Internal Use Only (非公開)

TR-I-0171

ワーク･ベンチ ツールキット･ウィジェット
Work Bench Toolkit Widget

北垣内 浩二
Koji Kitagaito

1990.9

概要

数あるウィンドウ･システムの中で今や事実上の業界標準のシステムともいえるX Window System Version 11であるが、その内部構造は一般的に良く知られていない。本稿はこのX Window System全体の構造を解説すると共に、今回実際に作成したワーク･ベンチ ツールキット･ウィジェットを紹介し、実際の使用方法を検討する。またそれと同時に、その中で使用している関数群が、どのような働きをしているのかなどの記述も行っているので、本稿の熟読によって、XWindow System Version 11 上でのプログラミングが容易に行えるようになって頂けることを期待する。

ATR Interpreting Telephony Research Laboratories
ATR 自動翻訳電話研究所

Contents

Chapter 1

Work Bench
Toolkit Widget
and
X Window System
Version 11

1.1 ワーク･ベンチ ツールキット･ウィジェット概要

本ソフトウェアは、X Window System Version 10 (以下 X10 と略す) にて稼働していた音声処理ワーク･ベンチ Ver3.0 をX Window System Version 11 (以下 X11 と略す)に対応するように書換えを行った際、今までのグラフィック･サブルーチンを他のX11のアプリケーションにおいても、効果的且つ容易に使用できるようにとウィジェットとして作成したものである。

このバージョンは、音声ファイル、ラベルファイルから、

- ○ 波形
- ○ ラベル
- ○ 時間軸スケール
- ○ ログ･パワー
- ○ ランニング･スペクトラム
- ○ FFT スペクトラム･スライス
- ○ LPC スペクトラム･スライス
- ○ スペクトログラム

を表示できるウィジェットから構成されている。これらは特別にWork Bench Toolkit Widget (以下 WBTWidget と略す)と命名した。

WBTWidgetはX11に標準で用意されているToolkit widgetと同じ書式をもち、またそれらと併用し同じ様に使用できる。

波形ウィジェット上にて行う波形の編集は、白黒の反転でより明確に操作出来るよう改善されている。波形、ラベル、時間軸スケール、ログ･パワー、ランニング･スペクトラム等は、それを実際に管理しているウィンドウのサイズの変更で同じ様に拡大/縮小出来るように作成されている。

今回これらのグラフィック表示ルーチンをWidgetとして作成した目的は、

- ○ 他のアプリケーションから使用する時の結合の容易性

○　目的に合ったパラメータ設定によるグラフィック表示の多様性

○　他のサブ･ルーチンから独立したイベント処理の実現

○　保守の容易性

○　他のウィンドウ･システムとの互換性

の向上である。よって今回のワーク･ベンチは勿論、他の如何なるアプリケーションとも結合が可能となっている。具体例を上げれば、音声ファイルとラベル･ファイルから、波形とラベルのみを表示するアプリケーション、相関計算後のデータ･ファイルを基に、ランニング･スペクトラムのみを表示するアプリケーションなどを作成することも簡単に出来る。

またこのWidgetというものを使用するの最大のメリットは、X11のXlibという最も低レベルの関数群、またX Toolkit Widgetという標準関数群のみを使用して作成されているので、DEC社のDECWindows®は勿論、OSF/Motif™とも容易に結合可能となっていることである。

1.2　X Window System について

ここまでX Window Systemの用語が使われたので、ここでそのX Window Systemについて少し解説を行いたい。

1.2.1　開発の歴史

X Window Systemはスタンフォード大学で開発されたWシステムを、マサチューセッツ工科大学のコンピュータ研究所と、IBM、DECによって資金援助された教育関係の共同プログラムであるAthenaプロジェクトが採用し、開発を推進してきた事実上の業界標準ウィンドウ･システムである。このシステムの特徴として、アプリケーションとそれを使用するワーク･ステーションが独立しており、その実現に独自の高性能ネットワーク･プロトコルを使用しているので、ベンダー、或る

いはオペレーティング・システムからの独立性を実現している。それまで各ベンダー毎に存在していた固有のウィンドウ・システムでの問題点としては、そのシステムにおいて開発されてきたアプリケーションは、他のベンダーのシステムでは互換性がなく、ソース・コードの大幅な書き換えを余儀なくされていた。しかしこのX Window System (特にX11) によってこの問題点はほとんど解消されたと言えるであろう。つまりソース・レベルでの移植が可能となったのである。

ここで特にX11と断ったのは、X10ではハードウェア依存性が結構高く、特にキーボードに関してはDECのLK201を前提としており、また標準でサポートされるハードウェアがX10R3ではDEC QDSS、DEC QVSS、DEC VS100、Sunに限定されていた。またソフトウェア的にもサポートされるOSが4.3BSDのみで、System V系ではかなり手を加えなければいけなかった。このような問題点はX11では解消されている。

1.2.2　構造

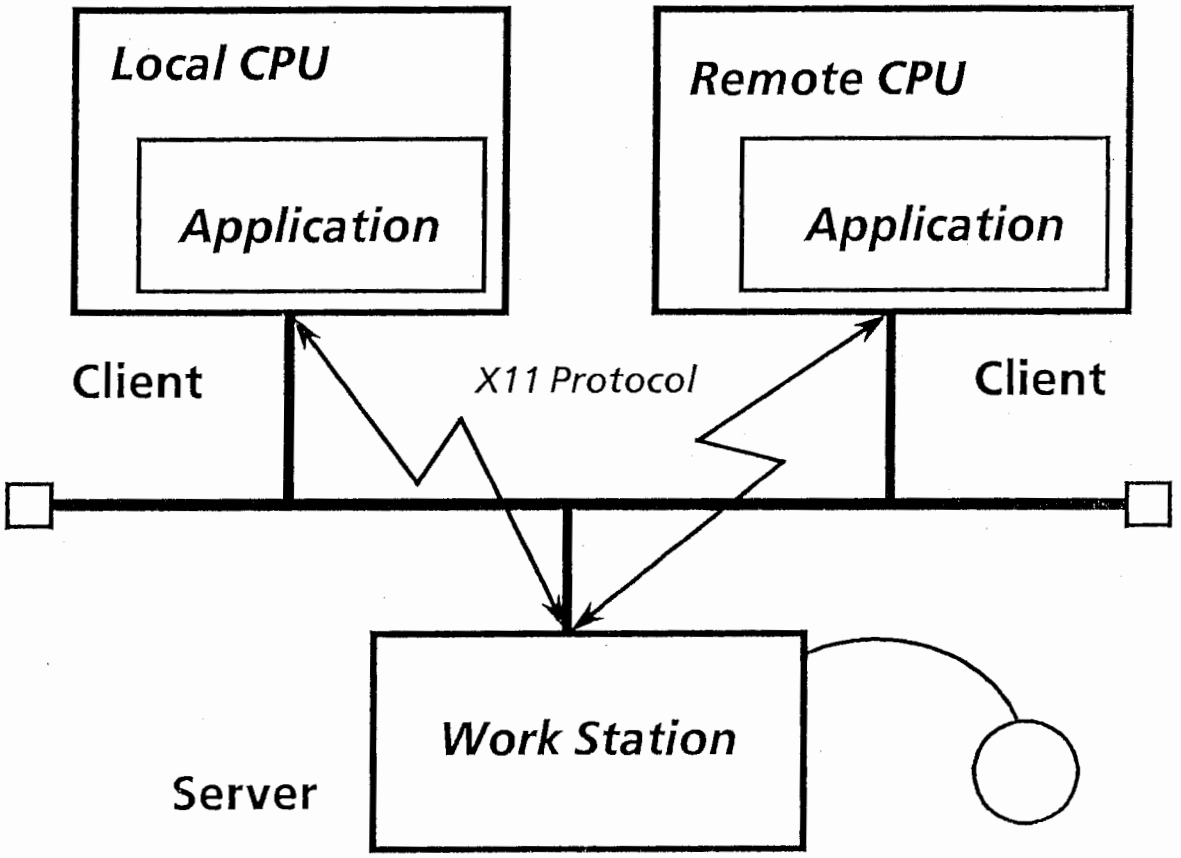

Figure 2-2-1　**X11 Server and Client Model**

X11は、Xサーバ、Xlib、X Toolkit、という3つの基本要素から成り立っている。このXlib、X Toolkit は標準のX11 であり、Xlib, XToolkit, XtWidget Set を使用したアプリケーションをXクライアントという。XサーバとXクライアントの結合はローカル･エリア又はワイド･エリアのネットワークのどちらでも良く、研究所内は勿論、インターネットに加入さえしていれば、例えば日本のワーク･ステーションから外国など遠隔地のシステムにリモート･ログインし、そこで実行し

