複数のワークステーションからネットワークを介して共有することは可能である。つまりネットワーク上のどこからでも受けることのできる音声合成というサービスが提供できる。しかし現状では音声合成装置ごとにその利用方法が異なる、同時に利用できないなどの種々の問題点がある。

このような混乱を取り除き、本当の意味で音声入出力デバイスを効果的に利用するためには、また音声合成装置を共有し有効に利用するためには、統一的なインターフェースを持った音声入出力デバイスを管理する機構や音声合成装置を管理する機構が必要である。本論文ではこれらの問題点を解決するために、Unix の環境に適した音声入出力デバイスを管理するための機構および音声合成装置を管理するための機構を設計する。

[1] NEWS では、同時に複数のプロセスが音声入出力デバイスをアクセスしようとするとブロックしてしまう。また Sun では、同時に呼出されたシステムコールがランダムな順序で順次処理されるため、出力される音声は混合されたものとなる。

第 2 章

現状および問題点

ワークステーション上で音声を取り扱うためには、まず音声とワークステーション上の音声のデータを対応させることができなくてはならない。それには、音声をワークステーションが取り扱うことのできる形態である音声のデータに変換すること、そしてその逆変換が可能である必要がある。本論文では、音声をワークステーション上で取り扱えうことのできる音声のデータに変換することを入力と言い、その逆変換を出力と言う。

しかし一口にワークステーション上で取り扱うことのできる音声のデータと言っても、その中には様々な形式のデータが含まれる。こういった音声のデータが取り得る形式としては、大きく分けて表 2.1に挙げる 3 つの形式があると考えられる。

表 2.1: ワークステーション上で取り扱うことのできる音声のデータ形式

1. **音声の波形を表すデータ**

   音声をサンプリングして A/D 変換して得たデータ、またはそのデータを圧縮したものである。

2. **音声の波形を記述するための言語で記述されたデータ**
   波形に名前を付け、その名前を集めて複雑な波形に対応付ける。楽譜や発音記号などもその一種である。

3. **平文**

   音声合成／認識することによって音声と対応付けることができる日本語や英語といった言語で記述された文章である。

この論文では、表 2.1に挙げた中の音声の波形を表すデータによる音声の入出力のことを “音声入出力” と言い、平文を音声合成することによって音声を出力することを “音声合成” と言う。また音声入出力に用いる音声のデータのことを “音声データ” と言う。

まず音声入出力デバイスと音声合成装置の管理機構について論じる前に、ワークステーション上で取り扱うことのできるこれらのデータ形式について、現状と問題点を考察する。その後音声入出力を行う場合について、また音声合成を行う場合について、それぞ

れ現状と問題点の考察を行う。

2.1 ワークステーション上で取り扱うことのできる音声データ

表 2.1に挙げたデータの形式の中でも最も基本的な音声のデータは、音声の波形を表すデータである。これは、この形式のデータがアナログの信号である音声を直にデジタル化したもので、音声とほぼ一対一に対応していることによる。

これに対して音声の波形を記述するための言語で記述されたデータでは、音声データ中に現れる音声の波形とデータの値とを対応付けている。例えば音楽を取り扱う楽譜の例では、A が 440Hz の波形に対応付けられる。

これら 2 種類の形式データでは、音声が最初にあって、その音声をそれぞれのデータに対応付けるという方法論によってデータの値が決められている。しかし表 2.1の最後に挙げた平文という形式のデータでは、最初に平文である文章があって、それをその文章を人間が読み上げた時の音声に対応付けている。

このように表 2.1において挙げた、ワークステーション上で取り扱うことが可能な音声の形式は実際は階層化されて取り扱われる。つまりすべての音声のデータが直に音声と相互変換される訳ではなく、図 2.1に示す様にそれぞれの形式間で形成されている階層を介すことによって相互変換されるのである。

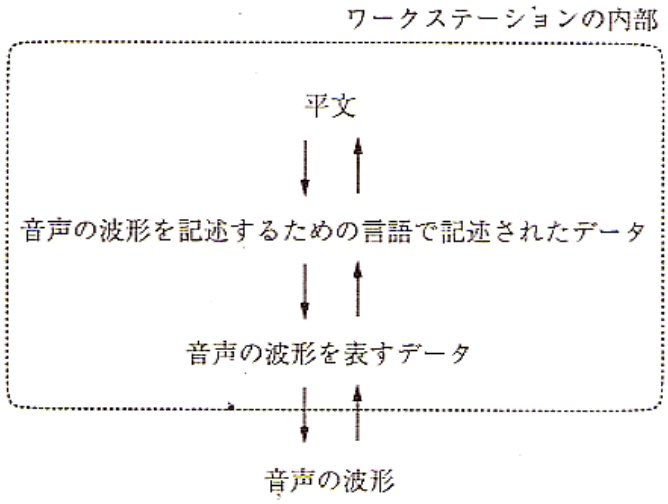

図 2.1: 音声の入出力の様子

例えば、音声を表 2.1に挙げた平文という形式の音声のデータとして取り込むためには、図 2.2に示す様に音声をまず音声の波形を表すデータとして取り込み、それを波形認識して語と語に分解して発音記号などに変換し、意味の解釈を行う必要があるのである。

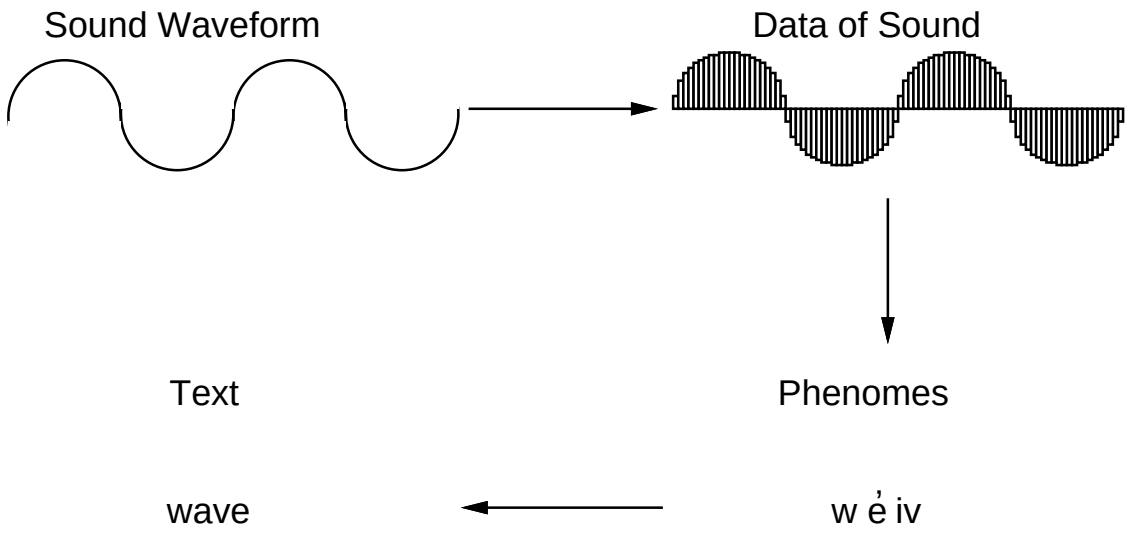

図 2.2: 音声を平文として取り込む様子

以降ではこの論文で取り扱う音声のデータ形式である、表 2.1中の音声の波形を表すデータと音声合成するための平文について考察する。

2.1.1　音声入出力に関するデータ

音声とは連続的に進む時間とともにその値も連続的に変化するデータであり、一方ワークステーション上で取り扱うことのできるのはバイナリデータだけである。そのためワークステーション上で音声と取り扱うには、音声をバイナリデータに変換しなければならない。これが音声入力であり、この逆が音声出力である。