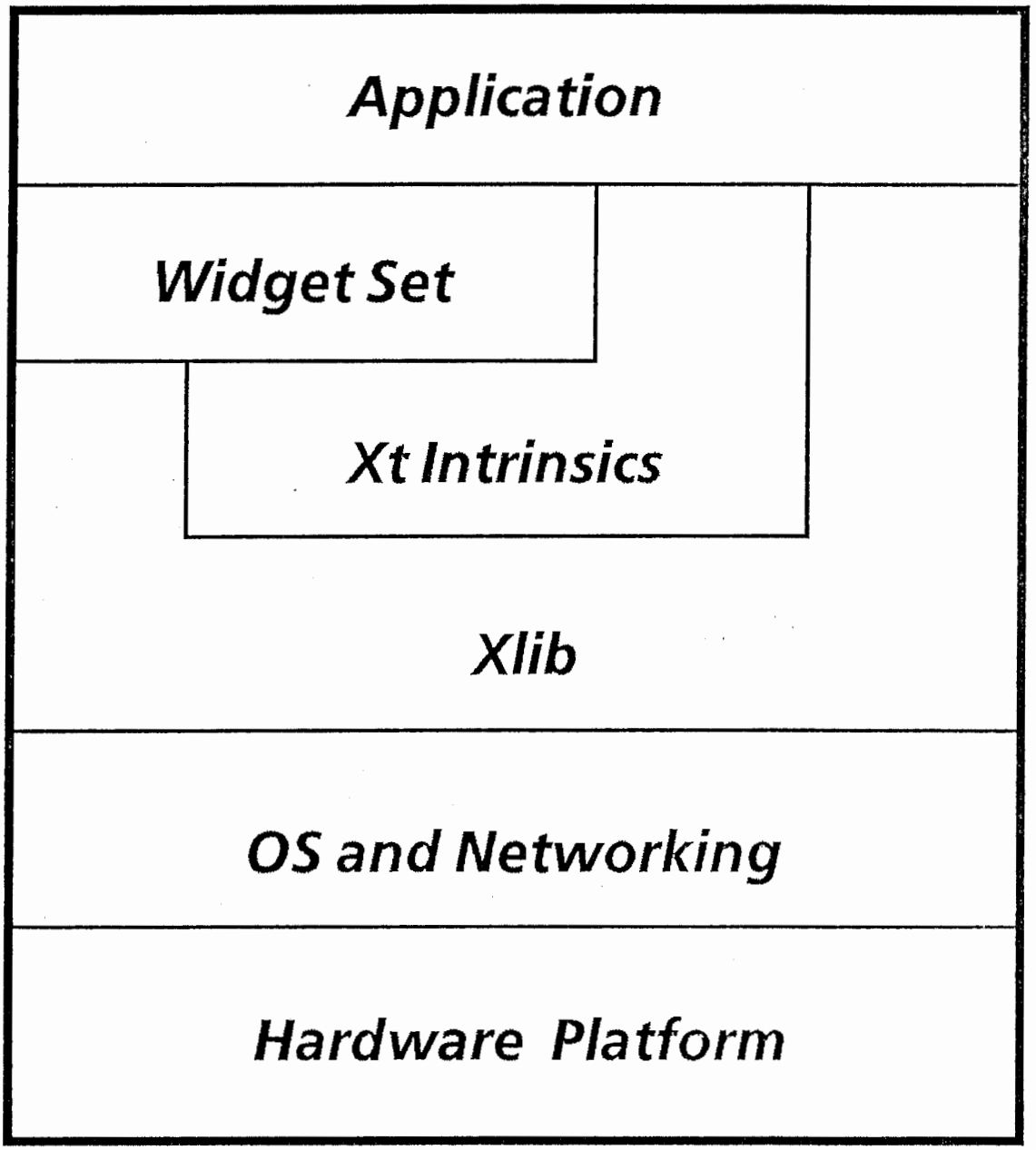

Figure 2-2-2 X11 Structure

たアプリケーションを日本のあるワーク･ステーション上に表示するということも可能である。

X11のネット･ワークの透過性を応用すれば、この様な遠隔地のシステム上でのアプリケーションの実行を始め、以下の様な使い方も可能である。

○　システムの資源が少なく環境の悪いパーソナル･コンピュータ上において、豊かな資源、優れた環境のスーパー･コンピュータのアプリケーションを有効に活用する。

○　UNIX 環境のみでなく、VAX/VMS 環境との接続をも可能としており、異機種の異なったOS との対話を実現している。これは、TCP/IP とDECnet という複数のプロトコルをサポートすることによって実現しているのである。

1.2.2.1 Xサーバ

Xサーバは、ディスプレイ、キーボードとマウスから構成されるハード･ウェアのことであり、アプリケーションのグラフィック機能の実現、各アプリケーションのウィンドウ操作機能、アプリケーションに対するイベント処理等の入力機能を提供している。

ここでいうアプリケーションとは勿論Xクライアントのことである。

1.2.2.2 Xlib

Xlibはベース･ウィンドウシステムであり、ネットワーク･プロトコルに対するインターフェイスをとるC言語ライブラリのパッケージである。これはプロシージャ･インターフェイスに渡されたパラメータを自動的にネットワーク･プロトコルに変換し、サーバから送られてきたアプリケーションの復帰値に変換する。これはOSに最も近い部分に存在するのであるが、アプリケーション･ユーザはネットワークのプロトコル及びハードウェア･コールを考えずに開発を行うことが可能である。

また、よりユーザ･インターフェイスを向上させるためのものが、X Toolkit関数として用意されている。これは、それぞれのインターフェイスであるWidgetとそれらを管理するIntrinsic関数から構成されている。

1.2.2.3 XToolkit

X Toolkitはユーザ･インターフェイスを実現するために必要なサブ･ルーチンの追加セットであり、Intrinsic(イントリンシック)とWidget(ウィジェット)から構成されている。これらを一言でいえば、ウィジェットはユーザ･インターフェイスを構築するためのルーチンであり、アプリケーションから容易に呼び出すことができる。イントリンシックは、ウィジェットを簡単に記述するために用意されている関数群で、同時にウィジェットに対するマネージメントも行っている。

1.2.3 X11 プロトコル

X11は独自のX Window System Protocol を採用し、デバイス･インディペンデントな環境を実現している。Xアプリケーションはネットワーク･プロトコルを直接要求し使用するのではなく、Xlib にこの機能を代行させてプログラミングのインターフェイスを使用して動作するので、ユーザはこのプロトコルを意識せずにプログラミングが可能となる。またネットワーク透過性があるので、ローカル及びリモートのCPUからアプリケーションをワーク･ステーション上に実行することができる。Xlib の関数は、X プロトコルと1対1で対応している。

X への要求はほとんど非同期である。よってクライアントは前の要求を待つことなく次の要求を送信できる。X11 への要求のうち、そのステータス返さなければならないものに関しては、サーバがそれを生成してクライアントに返す。またそのステータスによって次の動作を行うものに関しては、その要求は同期を取ろうとするのである。

勿論これらの要求に対するエラーも非同期で返ってくる。よってX11 でアプリケーションを開発するときは、そのエラー処理ループの中で常にステータスの状態をハンドリングするようにしておかなければならない。

1.2.4 ハードウェア依存性

Xクライアントは、如何なるベンダのXサーバに対しても、実行が保証されている。Figure-2-2-1の様に、1つのサーバに対して複数のクライアントからのアクセスは勿論可能なのであるが、この複数のクライアントはサーバと同じベンダの、また同じOS、ハードウェアで開発されたものである必要はない。またこれらのクライアントも同じベンダのCPUである必要は勿論ない。

特筆すべき事は、異なったベンダのクライアント間におけるデータ共有のハンドリングである。インテルの8086マイクロプロセッサ、モトローラの68000マイクロプロセッサなどのデータ・フォーマットは、記憶領域の小さな部分が最初のバイトである、Big Endian フォーマットであるが、DECのVAXコンピュータはその逆の Little Endian フォーマットである。これらの異なったアーキテクチァのクライアント間においてデータ共有するときは、プロトコル中でバイト順序を変える必要があるのだが、この処理をアプリケーションではなくワークステーションに行わせている。よってアプリケーション開発はこの処理を全く気にせずに行えるのである。

1.2.5 ポータビリティ

X11で開発されたクライアントは、どのベンダーのCPU、ハードウェアにおいても、そのソース・コードを再コンパイルするだけで実行が可能となっている。これは、X11におけるアプリケーション開発の大前提である。しかし一般ユーザ、またはこれからX11を使用してアプリケーション開発を行っていこうと考えている人々を混乱させていることがある。

例えばアプリケーション開発者が、今現在DECwindowsを使用しているとする。彼は前述の大前提に従って、このDECwindowsもX11に

は違いないから、DECwindowsで開発したアプリケーションは他のどのウィンドウ・システム、例えばOSF/Motifにおいてもコンパイルができ、実行が可能であると信じている。しかし答えは“ノー“である。Figure 2-2-2を見て頂きたい。DECwindowsもOSF/MotifもX11を含んでいる。これはXlibとXt Intrinsicsという標準のX11である。しかしそれだけではただのX11である。ウィンドウ・システムを開発しているあらゆるベンダーは、各自のウインドウ・システムにオリジナリティを加えている。例えばそれぞれウィンドウ・システムのスタイルを作り出す為に、Widget Setを作成している。この各ウィンドウ・システムのWidget SetはX11で書かれてはいるのだが、標準のX11とは言わない。このWidgetSetが何であるかによって、DECwindowsまたはOSF/Motifであるとの区別ができ、“DECwindows Widget Set“、“Motif Widget Set“という言い方ができるのである。よって“DECwindowsで開発する“というのは、このDECwindows Widget Setを含めた形で開発を行うことを言うのである。だから、Motifの中に在ろうはずがない、DECwindows Widget Setを使用したアプリケーションは、Motifでは動作しないのである。しかしそのアプリケーションがXlib, Xt Intrinsicsのみを使用したものであれば、それぞれのウィンドウ・システムにおいて互換性が保証される。

話がややこしくなるがMotifにて開発したアプリケーションは、“DECwindowsでは動作しないが、DECwindowsがインストールされたワークステーションでは動作可能”である。これはどういう事かというと、DECwindowsがインストールされたワークステーションに、Motifをインストールすればいいのである。MotifはDECwindowsとは違いOSFからソースで提供されているから、1つのワースステーションにこれら複数のウィンドウ・システムの共存は可能である。そして、それぞれのWidget Setのライブラリを同時にリンクすればいいのである。

1.2.6　クライアント間通信

X11にはAtomというものが存在し、各クライアントはこのAtomというものを利用してお互いの通信を可能としている。ではこのAtomとは一体どういうものであるか。

Atomは、概念のレベルではユニークな名前である。Atomは文字列の様に取り扱われ、32ビットを使用した識別子である。X11には既に定義済のAtomが多数存在する。これらは<X11/Xatom.h>にて定義されている。そのうちのいくつかを列挙してみよう。

Window Manager Hints	*Standard Color Maps*
XA_WM_CLASS	XA_RGB_BEST_MAP
XA_WM_COMMAND	XA_RGB_BLUE_MAP
XA_WM_CLIENT_MACHINE	XA_RGB_DEFAULT_MAP
XA_WM_HINTS	XA_RGB_GRAY_MAP
XA_WM_ICON_NAME	XA_RGB_GREEN_MAP
XA_WM_ICON_SIZE	XA_RGB_RED_MAP
XA_WM_NAME	
XA_WM_NORMAL_HINTS	
XA_WM_TRANSIENT_FOR	
XA_WM_ZOOM_HINTS	

Cut Buffers	*Selections*
XA_CUT_BUFFER0	XA_PRIMARY
XA_CUT_BUFFER1	XA_SECONDARY
XA_CUT_BUFFER2	
XA_CUT_BUFFER3	
XA_CUT_BUFFER4	
XA_CUT_BUFFER5	
XA_CUT_BUFFER6	
XA_CUT_BUFFER7	

Table 2-2-6　**Atoms**

Window Manager Hints のアトムに関しては、Window Manager のプロパティ、例えばその名前、アイコンの名前、サイズなどが設定されている。Standard Color Map のアトムは、ルート・ウィンドウのカラーマップに関する値、RGBの最大値などが設定されている。そして Cut Bufferes と Selections によってクライアント間通信が実現されている。Window Manager は異なった8つのバッファを用意しており、このバッファを利用してターミナル・エミュレータのクライアント間におけるテキストのカット・アンド・ペーストを実現している。また Selections も同様の働きをしている。

これら定義済の Atom の他に、アプリケーション開発者自身が特別に Atom を定義することも可能である。

Chapter 2

Attributes of Work Bench Toolkit Widget Functions

2.WBTWidget

2.1 Transform Widget

ユーザの座標系(ワールド座標系)とXウィンドウの座標系の座標変換を行うためのWidget。変換定数(scaleX/Y,deltaX/Y)の管理、座標変換関数の提供、変換定数が変更された場合の書き直しの要求などを行う。 複数の Transform Widget 間で、変換定数を共有させるための仕組みも提供する。

○ Widget の定義

クラス名	:	Transform
クラス･レコード	:	wbtTransformWidgetClass
スーパー･クラス	:	Simple
リソース	:	下記の全ての Name の頭に WbtN が付く

Attribute Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	100	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	100	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	1.0	X座標縮尺 (スケール)
scaleY	float	1.0	Y座標縮尺 (スケール)
deltaX	float	0.0	原点の移動量
deltaY	float	0.0	原点の移動量
scaleXInherit	float	NULL	定数(scaleX)継承する Widget ID
scaleYInherit	Widget	NULL	定数(scaleY)を継承する Widget ID

deltaXInherit	Widget	NULL	定数(deltaX)を継承するWidget ID
deltaYInherit	Widget	NULL	定数(deltaY)を継承するWidget ID
scaleXAcceptResize	Boolean	False	リサイズに伴ってスケールを自動的に
scaleYAcceptResize	Boolean	False	変更するか否か
windowGravity	int	SouthWestGravity	
scaleXUnit	char*	NULL	X座標の単位
scaleYUnit	char*	NULL	Y座標の単位
resizeCallback	XtCallbackList	NULL	ウィンドウ･リサイズ時に実行する関数
exposeCallback	XtCallbackList	NULL	エキスポーズ時に実行する関数