音声入力は、まず音声を一定の間隔で標本化することから始まる。こうして得た音声の振幅値を量子化してバイナリデータを得る。こうして音声をワークステーション上で取り扱うことのできるバイナリデータである音声データに変換するのである。その様子を図 2.3に示す。

またこの逆の操作をすることによって、ワークステーション上の音声データを音声に変換することも可能である。

しかし、この音声データと音声が完全に一対一に対応しているわけではない。サンプリング周期の間のデータは完全になくなる上に、量子化誤差も存在する。このようにサンプリング時や A/D 変換時に情報が欠落しているので、音声データから音声を復元し、その音声を再入力するといった過程を繰り返すと情報がますます欠落していくことになる。

ただし一度デジタル化された音声データの複写においては、その過程での情報の欠落は発生しない。つまりネットワークを介して他の場所に送ろうとも、またファイルにして保存しておこうとも、この音声データから出力することのできる音声は基本的には同じものである[1]。

[1]再生系によって音質が変化することもあるだろうが、その意味で同じというわけではない。

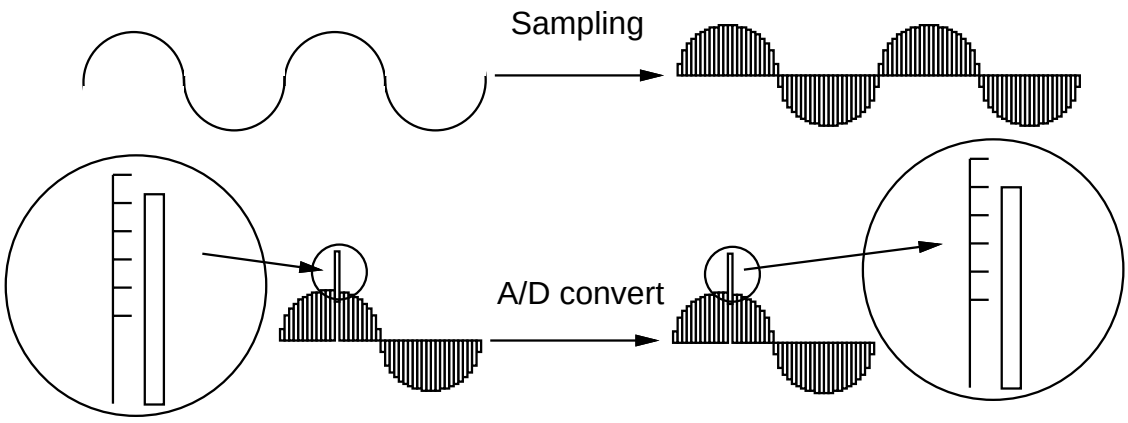

図 2.3: 音声入力の様子

標準的に電話で用いられている音声のデジタル化は、サンプリング周波数 8kHz で 8 ビットに符号化されたものである。ということは、こうした音声データは 64kbps のデジタル回線を必要とする。

しかし、単純にデジタル化しただけの音声データでは量が膨大であるため、普通は圧縮を施した後のデータを音声データとして扱う。こうした符号化方式の中でも現在一般的のものを表 2.2と表 2.3に挙げる。

こうして膨大な量の音声データを圧縮する反面、音質の良さを求めるという要求もある。音質を求めるということは、サンプリング周波数や量子化ビット数を上げるということで当然それにつれて音声データの量は増加する。しかし、元々人間の感じ取れる音質の上限があるため、過剰な音質を提供する必要はない。

例えば、サンプリング周波数なら人間が聞きとることのできる周波数が 20kHz 程度なので、それを再現できる 40kHz 程度のサンプリング周波数が上限になる。また量子化ビット数は、実用上信号対量子化雑音比 (S/N 比) が 66dB 程度あれば足りるため、12 ビット程度が上限になる [91]。

2.1.2 音声合成に関するデータ

2.1.1で既に述べたように、音声の入出力を行うことは可能である。しかしそれだけでは一度取り込んだ音声以外は出力できない。したがって、音声を取り込む以外の方法によって音声を生成するということも重要である。

音声を生成するための方法として、最も標準的な方法が音声合成である。この音声合成を利用することによって、平文を音声または音声データに変換することが可能になるのである。

それには、与えられた平文の構文解析をして語と語に区切り、辞書を引いてそれぞれの語句に対応した音声を取り出して、それを次々に発声していけばよい。その様子を図 2.4 に示す。

しかしこの手法は基本的な音声合成でしかないため、実際の人間の発音とはまだまだ

表 2.2: 代表的な音声データの符号化方式 その 1

1. **basic PCM**(pulse code modulation)

　音声の振幅データをそのままバイナリ化したデータである。音声の波形に対して線型なデータであるため、ワークステーション上での加工を非常に簡単に行うことができるという特徴がある。

2. **Nonlinear PCM**

　音声の振幅データを線型でない対応付けを用いてバイナリ化したデータである。特によく用いられる方式が、対応付けに対数を用いたものである。この方式では、振幅値が最大に近づくとよりまばらに対応付け、振幅値が 0 付近では密集させて対応付けている。この方法は対数圧縮とも呼ばれる。

　この方式のうち、日本やアメリカではμ-law 方式が、ヨーロッパでは A-law 方式がサンプリング周波数 8kHz で電話に使われている。これらの両方式とも対数圧縮の 1 つである。

　ある瞬間の音声データから元の音声の振幅値に復元できる。そのためデジタル化した音声を取り扱う場合には、大抵の場合この対数圧縮された音声データが使われている。

3. **Adaptive PCM**

　音声の振幅データを可変な値で割ったデータをバイナリ化したものである。この可変な値はそれまでの音声データの振幅から求めるのだが、その求め方によって様々な方法がある。

　ある瞬間の音声データから元の音声の振幅値を求めるには、その音声データの最初からその時点までのデータが必要になる。そのため音声をワークステーション上で保存するには適しているが、編集などには向いていない。これは以降の符号化方式に共通して特徴である。

表 2.3: 代表的な音声データの符号化方式 その 2

1. **Differential PCM**

   音声の振幅データではなくて、以前の音声の振幅データと現在の音声の振幅データとの差分をバイナリ化したものである。この方法はアナログの音声に対しても使えるし、デジタルになったあとの Basic PCM の音声データに対しても使える。