● scaleXInheritには、scaleXを継承(参照)するtransform widgetを指定する。これが、指定されるとscaleXは、そのwidgetからコピーされscaleXAcceptResizeは無視される。そして、ここで指定したwidgetのscaleXが変更されると、自動的にこちらのscaleXもその値と等しくなる。scaleYInherit、deltaXInherit、deltaYInheritについても同様である。

● scaleXAcceptResize、scaleYAcceptResizeは、ウィンドウがリサイズされた時、それに伴ってスケールを変更するか否かの指定。

● windowGravityは、これがFalseの場合にウィンドウの内容の出力位置を指定する。scaleXUnit、scaleYUnitは、ユーザの座標の単位を表す文字列で、スケールを継承する際の目安となる。(スケール継承先のwidgetの単位と等しく無い時は継承されない)。

● resizeCallbackには、座標変換定数が変更された時にウィンドウを書き直すための関数を指定する。これは、Transform widgetによって自動的にコールされる。

○ユーザ座標系とXウィンドウの座標系

ユーザが用いる座標系は、左下が原点(0、0)とする自然な座標系とする。scaleX,scaleY,deltaX,deltaY とXウィンドウの座標系との関係を以下に示す。

Ux, Uy	:	ユーザの座標
Wx, Wy	:	Xウィンドウの座標
Wheight	:	ウィンドウの高さ(ピクセル数)

Wx = Ux * scaleX + deltaX
Wy = Wheight - (Uy * scaleY + deltaY)

○ 座標変換のための関数

座標変換を行うために widget がアプリケーションに対して提供する関数を以下に示す。

●X座標値の変換関数

```
int WbtTransformX( widget, Ux )
Widget              widget;
float               Ux;

#define WbtX( widget,Ux ) WbtTransformX( widget,Ux )
```

●Y座標値の変換関数

```
int WbtTransformY( widget, Uy )
Widget              widget;
float               Uy;

#define WbtY( widget,Uy ) WbtTransformY( widget,Uy )
```

●ポイントの変換関数

```
typedef struct{
        float       x, y;
} WbtPointF;

void WbtTransformPoints( widget, npoints, point_float, point_return
)
Widget              widget;
```

```
int                 npoints;
WbtPointF           *point_float;
XPoint              *point_return;
```

●セグメントの変換関数

```
typedef struct{
        float       x1, y1, x2, y2;
} WbtSegmentF;

void WbtTransformSegments( widget, nsegments, segment_float,
segment_return )
Widget              widget;
int                 nsegments;
WbtSegmentF         *segment_float;
XSegment            *segment_return;
```

●レクタングルの変換関数

```
typedef struct{
        float       x, y;
        float       width, height;
} WbtRectangleF;

void WbtTransformRectangles( widget, nrectangles, rectangle_float,
rectangle_return )
Widget              widget;
int                 nrectangles;
WbtRectangleF       *rectangle_float;
XRectangle          *rectangle_return;
```

●弧の変換関数

```
typedef struct{
        float       x, y;
        float       width, height;
        short       angle1, angle2;
} WbtArcF;

void WbtTransformArcs( widget, narcs, arc_float, arc_return )
Widget              widget;
int                 narcs;
WbtArcF             *arc_float;
XArc                *arc_return;
```

●XウィンドウのX座標値をユーザのX座標値に変換する関数

```
float WbtTransformReverseX( widget, Wx )
Widget              widget;
int                 Wx;
```

●XウィンドウのY座標値をユーザのY座標値に変換する関数

```
float WbtTransformReverseY( widget, Wy )
Widget              widget;
int                 Wy
```

2.2 WaveWidget

波形表示と波形エディットを行うためのwidget。transform widgetをスーパー・クラスにすることによって、ラベル、時間軸スケール等のwidgetと同期し拡大/縮小、スクロールを可能としている。

マウス操作は、XUI Style Guideに沿う。

○ Widgetの定義

クラス名	:	Wave
クラス・レコード	:	wbtWaveWidgetClass
スーパー・クラス	:	Transform
リソース	:	下記の全てのNameの頭にWbtNが付く

Attribute Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	100	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	波形全体の大きさ	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	wave widget内部で使用	
scaleY	float	wave widget内部で使用	
deltaX	float	wave widget内部で使用	
deltaY	float	wave widget内部で使用	
scaleXInherit	Widget	NULL	定数(scaleX)を継承するWidget ID
scaleYInherit	Widget	NULL	定数(scaleY)を継承するWidget ID

deltaXInherit	Widget	NULL	定数(deltaX)を継承するWidget ID
deltaYInherit	Widget	NULL	定数(deltaY)を継承するWidget ID
scaleXAcceptResize	Boolean	False	width (幅)のリサイズではscaleXは変更しない。
scaleYAcceptResize	Boolean	True	height (高さ)のリサイズでscaleYは変更する
windowGravity	int	WestGravity	
scaleXUnit	char*	"msec"	X座標の単位
scaleYUnit	char*	NULL	Y座標の単位
resizeCallback	XtCallbackList	NULL	wave widgetの内部で使用
exposeCallback	XtCallbackList	NULL	wave widgetの内部で使用 (以上はTransformから)
waveData	short*	NULL	音声データ
waveDataSize	int	0	音声データの数
startPosition	int	0	音声データ表示開始位置 waveData配列の添字
frequency	float	12.0	音声データのサンプリング周波数 (kHz)
msecPerPixel	float	10.0	時間軸の表示スケール (1ピクセル当りのミリセカンド)
powerPerPixel	float	0x10000/height	パワー軸の表示スケール (1ピクセル当りのパワー)

- 音声データ (waveData) は、signed shortの配列とする。scaleX/Y, deltaX/Yなどtransform widgetの定数は、frequency, msecPerPixel, powerPerPixelを基に計算される。
- scaleXUnit, scaleYUnitは、scaleXInherit,scaleYInheritで指定したwidgetと照合する。
- scaleXUnit, scaleYUnitが異なっている場合は継承しない。
- deltaXInherit, deltaYInheritについても同様である。

ユニットは、以下の様に定義されている。

```
#define   WbtNnoName        NULL
```

```
#define   WbtNmsec          "msec"
#define   WbtNsec           "sec"
#define   WbtNkhz           "kHz"
#define   WbtNhz            "Hz"
```

○ マウスの操作

以下に示すマウスの操作は、既に wave widget がリアライズされ、波形表示がされているものとして記述する。

●左ボタン･クリック
　　波形データのセレクト
●左ボタン･ドラッグ
　　波形データのセレクト拡張
●左ボタン･リリース
　　波形データのセレクト終了
●シフトキー/左ボタン･クリック
　　セレクト範囲の始点/終点の変更。クイック位置に近い点(始点/終点)が変わる
●シフトキー/左ボタン･ドラッグ
　　セレクト範囲の変更の拡張
●右ボタン･クリック
　　セレクトされているデータをクリックした位置にコピーする
●コントロールキー/右ボタン･クリック
　　セレクトされているデータをクリックした位置に移動する
●シフトキー/中ボタン･クリック
　　セレクトされているデータを無音状態にする
●コントロールキー/中ボタン･クリック
　　セレクトされているデータを削除する