   しかし差分が音声の振幅データをバイナリ化したものより小さいことを期待した方法であるため、差分がその期待した幅よりも大きい場合は、その変化を表すことができない。

2. **Adaptive Differential PCM**

   音声の差分データを、それまでの音声の差分データから求めた可変な値で割ったデータをバイナリ化したものである。

   この方法は、Differential PCM での弱点である、音声の差分データが急激に変化した場合にそれに追従できないという問題を解決するためのものである。

   この Adaptive Differential PCM や Differential PCM といった方式では、圧縮することが目的ではなく、圧縮することによって同じビット数で表すことのできる音質を良くすることが目的であると言える。

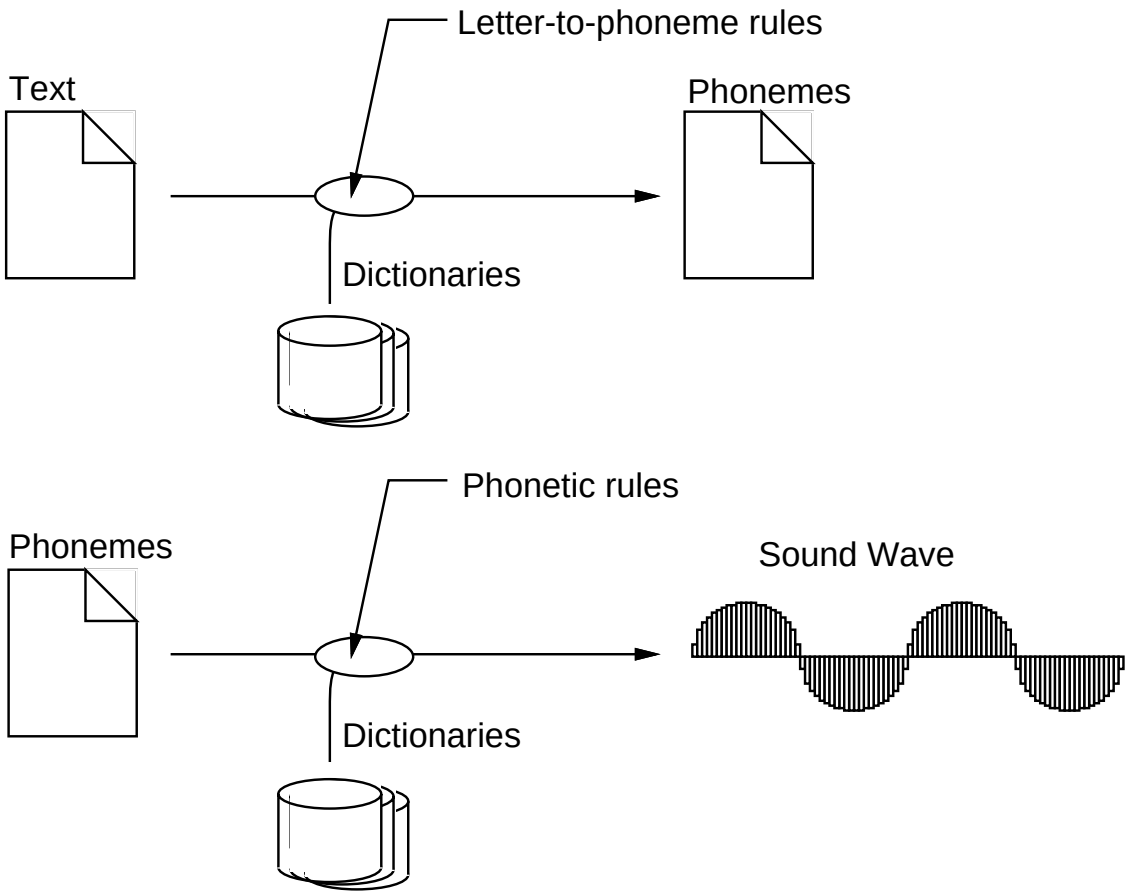
Letter-to-phoneme rules
Text
Phonemes
Dictionaries
Phonetic rules
Phonemes
Sound Wave
Dictionaries

図 2.4: 音声合成の様子

異なる音声になってしまう上に、人間にとってはかなり聞きづらい。人間の発音と較べると、同じ語句を発音する場合に前後に続く語句や音声によって様々に変化した発音をするという点や、語句と語句の間での抑揚を付けるという点がない。

発音にこのような変化を付けるために、実際の音声合成装置では音声に直接変換するのではなく、一度中間コードに変換しておいてそこで語句と語句の接続などの調整をした後に音声化する場合が多い。

また構文解析だけではその語の読みや抑揚を正しく判断できない場合も多々あるため、意味の解釈を行う場合もある。

2.2 音声入出力についての考察

この論文では、先に述べた音声入出力を行うハードウェアとソフトウェアがワークステーションに既に装備されていることを前提としている。この音声入出力を実際に行うハードウェアを音声入出力インタフェースと呼び、そのインタフェースを Unix から使用できるようにしたソフトウェアを音声入出力デバイスと呼ぶ。

この音声入出力デバイスを利用することによって、音声を音声データに変換してワークステーション上に取り込むこと、そして取り込んだ音声データを音声に変換して外界で発声することが可能になる。また取り込んだ音声データを保存することや、加工したその結果を出力することなども可能になる。

しかし音声入出力デバイスを直接使って音声データを取り扱う場合には、以下に挙げる様な問題が発生する。

この論文では、これらの問題のうち音声入出力デバイスに密接した項目 1から、項目 3について考察し、それらの問題を解決するために音声入出力デバイスを管理する機構を構築する。また項目 4についても管理機構がネットワークを介して音声データを送る必要があるために考察する。ただし項目 5については取り扱っていない。

以下ではこれらの問題についての考察とその問題を解決するための管理機構についての考察、そして音声データの取り扱いに関する問題に対してこれまで取られてきた解決方法について述べる。

2.2.1 音声入出力デバイスについての考察

表 2.4に挙げた問題の中には、まず音声入出力デバイスのアクセス方法が統一されていないという問題があった。またこの問題に関連しているのだが音声入出力時にデバイスに対して読み書きする音声データの形式が統一されていないという問題もあった。

従来これらの問題を解決するためには、デバイスを操作するプログラムをデバイス別または音声データ別に用意することによって対応するという解決方法しか取ることができなかった。

またデバイスが管理されていないために、複数のプロセスが同時にデバイスをアクセスすると混乱が生じてしまうという問題もあったが、この問題を解決するにあたっても、

表 2.4: 音声入出力デバイスを直接使った場合の問題点

1. 音声入出力デバイスのアクセス方法が統一されていない。

   音声入出力インタフェース毎に、音声入出力を行う方法が異る。

2. 音声入出力デバイスが管理されていない。

   同時に音声入出力デバイスをアクセスした場合に正常に動作しない。

3. 音声入出力する音声データの形式が統一されていない。

   音声データの変換をユーザが行う必要が生ずる。

4. 音声データをネットワーク上で転送するためのプロトコルが統一されていない。

   音声データをネットワーク上でリアルタイムに送信することが困難である。

5. 音声データをファイルとして保持しておくためのフォーマットが統一されていない。

   音声データの変換をユーザが行う必要が生ずる。

デバイスをアクセスするプロセスを複数起動させないといった消極的な解決方法した取られていなかった。

しかし根本的にこれらの問題を解決するためには、統一的なインタフェースを備えた音声入出力デバイス管理機構を構築し、音声入出力デバイスのアクセスはその管理機構を介して行うことが必要である。