白黒反転の表示がされている間は、セレクトは有効である。同じ位置で、左ボタンをクリック/リリースすればセレクトは無効になる。

2.3 Wave Label Widget

音声波形のラベルを表示するwidget。transform widgetをスーパー･クラスにすることによって、波形、時間軸スケール等のwidgetと同期し拡大/縮小、スクロールを可能としている。

○ Widgetの定義

クラス名	:	WaveLabel
クラス･レコード	:	wbtWaveLabelWidgetClass
スーパー･クラス	:	Transform
リソース	:	下記の全てのNameの頭にWbtNが付く

Attribute Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	フォントの高さと、表示するラベル層による	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	ラベル全体の長さ	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	wave label widget内部で使用	
scaleY	float	wave label widget内部で使用	
deltaX	float	wave label widget内部で使用	
deltaY	float	wave label widget内部で使用	
scaleXInherit	Widget	NULL	定数(scaleX)を継承するWidget ID
scaleYInherit	Widget	NULL	定数(scaleY)を継承するWidget ID
deltaXInherit	Widget	NULL	定数(deltaX)を継承するWidget ID

deltaYInherit	Widget	NULL	定数(deltaY)を継承するWidget ID
scaleXAcceptResize	Boolean	False	width (幅)のリサイズではscaleXは変更しない。
scaleYAcceptResize	Boolean	True	height (高さ)のリサイズでscaleYは変更する
windowGravity	int	SouthWestGravity	
scaleXUnit	char*	"msec"	X座標の単位
resizeCallback	XtCallbackList	NULL	wave label widgetの内部で使用
exposeCallback	XtCallbackList	NULL	wave label widgetの内部で使用
		(以上はTransformから)	
labelData	WbtLabel* NULL	ラベル・データ	
startPosition	int	0	音声データ表示開始位置 waveData配列の添字
frequency	float	12.0	音声データのサンプリング周波数 (kHz)
msecPerPixel	float	10.0	時間軸の表示スケール (1ピクセル当りのミリセカンド)
font	XFontStruct*	fixed	ラベルのフォント名
displayPhone	Boolean	True	Phone層のラベルを表示するか否か
displayEvent	Boolean	True	Event層のラベルを表示するか否か
displayAloph	Boolean	False	Aloph層のラベルを表示するか否か
displayInsep	Boolean	False	Insep層のラベルを表示するか否か
displayVowel	Boolean	False	Vowel層のラベルを表示するか否か

● Phone, Event, Aloph, Insepの各層を表示するには、1層につき低くとも

(フォントの高さ)+(フォントの高さ/2)+5ピクセル

のウィンドウの高さが必要である。Vowel層を表示するには、低くとも

(フォントの高さ)+(フォントの高さ/2)+8ピクセル

のウィンドウの高さが必要である。

● ラベル･データは以下に示す構造体で受渡しする。

```
typedef   struct    __label1 {
          float     stt, end ;
          char      *lb ;
          int       n__lbl ;
} WbtLabel1 ;


typedef   struct    __label2 {
          float     center ;
          char      vowel ;
} WbtLabel2 ;

typedef   struct    __label {
          WbtLabel1          **phone;
          WbtLabel1          **event;
          WbtLabel1          **aloph;
          WbtLabel1          **insep;
          WbtLabel2          **vowel;
          char               *comm;
          int                lblsize[6];
} WbtLabel ;
```

○ラベルに関する関数

･ラベル･データを読み込む関数

```
WbtLabel *WbtGetLABEL( fp )
FILE                *fp;      /* ラベル･データファイルの FILE 構造体
                              のポインタ */
```

○ラベル･データの構造体の領域をフリーにする関数

```
void WbtFreeLABEL( label )
WbtLabel            *label ;
```

2.4 Wave Scale Widget

時間軸のスケールを表示するwidget。transform widgetをスーパー・クラスにすることによって、波形、ラベル等のwidgetと同期し拡大/縮小、スクロールを可能としている。

○ Widgetの定義

クラス名	:	WaveScale
クラス・レコード	:	wbtWaveScaleWidgetClass
スーパー・クラス	:	Transform
リソース	:	下記の全てのNameの頭にWbtNが付く

Attributes Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	フォントの高さによる	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	表示スケール全体の長さ	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	wave scale widget内部で使用	
scaleY	float	wave scale widget内部で使用	
deltaX	float	wave scale widget内部で使用	
deltaY	float	wave scale widget内部で使用	
scaleXInherit	Widget	NULL	定数(scaleX)を継承するWidget ID
scaleYInherit	Widget	NULL	定数(scaleY)を継承するWidget ID
deltaXInherit	Widget	NULL	定数(deltaX)を継承するWidget ID

deltaYInherit	Widget	NULL	定数(deltaY)を継承するWidget ID
scaleXAcceptResize	Boolean	False	width (幅)のリサイズではscaleXは変更しない。
scaleYAcceptResize	Boolean	True	height (高さ)のリサイズでscaleYは変更する
windowGravity	int	SouthWestGravity	
scaleXUnit	char*	"msec"	X座標の単位
resizeCallback	XtCallbackList	NULL	wave scale widgetの内部で使用
exposeCallback	XtCallbackList	NULL	wave scale widgetの内部で使用 (以上はTransformから)
font	XFontStruct*	fixed	スケールのフォント名
frequency	float	12.0	音声データのサンプル周波数(khz)
msecPerPixel	float	10.0	時間軸の表示スケール(1ピクセル当りのミリセカンド)
startPosition	int	0	音声データ始点 (waveData配列の添字)
endPosition	int	0	音声データ終点 (waveData配列の添字)
scaleStep	int	0	時間軸のスケール間隔
scaleUnit	char*	"msec"	時間軸の表示スケール単位
tickStep	int	0	時間軸のテック間隔
tickUnit	char*	"msec"	時間軸のテック表示単位

● scaleStepは、scaleUnitを単位とした値である。tickStepは、tickUnitを単位とした値である。ウィンドウの高さは、低くともフォントの高さの2倍+2ピクセル分必要とする。

2.5 Log Power Widget

MakeAutoCorrelation (音声データに対する相関計算)により計算された相関データを基に、ログ・パワーを計算して画面上に表示する widget。transform widget をスーパー・クラスにすることによって、波形、ラベル等の widget と同期し拡大/縮小を可能としている。

○ Widget の定義

クラス名	:	LogPower
クラス・レコード	:	wbtLogpowertWidgetClass
スーパー・クラス	:	Transform
リソース	:	下記の全ての Name の頭に WbtN が付く

Attributes Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	100	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	フレーム数 (numberOfFrame)	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	LogPower widget 内部で使用	
scaleY	float	LogPower widget 内部で使用	
deltaX	float	LogPower widget 内部で使用	
deltaY	float	LogPower widget 内部で使用	
scaleXInherit	Widget	NULL	定数(scaleX)を継承する Widget ID
scaleYInherit	Widget	NULL	定数(scaleY)を継承する Widget ID
deltaXInherit	Widget	NULL	定数(deltaX)を継承する Widget ID

deltaYInherit	Widget	NULL	定数(deltaY)を継承するWidget ID
scaleXAcceptResize	Boolean	False	width (幅)のリサイズではscaleXは変更しない。
scaleYAcceptResize	Boolean	True	height (高さ)のリサイズでscaleYは変更する
windowGravity	int	SouthWestGravity	
resizeCallback	XtCallbackList	NULL	LogPower widgetの内部で使用
exposeCallback	XtCallbackList	NULL	LogPower widgetの内部で使用 (以上はTransformから)
correlData	float*	NULL	コーレル(相関計算)データ
windowLength	int	256	分析ウィンドウ長
frameShift	int	64	シフト数
numberOfFrame	int	0	フレーム数 (音声データの数/シフト数)
frequency	float	12.0	音声データのサンプル周波数(khz)
msecPerPixel	float	10.0	時間軸の表示スケール