また音声データをネットワーク上で転送するためのプロトコルが統一されていないという問題も存在する。ここで言うプロトコルとは、例えばデータを転送速度が遅いネットワークを利用して音声データを交換するために音声を圧縮してして転送するプロトコルであるとか、リアルタイムに音声データを交換するために時間情報を付加して転送するプロトコルのことである。

このようなプロトコルを設計し、ネットワーク間で音声データを転送するための機構を構築することによって、ネットワーク上で音声データを転送するためのプロトコルを統一することが可能である。

しかしこの論文ではそこまでの機構を実現していない。本論文では音声入出力デバイスの管理機構とのプロトコルを決めているため、そのプロトコルをそのままネットワーク上でも使用している。

2.2.2 音声入出力デバイス管理機構についての考察

序論で述べた様に音声メッセージを出力するなどといった音声入出力を利用した様々な用途を実現するには、この管理機構に音声を入出力する機能が必要である。これらの機能をどう実現するかについて考察する。

音声の入力作業には音声インタフェースが必要である。このインタフェースを使って入力を行うことができるのは 1 人だけであるため、複数の入力処理を同時に行うことはない。したがって、入力に関しては原則としてデバイスの `open` を排他的に行うことで解決できる。そこでまず、音声の出力に関して考察を行うことにする。

出力に関して考えると、音声入出力デバイスは、複数の処理によってデバイスの共有が困難であり、デバイスによって制御方法が異なるなどの点において、プリンタデバイスと類似点が多い。したがって、4.3BSD のラインプリンタスプーラ [92] を参考にすることができる。

4.3BSD のプリンタスプーラは、デバイスを管理するデーモン `lpd` と出力の形式に応じて使い分けるフィルタ、出力待ちデータを一時格納するスプールディレクトリ、スプールディレクトリへデータを格納しプリント要求を出す `lpr` で構成されている (図 2.5参照)。また、種々のプリンタに対応するため設定ファイルとして `/etc/printcap` が用意されている。

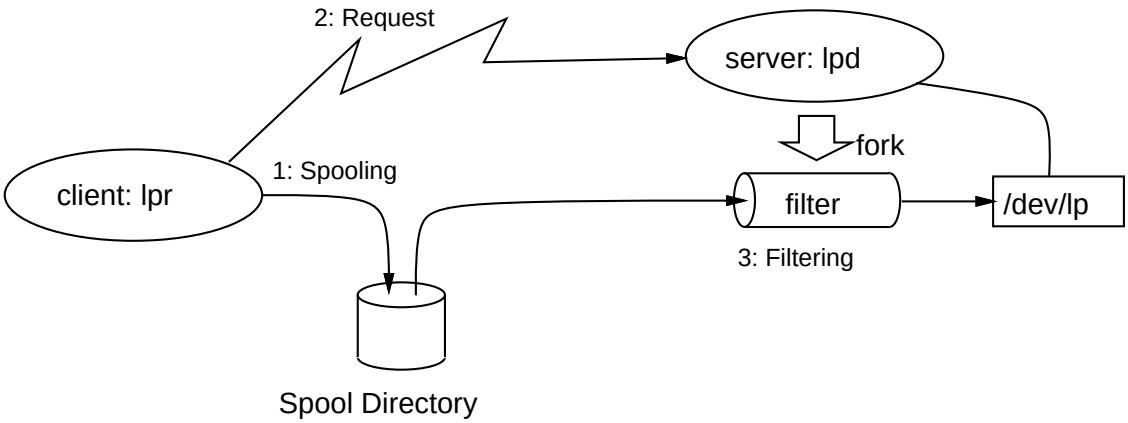

図 2.5: 4.3BSD ラインプリンタスプーラの構成

音声入出力デバイス管理機構でも、プリンタスプーラの構造をそのまま継承することが可能である。しかし、音声出力の場合には通常音声出力を一時停止して緊急音声出力を行い、その後で停止した音声出力を再開するという動作も必要である。つまり必ずしもリクエストの順に出力が行われるわけではなく、優先度などの考慮が必要である。また、リアルタイムにデータを送り出さなければならないためアクセス速度も問題となり、スプール機構としてファイルを用いることについても再検討を必要とする。

また、NEWS の音声入出力デバイスのように、入力と出力が同時にできない半二重型の音声入出力デバイスでは、音声入力を行う時に音声出力を止めなければならない。し

たがって、このようなデバイスを用いる場合は、出力と入力を同時に管理しなければならない。

これらの議論を元に、3.1において音声入出力デバイス管理機構を設計する。

2.3　音声合成についての考察

音声入出力についてと同様に、この論文では先に述べた音声合成を行うハードウェアとソフトウェアが、ネットワークに接続できるコンピュータ上に用意されていることを前提としている。この音声合成を実際に行う装置を音声合成装置と呼ぶ。

この音声合成装置を利用することによって、漢字仮名交じり文といった平文を音声に変換することが可能になる。すると利用方法としては、音声入出力だけでは不可能であったものが中心になる。例えば定形的でない音声によるメッセージを生成し発声するといった利用方法である。

しかし音声合成装置を直接使って音声データを生成する場合には、以下に挙げる様な問題が発生する。

表 2.5: 音声合成デバイスを直接使った場合の問題点

1. 音声合成装置のアクセス方法が統一されていない。

   音声合成装置毎に、音声合成を行う方法が異なる。

2. 音声合成装置が管理されていない。

   同時に音声入出力デバイスをアクセスした場合に正常に動作しない。

3. 音声合成した音声データのフォーマットが統一されていない。

   音声データの変換をユーザが行う必要が生ずる。

4. 音声合成装置を多数用意できない。

   その場で音声合成を行うことができない。

この論文では、これらの問題について考察しそれらの問題を解決するために音声合成装置を管理する機構を構築する。以下ではこれらの問題についての考察とその問題を解決するための管理機構についての考察、そして音声合成装置の取り扱いに関する問題に対してこれまで取られてきた解決方法について述べる。

2.3.1 音声合成装置についての考察

表 2.5に挙げた問題の中には、音声入出力デバイスの時に挙がった問題と同じ問題が多数含まれている。それらのアクセス方法が統一されていない、管理されていない、音声データの形式が統一されていないといった問題に関しては、音声入出力デバイスの時と同じ解決方法が取れる。つまり、統一的なインターフェースを備えた音声合成装置管理機構を構築し、音声合成装置のアクセスはその管理機構を介して行うという方法である。

また音声合成装置が高価であるため多数用意できないという問題があるが、これはその音声合成装置をネットワークに繋がっているコンピュータに接続しておき、そのコンピュータ上でネットワークに対応した管理機構を動作させればある程度解決する問題である。つまり、ネットワーク上で音声合成装置を共有して解決するのである。

2.3.2 音声合成装置管理機構についての考察

序論で述べた様にメッセージを音声化して出力するなどといった音声合成を利用した様々な用途を実現するには、この管理機構には平文を受け取り音声を出力するといった機能が必要である。この機能をどう実現するかについて考察する。

音声合成に関して考えると、音声合成装置は複数の処理を同時に行うことが困難である、音声合成装置によって制御方法が異るなどの点において、音声入出力管理機構の出力側と同じく本論文の 2.2.2で説明しているプリンタデバイスと類似点が多い。したがって同様に、4.3BSD のラインスプーラ [92] を参考にすることができる。

このスプーラについては既に本論文の 2.2.2で説明しているためここでは説明しないが、音声合成装置管理機構でも、プリンタスプーラの構造をそのまま継承することが可能である。

また音声合成においては、文章の途中で音声合成を中断し、他の文章の音声合成を行った後に再開するという動作が不可能である。これは、音声合成の対象が文章であり、文節や文字だけを音声合成すると抑揚や発音がおかしくなるためである。そのため、音声入出力デバイス管理機構で行ったように優先度を設定するということはできない。

つまり、構造的には完全にプリンタスプーラの構造を継承することになる。

こうした平文を音声として合成し音声データを提供する機構がネットワーク上に実装されているものとして、“Etherphone” という装置がある [93]。この装置は基本的に音声をネットワーク上で使用するための装置であり、音声ファイルの管理や音声合成というサービスを提供する。

しかしこの装置でいう音声合成とは英文の音声合成のことであり、現在必要なのは日本文の音声合成である。そのためこの装置でいう音声合成の外見、つまりネットワークを介して送られてきた平文を音声合成し音声データをネットワークを介して返すという部分を踏襲して管理機構を構築する。

これらの議論を元に、3.2において音声合成装置管理機構を設計する。

第 3 章

設計

3.1 音声入出力デバイス管理機構

音声入出力デバイス管理機構の構成図を図 3.1に示す。この図中の audiod はデーモンとして動作し音声入出力デバイスファイルの管理を行なう。この機構は複数のクライアントからの音声入力や音声出力といった要求を受けて、優先度などを考慮しながらそれらの要求を順次音声入出力デバイスに割当てる。

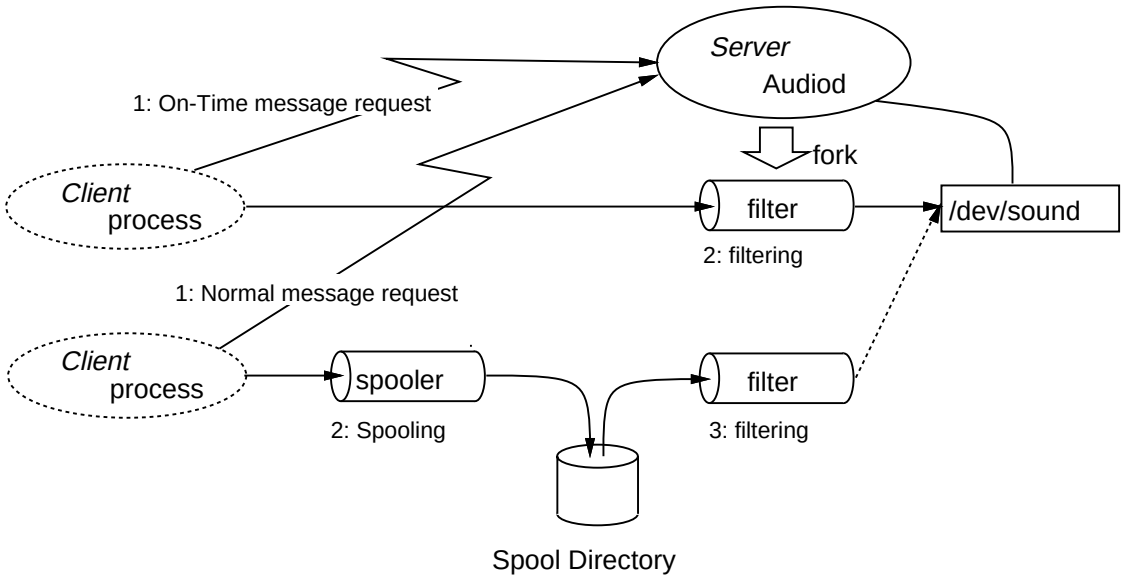

図 3.1: 音声入出力デバイス管理機構の構成図

クライアントからの要求には、その性格に応じて表 3.1に示す 4 つの優先度を設けている。

1から 4の順に優先度は低くなり、優先度の高いメッセージの入出力要求が発せられると、優先度の低いメッセージの入出力は中断され、優先度の高いメッセージにデバイスを解放する。ただし、優先度のもっとも高い Emergency Message は、スーパーユーザプロセスからしか用いることはできない。

また、表 3.1では音声入力が Emergency Message と On-Time Message の間の優先度として取り扱われているが、これは音声入出力デバイスが、前節で問題となった半二重型

表 3.1: 音声入出力デバイス管理機構の受け付ける要求

1. **Emergency Message**

   システム異常の報告など緊急時のメッセージ用の音声を出力するための要求。

2. **Input Message**

   音声を入力するための要求。

3. **On-Time Message**

   音声 phone などで用いるための、即時性の高いメッセージ用の音声を出力するための要求。

4. **Normal Message**

   一般のメッセージ用の音声を出力するための要求。

である場合の取り扱いである。つまり半二重型の音声入出力デバイスを用いて音声を入力する場合は、On-Time Message を出力中でも中断して入力作業を行なうようになっている。

これは音声出力では既にデータが用意されているのに対して、音声入力ではこれからユーザが話す内容をリアルタイムで入力しなくてはならないわけであり、特にリアルタイムなアプリケーションである音声 phone などを実装する場合、ユーザに再度の入力を促すことは可能な限り避ける必要があると考えたからである。

実際にデバイスとの入出力動作を行なうのはフィルタプロセスであるが、これは音声データの内部表現と実際にデバイスに対して読み書きするデータフォーマットとの変換を行なう。

また他にも入出力を行おうとしているフィルタが存在するにもかかわらず、1 つのフィルタが音声入出力デバイスを占有して入出力を続けることを防ぐために、1 つのフィルタが連続して入出力できる時間は制限されている。

音声メッセージを用いる各アプリケーションから発せられる音声入出力の要求としては既に簡単に説明した 4 種類があるが、以下に詳しく説明する。

3.1.1 Normal Message

Normal Message は出力されればそれでよいというメッセージであり、その出力にかかった時間は問題にならない。そのため一度ディスク上にスプールした後に出力している。

このメッセージならば、どれほど遅いプロセスや遠いホストから送ったとしても音声が

再生の途中で途切れることはない。

また再生している最中に、他に優先度の高いメッセージが送られてきた場合はそこで再生は停止する。これを再開する時には、音声の最初から再生するか、それとも停止した所から再生するかを選択できる。ただしその選択は音声を送る前に行う必要がある。

3.1.2 On-Time Message

On-Time Message はリアルタイムで出力するためのメッセージであり、Normal Message とは逆にその出力にかかる時間が重要な問題になる。そのため、ディスクにスプールするようなことはせずに、直接メモリ上でバッファリングしながら出力している。

このメッセージを遅いプロセスや遠いホストから送った時には、音声が再生の途中で途切れてしまう可能性がある。これを解決するには、メモリ上に数秒程度のバッファ [1] を用意して音量をチェックしながらスプールし、有音の間だけ再生するようにする。しかし今回の実装では以下に挙げた理由で、スプールは行わないことにした。