2.6 Running Spectorogram Widget

MakeAutoCorrelation (音声データに対する相関計算)により計算された相関データを基に、ランニング·スペクトラムを計算して画面上に表示する widget。transform widget をスーパー·クラスにすることによって、波形、ラベル等の widget と同期し拡大/縮小を可能としている。

○ Widget の定義

クラス名	:	RunningSpect
クラス·レコード	:	wbtRunningSpectWidgetClass
スーパー·クラス	:	Transform
リソース	:	下記の全ての Name の頭に WbtN が付く

Attributes Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	100	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	フレーム数 (numberOfFrame)	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	RunningSpect widget 内部で使用	
scaleY	float	RunningSpect widget 内部で使用	
deltaX	float	RunningSpect widget 内部で使用	
deltaY	float	RunningSpect widget 内部で使用	
scaleXInherit	Widget	NULL	定数(scaleX)を継承する Widget ID
scaleYInherit	Widget	NULL	定数(scaleY)を継承する Widget ID

deltaXInherit	Widget	NULL	定数(deltaX)を継承する Widget ID
deltaYInherit	Widget	NULL	定数(deltaY)を継承する Widget ID
scaleXAcceptResize	Boolean	False	width (幅)のリサイズでは scaleXは変更しない。
scaleYAcceptResize	Boolean	True	height (高さ)のリサイズで scaleY は変更する
windowGravity	int	SouthWestGravity	
resizeCallback	XtCallbackList	NULL	RunningSpect widget の内部で使用
exposeCallback	XtCallbackList	NULL	RunningSpect widget の内部で使用 (以上は Transform から)
correlData	float*	NULL	コレル(相関計算)データ
lpcOrder	int	13	LPC次数
frameShift	int	64	シフト数
numberOfFrame	int	0	フレーム数 (音声データの数/シフト数)
frequency	float	12.0	音声データのサンプル周波数(khz)
msecPerPixel	float	10.0	時間軸の表示スケール

2.7 FFT slice Widget

音声データとFFT分析を行う音声データのポイントに基づき、FFTスライスを画面上に表示するwidget。

○ Widgetの定義

クラス名	:	FFTslice
クラス･レコード	:	wbtFFTsliceWidgetClass
スーパー･クラス	:	Transform
リソース	:	下記の全てのNameの頭にWbtNが付く

Attributes Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	100	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	100	
x	int	0	
y	int	0	
Widget Specific Attribute			
scaleX	float	FFTslice widget内部で使用	
scaleY	float	FFTslice widget内部で使用	
deltaX	float	FFTslice widget内部で使用	
deltaY	float	FFTslice widget内部で使用	
scaleXAcceptResize	Boolean	True	width(幅)のリサイズでscaleXは変更する
scaleYAcceptResize	Boolean	True	height(高さ)のリサイズでscaleYは変更する
windowGravity	int	SouthWestGravity	
resizeCallback	XtCallbackList	NULL	FFTslice widgetの内部で使用

exposeCallback	XtCallbackList	NULL	FFTslice widgetの内部で使用 (以上はTransformから)
waveData	short*	NULL	音声データ
waveDataSize	int	0	音声データの数
fftLength	int	512	FFTレングス
analyzeWindow	char*	"ham"	分析ウィンドウ (ham,han,rect)
anlWindowLen	int	256	分析ウィンドウ·レングス
analyzePoint	int	0	FFT分析ポイント

※ analyzePoint = FFT分析を行う音声データの位置 (short配列の添字)

2.8 LPC slice Widget

MakeAutoCorrelation (音声データに対する相関計算)により計算された相関データと、LPC分析を行う音声データのポイントに基づき、LPCスライスを画面上に表示するwidget。

○ Widgetの定義

クラス名	:	LPCslice
クラス･レコード	:	wbtLPCsliceWidgetClass
スーパー･クラス	:	Transform
リソース	:	下記の全てのNameの頭にWbtNが付く

Attribute Name	Type	Default	Description
Core Attributes			
background	pixel	White	
border	pixel	Black	
borderWidth	int	1	
cursor	Cursor	None	
destroyCallback	Pointer	NULL	
foreground	pixel	Black	
height	int	100	
insensitiveBorder	Pixmap	Gray	
mappedWhenManaged	Boolean	True	
sensitive	Boolean	True	
width	int	100	
x	int	0	
y	int	0	
Widget Specific Attributes			
scaleX	float	LPCslice widget内部で使用	
scaleY	float	LPCslice widget内部で使用	
deltaX	float	LPCslice widget内部で使用	
deltaY	float	LPCslice widget内部で使用	
scaleXAcceptResize	Boolean	True	width (幅)のリサイズでscaleXは変更する
scaleYAcceptResize	Boolean	True	height (高さ)のリサイズでscaleYは変更する
windowGravity	int	SouthWestGravity	
resizeCallback	XtCallbackList	NULL	LPCslice widgetの内部で使用

exposeCallback	XtCallbackList	NULL	LPCslice widgetの内部で使用 (以上はTransformから)
correlData	float*	NULL	コーレル(相関計算)データ
lpcOrder	int	13	LPC次数
numberOfFrame	int	0	フレーム数(音声データの数/シフト数)
analyzePoint	int	0	LPC分析ポイント

※ analyzePoint = (フレーム数) * (音声データの位置) / (音声データの数)

2.9 Spectrogram Widget

MakeSpectrogramにより計算された実数型のパワースペクトラムの配列を、X Window SystemのBitmap配列に変換する。そしてそのBitmapを画面上に表示し、ウィンドゥのマネージメントを行う。

○ Widgetの定義

クラス名	:	Spectrogram
クラス･レコード	:	wbtSpectrogramWidgetClass
スーパー･クラス	:	Transform
リソース	:	下記の全てのNameの頭にWbtNが付く

Name	Type	Default Description
Core Attributes		
background	pixel	White
border	pixel	Black
borderWidth	int	1
cursor	Cursor	None
destroyCallback	Pointer	NULL
foreground	pixel	Black
height	int	(dataYPixel * numberPerFrame)の値が自動セットされる。
insensitiveBorder	Pixmap	Gray
mappedWhenManaged	Boolean	True
sensitive	Boolean	True
width	int	1000
x	int	0
y	int	0
Widget Specific Attributes		
scaleX	float	Spectrogram widget内部で使用
scaleY	float	Spectrogram widget内部で使用
deltaX	float	Spectrogram widget内部で使用
deltaY	float	Spectrogram widget内部で使用
scaleXAcceptResize	Boolean	False
scaleYAcceptResize	Boolean	False
windowGravity	int	SouthWestGravity
resizeCallback	XtCallbackList	Spectrogram widget内部で使用
exposeCallback	XtCallbackList	Spectrogram widget内部で使用
		(以上はTransformから)

spectData	float*	NULL	パワースペクトラムデータ
numberOfFrame	int	0	全フレーム数 ファイルサイズ /(numberPerFrame * sizeof(float))
numberPerFrame	int	65	1フレーム内の要素数
startFrame	int	0	スタートフレーム番号
endFrame	int	0	エンドフレーム番号
dataXPixel	int	4	1要素に対応するX方向のピクセル数
dataYPixel	int	4	1要素に対応するY方向のピクセル数
frequency	int	12	最大サンプリング周波数 (目盛りの最大値は、frequency/2)
shiftTime	float	2.5	サンプリングシフト時間 (フレームとフレームの間の時間)
msecPerPixel	int	100	時間目盛りの数字表示間隔
framePosition	int	0	表示開始位置(フレームで指定)