- フィルタは複数同時に起動していることも多いため、メモリの消費量は少ない方が良い。
- 音声入出力デバイス自体がバッファリングされていることもあり、普通の負荷であれば音声出力が途切れることはない [2]。
- 最も負荷が集中する状態は、音声入力をそのまま On-Time Message で再生している状態である。この負荷を少しても軽減するために、後で述べるように音声入力の時にはスプールしながら有音部分だけを録音し転送するようになっている。そのため On-Time Message を処理するフィルタが受け取る音声データにはほとんど無音部分が含まれていない。

また再生している最中に、優先度の高い他のメッセージが送られてきた場合はそこで再生が停止する。これを再開すると、停止した所から再生が続けられる。

3.1.3 Emergency Message

Emergency Message も On-Time Message と同様にリアルタイムで即時に出力するためのメッセージである。そして処理の内容も同様である。違いは優先度と、そのメッセージを送ってきたプロセスが root であるかどうかのチェックを行っている点である。

---

[1] 8KHz で 8bit サンプリングした音声データなら、秒数× 8Kbyte のサイズになる。

[2] 実際負荷がロードアベレージで 6 程度のホストで音声再生をバッファリングするフィルタを用いて再生してみても音声が途切れることはなかった。

3.1.4 音声入力

この音声入力もリアルタイムで入力するためのメッセージであるため、メモリをバッファとして使っている。ただ有音の部分だけを録音する必要があるため、バッファ中で一度スプールして音量のチェックを行っている。

またデバイスから入力する時に指定するサイズが大きすぎるとブロックされてしまうため、一度に入力するサイズを適切な値に調整する必要がある。ここで言う適切な値とはブロックされても大丈夫な値、つまり有音であると判断するのに必要なサイズ程度の値である。

3.2 音声合成装置管理機構

音声合成装置管理機構の構成図を図 3.2に示す。この図中の `voiced` はデーモンとして動作し音声合成装置の管理を行う。この機構は複数のクライアントからの音声合成という要求を受けて、それらの要求を順次音声合成装置に割当てる。

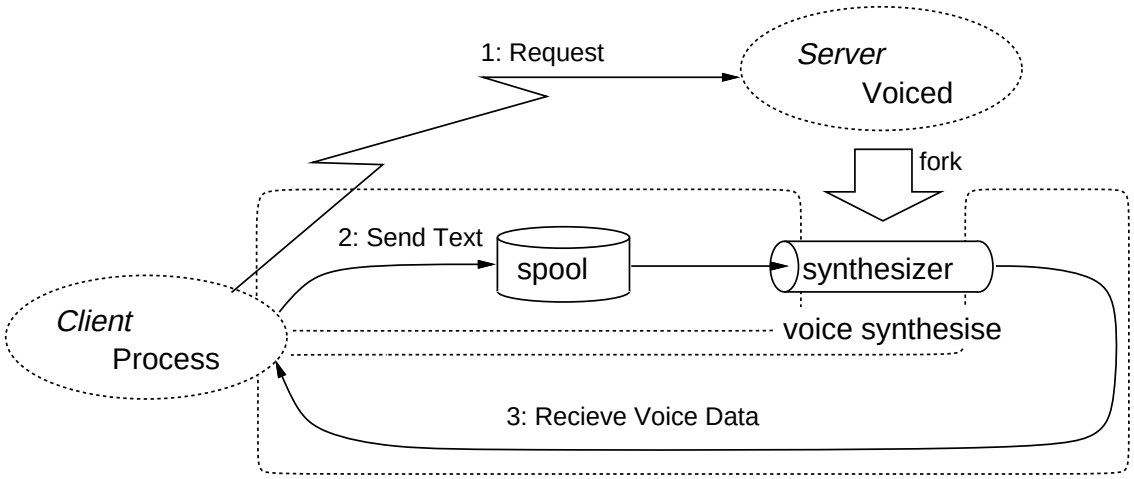

図 3.2: 音声合成装置管理機構の構成図

図 3.2に示されるように、音声合成ではクライアントが音声合成するための平文と音声合成した音声データとを取り扱わなくてはならない。この出力と入力を取り扱う方法としては、両者を同時に行う方法と、出力を終えてからその結果を入力するという方法を考えることができる。

前者の方法を取った場合には、同時に行うクライアントを記述する必要があるのだが、それには読み書きの最中にブロックしないようにするなどの複雑な処理を行うことになるため、そのようなクライアントを記述することは大変である。それに対して後者の方法を取った場合には、タイムラグが生じるが記述は簡単である。

しかし音声合成を行う場合には、送った平文を音声合成した結果である音声を得ることができればよいわけで、送った文章を次々に音声にする必要はないため、タイムラグが生じても問題にはならない。

そのため、本論文の実装では後者の簡単な方法を取り、音声合成するための平文を送るフェーズとその結果である音声データを受け取るフェーズを分けることにした。しかしこれらのフェーズを分けるためには、送るべき平文の大きさをあらかじめ音声合成管理機構に伝えておく必要がある。そこで音声合成の要求時には、合成する平文の大きさも伝えることにする。

また実際に音声合成装置との入出力動作を行うのはシンセサイザプロセスであるが、図 3.2に見られるようにこのシンセサイザを管理機構から分離することによって、他の種類の音声合成装置であってもこのシンセサイザを記述しなおすことによって対応可能な設計にした。

第 4 章

実装

4.1 音声入出力デバイス管理機構の実装

audiod という音声入出力デバイスを管理するデーモンと、そのデーモンが起動する数種類のフィルタおよび音声入出力を行うアプリケーション用のライブラリを実装した。

デーモンはクライアントからの音声入出力メッセージに従って、フィルタを起動する。その後はクライアントはフィルタと接続されて音声データの交信を行う。またデーモンはフィルタ間の音声入出力デバイスの排他制御も行っている。図 4.1にその動作している様子を示す。