Chapter 3

Creation of the Application used WBTWidgets

サンプル・プログラム
(spect_view.c)

```
/*
 *
 *          This is sample program to create Spectrogram
 *
 *          Compilation :
 *          % cc -o spect_view spect_view.o Spectrogram.o Transform.o
 *
 *
 */

#include <stdio.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>

/* X11用、各ヘッダ・ファイル */
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/DwtAppl.h>
#include <X11/Xatom.h>
#include <X11/DECWmHints.h>
#include <X11/decwcursor.h>

#include "Transform.h"      /* Transform Widget用のヘッダ・ファイル */
#include "Spectrogram.h"    /* Spectrogram Widget 用のヘッダ・ファイル */

#define  WIDTH   1000
#define  HEIGHT 200

int      nod = 65;
int      nf;                /* フレーム数 */
int      sf, ef;            /* スタート・フレームとエンド・フレーム */
int      xpixels, ypixels;  /* ビット・マップ作成時に使用するピクセル値 */
int      samplingrate;      /* サンプリング周波数 */
float    st, sff;
int      d;
float    *spectrum;         /* spectrum 計算データ */

Widget   toplevel;          /* Shell Widget */
Widget   dialog_box;        /* Dialog Box Widget */
Widget   Spectrogram;       /* Spectrogram Widget */

main (argc, argv)
int      argc;
char     *argv[];
{
```

```
int                 i, j, k;
int                 fp;
double              pow();
struct stat                 file__stat;

/* Shell Widget のイニシャライズ */
toplevel = XtInitialize("Spectrogram", "Spect", 0,0,&argc, argv);


/* Read Parameter file */
if (argc != 8) {
    printf("%c%cargument error ??¥n",'¥007','¥007');
    exit();
}
sscanf(argv[2],"%d",&xpixels);
sscanf(argv[3],"%d",&ypixels);
sscanf(argv[4],"%d",&samplingrate);
sscanf(argv[5],"%f",&st);
sscanf(argv[6],"%f",&sff);
sscanf(argv[7],"%d",&d);

if ((stat  (argv[1], &file__stat)) < 0) {
    perror("Stat Failure");
    exit(0);
}

nf = file__stat.st__size / (nod * sizeof(float));

spectrum = (float *)calloc(nf * nod, sizeof(float));

/* データ・ファイルのオープン */
if ((fp = open(argv[1], O__RDONLY)) < 0) {
    perror("Open Failure");
    exit(0);
}
sf = 0;
ef = nf - 1;

j = 0;
for (i = 0; i < nf; i++) {
    if((read(fp, &spectrum[j], nod*sizeof(float))) < 0)
        perror("Read Error");
    j += nod;
}
close(fp);

/* イベントのバッファリングを無効にする */
XSynchronize(XtDisplay(toplevel),1);
```

```
        /* 実際にSpectrogram Widgetを作成する関数 */
        create__spect();

        /* それ自身とそれに管理されているWidgetの表示 */
        XtRealizeWidget(toplevel);

        /* ユーザからの入力待ちとなる無限ループ */
        XtMainLoop();

}

create__spect()     /* Widget作成用関数 */
{
        int     i;
        Arg     Argl[15];

        /* Dialog Box Widgetに対するリソースの設定 */
        i = 0;
        XtSetArg(Argl[i], DwtNx, 0); i++;
        XtSetArg(Argl[i], DwtNy, 200); i++;

        /* Spectrogram Widget表示の為の Dialog Box Widgetの作成 */
        dialog__box = DwtDialogBoxCreate (toplevel,
                                          "Spect__Dialog__Box",
                                          Argl,
                                          i);

        /* Spectrogram Widgetに対するリソースの設定 */

        i = 0;
        XtSetArg(Argl[i], WbtNx, 0); i++;
        XtSetArg(Argl[i], WbtNy, 0); i++;
        XtSetArg(Argl[i], WbtNwidth, WIDTH); i++;
        XtSetArg(Argl[i], WbtNheight, HEIGHT); i++;
        XtSetArg(Argl[i], WbtNborderWidth, 1); i++;
        XtSetArg(Argl[i], WbtNspectData, spectrum); i++;
        XtSetArg(Argl[i], WbtNnumberOfFrame, nf); i++;
        XtSetArg(Argl[i], WbtNstartFrame, sf); i++;
        XtSetArg(Argl[i], WbtNendFrame, ef); i++;
        XtSetArg(Argl[i], WbtNdataXPixel, xpixels); i++;
        XtSetArg(Argl[i], WbtNdataYPixel, ypixels); i++;
        XtSetArg(Argl[i], WbtNfrequency, samplingrate); i++;
        XtSetArg(Argl[i], WbtNmsecPerPixel, d ); i++;
        XtSetArg(Argl[i], WbtNshiftTime, *((int*)(&sff))); i++;

        /* Dialog Box Widgetに表示する Spectrogram Widgetの作成 */
        Spectrogram = XtCreateWidget ("Spectrogram",
                                      wbtSpectrogramWidgetClass,
                                      dialog__box,
```

```
                                                    Argl,
                                                    i);
        /* Dialog Box Widget の管理 */
        XtManageChild (dialog__box);

        /* Spectrogram Widget の管理 */
        XtManageChild (Spectrogram);
}
```

○ XtInitialize について

toplevel = XtInitialize("Spectrogram", "Spect", 0,0,&argc, argv) ;

XtIntrinsics の関数群をコールする前には、必ずXtIntrinsics をイニシャライズしなければならない。関数XtInitialize は、ディスプレイ・サーバに対する接続を行い、アプリケーションが要求するコマンド行を解析し、リソース・データベースをロードし、アプリケーションに対する親のような役目をするシェル・ウィジェットをつくる。
XtInitialize はアプリケーション実行時に、ユーザがそのリソース(例えばフォントやカラーなど)に対して要求した内容を、コマンド行を解析することによって実現している。
XtIntrinsics の記述は以下の通りで、Widget という型の値を返す。

```
Widget XtInitialize(name, class_name, options, num_options, argc, argv)
String                  name;
String                  class_name;
XrmOptionDescRec        options[];
Cardinal                num_options;
Cardinal                *argc;
String                  argv[];
```

name	シェル・ウィジェット・インスタンスの名前を記述する。“main“と書くのが一般である。この名前はXtIntrinsics がこのシェル・ウィジェットに常駐しているリソースを検索するのに使用される。
class_name	アプリケーションのすべてのインスタンスに対する一般名である、アプリケーション・クラス名を指定する。アプリケーション名の最初の文字による交換によって設定される。この名前によってリソース・データベースを設定したファイルが指定される。

options	アプリケーションで特別に定義されたリソースに対し、コマンド行を如何に解析するかを指定する。例えばXrmParseCommandなどがある。
num_options	コマンド行における登録数を指定する。
argc	コマンド行のパラメータ数へのポインタ。
argv	コマンド行。

Xt Intrinsicsをイニシャライズするだけであれば、XtToolkitInitialize関数が利用できる。これは、XtInitializeほど便利ではないが、より柔軟性がある。というのも使おうとするシェル・ウィジェットを指定しなくてもよいからである。これはディスプレイをオープンしないし、またアプリケーション・シェルの作成も行わない。よってこの関数を使用するときはXtOpenDisplayとXtAppCreateShellを使用しなければならない。