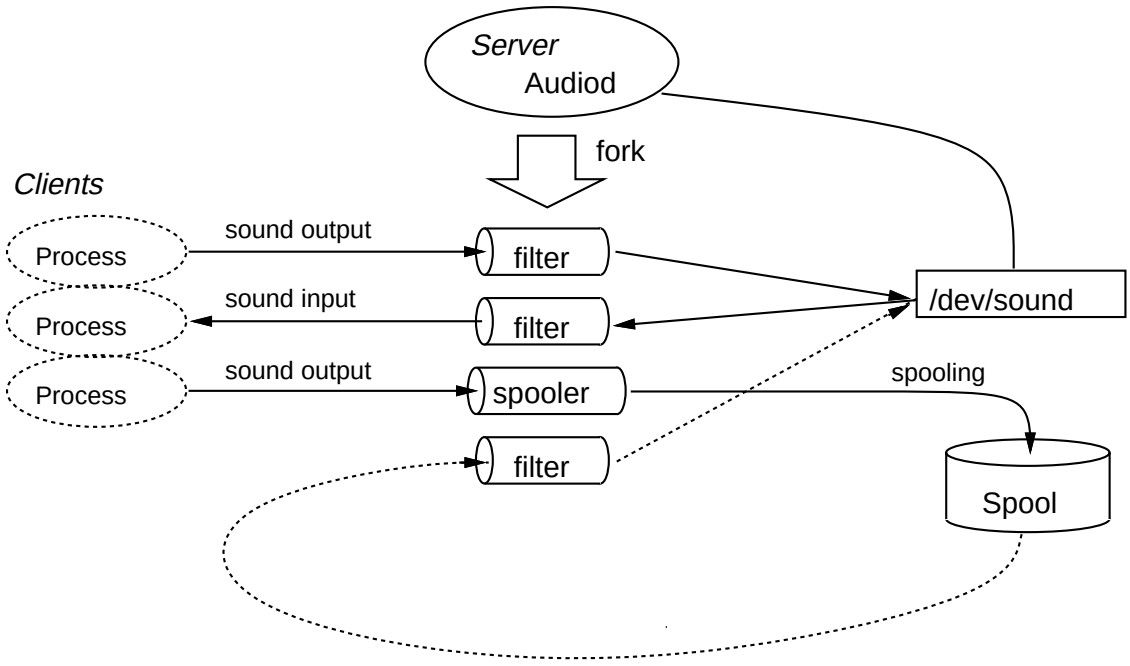

図 4.1: 音声入出力デバイス管理機構の動作状態

4.1.1 デーモンの実装

audiod は、最初に音声入出力デバイスをオープンする。このデバイスはデーモンから

起動されるフィルタへと引き継がれ、クローズされることはない。ただしデーモンとすべてのフィルタがデバイスのファイルディスクリプタを持っているため、フィルタ間でデバイスの排他制御を行う必要がある。この制御はデーモンが決定して、フィルタにデバイスの使用許可を与えることで実現している。

このデーモンが処理しなくてはならないことは 3 種類ある。1 つは音声メッセージを受け取ってその要求に答えること。次にフィルタがデバイスを開放した時に適当なフィルタにデバイスの使用許可を与えるということ。最後にスプールされた音声データを探してそれを再生することである。これら 3 種類の処理をどのように実装したか述べる。

- 音声メッセージを受け取り、その要求に答える。

  デーモンに音声メッセージが届いた場合、そのメッセージの優先度と、現在デバイスを確保しているフィルタの優先度を比較する。もし新しいメッセージの優先度の方が高ければ、フィルタにメッセージを送ってデバイスを開放させた後に、新しいメッセージ用のフィルタにデバイスを使用させる。ただし、全二重型の音声入出力デバイスならば、音声の入力と出力を別々に開放／再開する必要がある。またデバイスを開放したフィルタは停止状態になっているが、このフィルタに再度デバイスの使用許可を与えることによって入出力が再開される。

  またその優先度が低ければ、フィルタを起動するだけである。そうすると、後にデバイスが開放された時にそのフィルタにデバイスの使用許可が与えられる。またもしそのメッセージが Normal Message ならば、デバイスを使用する必要がないため、即座に音声データをスプールするプログラムを起動してスプールを始める。

  ここでデバイスを開放するタイミングをフィルタにまかせているのは、後のフィルタの実装の所で述べるように、音声が突然途切れないようにするためである。また入出力を再開する時に、その再開する場所もフィルタにまかせている。そのため Normal Message 用のフィルタでは、再開した時に音声の最初から再生するか、停止した所から再生するかを選択できる。

- 適当なフィルタにデバイスの使用許可を与える。

  デバイスが開放された時には、デーモンは次のフィルタにデバイスを割当てる。これは、基本的には優先度の順にデバイスの使用許可が与えられるのだが、長時間に渡って動作してきたフィルタは徐々に優先度が下がるように設計した。

- スプールされた音声データを再生する。

  もし上記の動作をする必要がなければスプールされた Normal Massage の音声データがあるかどうか調べて、あればその音声データを徐々に再生する。

4.1.2 フィルタの実装

フィルタはデバイスに対する入出力動作を行なうため、他のフィルタと同時にデバイスをアクセスしたりしないように、デーモンの決定に従って動作しなくてはならない。

つまりフィルタは、デバイスの使用許可が降りている間だけデバイスに対して音声データを読み書きできる。先に述べたようにデバイスを連続して使用できる時間は制限されているため、その後はデバイスを開放して、再度デバイスの使用許可を待たなくてはならない。この時またデバイスを開放したことを確実にデーモンに伝えるために、メッセージを送る必要がある。

ただし音声を途切るときには、突然音声が途切れることを防ぐために、音量のチェックをして可能ならばある程度無音部分が続いた後にデバイスを開放するということを行う。

4.1.3 ライブラリの実装

音声入出力を行うアプリケーションがこの管理機構を利用して音声を入出力するには、まず audiod へソケットを用いて接続しなければならない。次に表 3.1に示した音声入出力の要求を送ることによって音声データを入出力するための接続が確保される。その後は、確保された接続に対して音声データを読み書きすればよい。

今回実装したライブラリでは、以下の関数を用意した。

- audiod へのソケットをオープンし要求を送る関数。
- 音声データの入出力を行う関数。
- ソケットをクローズする関数。

4.1.4 音声入出力を行うアプリケーション

この音声入出力デバイス管理機構の機能と、実装したライブラリを利用したアプリケーションとして、リアルタイム型音声メッセージ交換システム Voice Phone を構築した。このアプリケーションでは、音声入力と On-Time Message という 2 種類のメッセージを用いている。

また簡単なツールとして、音声入力を標準出力に出力するツールと、標準入力を音声出力するツールの実装も行った。

このライブラリを使用したため音声の入出力については大変簡単に済んだ。

4.2 音声合成装置管理機構の実装

voiced という音声合成装置を管理するデーモンと、そのデーモンが起動する数種類のフィルタおよび音声合成を行うアプリケーション用のライブラリを実装した。

今回の実装において使用した音声合成装置は PC-9801 の拡張ボードの形態のもので、ライン出力しか備えられていない [94]。そこで、制御は PC9801 の Ethernet ボード経由で行ない、出力は一旦 NEWS の音声入出力デバイスからデータを取り込むようになっている。図 4.2にその構成を示す。

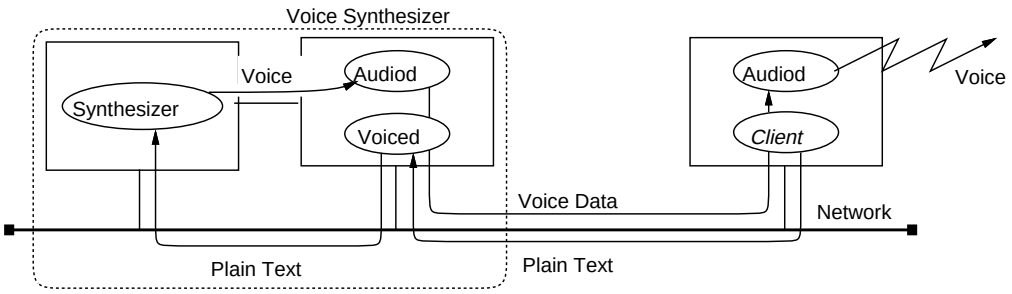

図 4.2: 音声合成装置管理機構の動作状態

この図に示す通り管理機構に音声合成という要求を送り、漢字仮名交じり文を与えると、その音声データを送り返すようになっている。その音声データは既に実装した音声入出力デバイス管理機構を用いて音声入力したものであるため、この際送り返してくる音声データのフォーマットなどについては、4.1において説明した。

4.2.1　デーモンの実装

`voiced` は、最初に音声合成装置に対してコネクトする。この装置へのコネクションはデーモンからフィルタへと引き継がれ、クローズされることはない。こうして音声合成装置の排他処理を実現する。

このデーモンが行わなくてはならないことは、クライアントの出す要求を受けそれに応じてフィルタを起動することと、フィルタが音声合成を終えたら次に処理すべき要求に答えるフィルタを起動することである。

4.2.2　フィルタの実装

フィルタは音声合成装置に平文を音声合成させ、その結果である音声データを受け取るという動作をする。ただし複数のフィルタが同時に動作することはないため、音声入出力デバイス管理機構として実装したフィルタのように、装置の排他処理を行う必要はない。実際に行う処理は、一度スプールしておいた平文を読み取りながらその内容をクライアントに送り返すというものである。

今回の実装では音声合成装置が管理機構とは別の機械上に存在するため、実際のフィルタの実装では、この管理機構で動作する部分と、音声合成装置のある機械上で動作する音声合成装置を制御する部分とに分けて実装した。

前者は、スプールしておいた平文を読み込み後者に転送しながら音声入力を行いクライアントに入力した音声データを転送するという動作を行う。また後者は、送られて来る平文を次々音声合成していくという動作を行う。

4.2.3 ライブラリの実装

音声合成を行うアプリケーションがこの管理機構を利用して音声合成を行うにはまず voiced へ接続して音声合成の要求と合成するデータの大きさを送らなくてはならない。次に先に伝えておいた大きさ分のデータを送り、送り終えた後はフィルタからの音声データを読み込み続ければよい。

今回実装したライブラリでは、以下の関数を用意した。

- voiced へのソケットをオープンする関数。
- 音声合成するデータをスプールする関数。
- スプールしておいたデータを用いて音声合成を開始する関数。
- voiced へのソケットをオープンし、指定したファイルの内容を用いて音声合成を開始する関数。
- 音声合成した音声データを読み込む関数。
- ソケットをクローズする関数。

4.2.4 音声合成するアプリケーションの実装

この音声合成装置管理機構の機能と、実装したライブラリを利用したアプリケーションとして、標準入力からのデータを音声合成するツールと、メイルを音声合成するツールを実装した。

こうしたツールを用いて得た音声データを、4.1.4で実装した音声の入出力を行うツールに送ることによって、手元の音声入出力デバイスを備えたワークステーションで、ネットワーク上の音声合成装置で合成した結果の音声を出力することが可能になった。

第 5 章

評価

音声入出力デバイス管理機構および音声合成装置管理機構を評価しようとした場合、その基準として利用できる対象がないため、ここではあまり定量的な評価を行うことができない。そのため、主に定性的な評価と、機構を実現するに当たって採用した手法から考えられる遅れなどについて簡単な定量的な評価を行う。

5.1 音声入出力デバイス管理機構の評価

本論文で構築した入出力デバイス管理機構において、半二重方式の音声入出力デバイスに対する考察が甘いものであったため、特に音声入力において音が途切れることもあった。

また音声データの圧縮は、LOG-PCM を利用しているだけであり、このままではネットワークに負荷をかけすぎるため、音声をリアルタイムで転送しつづけた場合少しづつ遅れていってしまう。しかし実際は音声の有音部分と無音部分を識別して有音部分だけを転送しているため、問題にはならない。

またこの機構を使った音声入出力とデバイスに対して直接行った音声入出力とを比較すると、管理機構の取り扱うメッセージの中では Normal Message 以外の音声入出力はリアルタイム性を重視して設計しているため、それほど遅れることはない。実際の遅れは、バッファリングで引き起こされるオーバヘッドとソケット間でデータを転送する時に発生する。

しかしネットワークを経由した先のワークステーション上で音声入力をしてその音声を手元のワークステーションで音声出力する場合には、ネットワークの状態によって遅れが変化する。現在の音声データの形式では、64Kbps の容量を必要とするため、イーサネットなどで接続されている場合には数秒程度の遅れで済むが、ISDN などで接続されている場合にはその容量を完全に使い切ってしまうため、遅れは蓄積していってしまう。

このように入力に問題があるものの、序論で述べた音声を入出力することによってインタフェースに役立てるクライアントを装置に依存せずに記述できるような環境は完成した。

この環境を利用して音声による交信を行うシステム、また音声の入力や出力を行うクライアントなども実装した。

5.2 音声合成装置管理機構の評価

本論文において構築した音声合成装置管理機構の評価としては、遅れなどを評価する意味がないのでどのような機能を実現したかについて述べることによって評価に代える。

最初の目的であったネットワーク上のどのワークステーションからでも音声合成を行うことは今回構築した音声合成装置管理機構において達成された。しかし本来音声合成装置は、プリンタと同様にネットワーク上のどのワークステーションからでも共用し利用できるべき装置である。またプリンタとは違い、ネットワーク上に合成できる音質が同じ複数の音声合成装置が用意してあった場合には、そのどれで音声合成を行おうとも問題はないはずである。

したがって、音声合成装置を備えていないワークステーション上でも音声合成装置管理機構は動作していて、もし音声合成の要求を受けた場合は他の音声合成装置を備えているワークステーションにその音声合成を行わせるといった機能も備えているべきである。

このように完全ではないものの、序論で述べた音声を合成することによってインターフェースに役立てるクライアントを装置に依存せずに記述できるような環境は完成した。

この環境を利用して音声合成を行うクライアントを実装し、前節で述べた音声を出力するクライアントと併用することによって、平文を音声合成して出力するといったことができるようになった。

第 6 章

結論および今後の課題

本論文において構築した音声入出力デバイス管理機構および音声合成装置管理機構は、音声入出力デバイスや音声合成装置を持ったワークステーション上でしか動作しない。

音声入出力デバイス管理機構ならば、そういったデバイスを備えたワークステーション上で動作させなくては意味が無いため問題はない。

しかし本来音声合成装置は、5.2でも述べているようにネットワークのどこからでも同様なサービスを受けることができるべきであり、特定のサービスをワークステーションを陽に示す必要はないはずのものである。このようなネットワークにきちんと対応した音声合成装置管理機構を構築することが今後の課題と言えよう。

また本論文で構築した機構によって、外界とワークステーションの間で音声を交換すること、そしてワークステーションから文章やエラーメッセージなどを発声することが可能になった。そうすると次に考えられるのが、音声を認識し平文に変換することである。この機能を実現することによって、音声による命令をワークステーションが認識することなどができる。

音声認識装置を入手できたので、この装置をシステム中に取り込むことを考えている。

今後考えなければならないことは、音声メッセージを何に利用するかということである。すでに幾つかの利用形態を実現したが、もっと様々な利用方法が考えられるであろう。またマルチメディア型のメッセージ交換システムだけでなく、より有効な応用について考えていきたい。