○　XtSetArg について

```
i = 0;
XtSetArg(Argl[i], WbtNx, 0); i++;
XtSetArg(Argl[i], WbtNy, 0); i++;
XtSetArg(Argl[i], WbtNwidth, WIDTH); i++;
XtSetArg(Argl[i], WbtNheight, HEIGHT); i++;
XtSetArg(Argl[i], WbtNborderWidth, 1); i++;
XtSetArg(Argl[i], WbtNspectData, spectrum); i++;
XtSetArg(Argl[i], WbtNnumberOfFrame, nf); i++;
XtSetArg(Argl[i], WbtNstartFrame, sf); i++;
XtSetArg(Argl[i], WbtNendFrame, ef); i++;
XtSetArg(Argl[i], WbtNdataXPixel, xpixels); i++;
XtSetArg(Argl[i], WbtNdataYPixel, ypixels); i++;
XtSetArg(Argl[i], WbtNfrequency, samplingrate); i++;
XtSetArg(Argl[i], WbtNmsecPerPixel, d ); i++;
XtSetArg(Argl[i], WbtNshiftTime, *((int*)(&sff))); i++;
```

Widgetはその外観や機能をコントロールするアーギュメント･リストを受け取る。これは、Widgetのリソースと値のセットから成り立っている。Widgetに受け入れられるリソースのリストは、そのWidget自身にユニークなものだけでなく、他のWidgetから継承されるものも含まれている。

アーギュメント･リストを構成する要素には、このマクロXtSetArgを使用する。

```
XtSetArg(arg, name, value)
Arg             arg;
String          name;
XtArgVal        value;
```

arg	リソースの名前と値が対になった組合せである。
name	リソースの名前を指定する。
value	XtArgVal構造体に適合するリソースの値または、アドレスを指定する。

ここで注意しないといけないのは、アーギュメント・リスト数のカウンタであるiの値をXtSetArgの中で変更してはいけないということである。なぜならXtSetArgは関数ではなく、マクロなのである。

○ XtCreateWidget と XtManageChild について

```
Spectrogram = XtCreateWidget ("Spectrogram",
                              wbtSpectrogramWidgetClass,
                              dialog__box,
                              Argl,
                              i);
XtManageChild (Spectrogram);
```

アーギュメント･リストの設定が終了したら、Widget Instance を作成することが可能である。これには、XtCreateWidget を使用する。この関数は、Widget型の構造体へのポインタを返す。そしてそのポインタの値を利用して、XtManageChild にて作成したWidget の管理を行う。文法は以下の通りである。

```
Widget XtCreateWidget (name, widget__class, parent, args, num__args)
String          name;
WidgetClass     widget__class;
Widget          parent;
ArgList         args;
Cardinal        num__args;
```

name	作成されたWidgetのリソース名を設定する。この名前はリソースを呼び出す為に使われ、よってもしこのWidgetの兄弟Widgetとリソースを共有するが、固有の値を設定しなければならないものがある時は、異なった名前を使用するべきである。
widget__class	作成されるWidgetのクラスへのポインタを設定する。
parent	親Widgetを設定する。
args	リソースのデフォルト値を変更し、明示的に値を設定するためのアーギュメント･リストを設定する。

num_args	args のアーギュメント数を設定する。スタティックに定義されたリストであれば、XtNumber マクロを使用してこれを自動的に計算することもできる。

○　XtRealizeWidget と XtMainLoop について

```
XtRealizeWidget(toplevel);
XtMainLoop();
```

すべてのWidgetが作成、そしてリンクされ、それらの木構造が出来上がったらこれら2つの関数を使用して、実際に目に見えるようにする。

シンタクスは

```
void XtRealizeWidget(w)
Widget          w;
```

w　　Widgetを設定する。

```
void XtMainLoop()
```

○　リンク方法について

X11では、Widgetプログラム、Widgetを作成、管理するプログラムをそれぞれコンパイルした後、X11用のライブラリとのリンクが必要である。

●libX11.a	X11関数用のライブラリ
●libXt.a	Xt Intrinsics用のライブラリ
●libXaw.a	AthenaWidget用のライブラリ
●libdwt.a	DECwindowsWidget用のライブラリ
●libXm.a	OSF/Motif用のライブラリ

○ Default file の作成について

Widgetのリソースの設定は、アプリケーションにおけるアーギュメント･リストを使用した形で設定する以外にも方法はある。Default fileというASCIIキャラクタの集まったファイルを使用するのである。このファイルは各ユーザごとに作成可能であり、これを使用することによりユーザごとに異なった環境を設定することができる。プログラムを作成するとき、以下のことを参考にしてdefault fileかアプリケーション･プログラムの中のどちらで設定するのかを決定して頂きたい。

● Default file を使用することにより、ユーザごとの設定ができるので、システム管理者はアプリケーションをシステム･ワイドに設定することができる。

● プログラム中でのリソースの設定は最上位のものであり、これらは一般ユーザからは直接に書き換えられない。

● Default file の使用により、迅速な開発ができる。default file の書き換えによるリソースの変更は、テキスト･エディタを使用するのみであり、コンパイル、リンクのやり直しを必要としない。

● Default file の使用は、プログラムを簡潔に記述できる。Default file 中のリソースは文字列として取り扱われる。リソースがプログラム中にて設定されれば、幾つかの計算を要求する内在する構文を使用しなければならない。

● オプションを設定することは余計なオペレーションを行わなければならない。Default file を読み込みその内容を変換することは、そのプロセスにより多くのオーバー･ヘッドを与える。

カスタマイズには次の7つの方法がある。

① /usr/lib/X11/app-defaults/<class> の中にシステム･ワイドな設定をしているdefault file が存在し、それを設定する。<class> はアプリケーションのクラス名である。

② 環境変数 XAPPRESDIR が設定されており、$XAPPRESDIR/<class> の中のdefault file を設定する。<class> はアプリケーションのクラス名である。

③ リソース･マネージャ プロパティのデータをロードする。

④ リソース･マネージャ プロパティが存在しなければ、$HOME/.Xdefaults をロードする。

⑤ $XENVIRONMENT で指定されたファイルをロードする。

⑥ $XENVIRONMENT が設定されていなければ、$HOME/.Xdefaults-<host> をロードする。<host>はクライアントが実行されているマシン名である。

⑦ コマンド行のオプションをロードする。

これらのファイルはどれも同じフォーマットである。

以下に標準コマンド行オプションとリソース名の対応表を記す。

Command-LineOption	*ResourceName*
+rv	reverseVideo
+synchronous	synchronous
-background	background
-bd	borderColor
-borderColor	borderColor
-bg	background
-borderwidth	TopLevelShell.borderWidth
-bw	TopLevelShell.borderWidth
-display	display
-fg	foreground
-foreground	foreground
-fn	font
-font	font
-geometry	TopLevelShell.geometry

-iconic	TopLevelShell.iconic
-name	name
-reverse	reverseVideo
-rv	reverseVideo
-selectionTimeout	selectionTimeout
-synchronous	synchronous
-tile	TopLevelShell.tile
=	TopLevelShell.geometry
-xrm	NULL

○　アプリケーションごとのDefault file の作成について

アプリケーションごとのDefault file は、アプリケーション開発者またはシステム管理者によって作成される。これらはシステムの中のアプリケーションが常駐している、

/usr/lib/X11/app-defaults

ディレクトリに置かれている。アプリケーション･プログラムは関数XtInitialize をコールする時、そのdefaults file を指定する。以下の例は、リソース値の内容をファイル

/usr/lib/X11/app-defaults/Spect

に従って設定するという例である。

```
toplevel = XtInitialize("Spectrogram", "Spect", 0,0,&argc, argv) ;
```

/usr/lib/X11/app-defaults/Spect の内容

```
*background :   black
*foreground :   white
```

○　ユーザごとのDefault file の作成について

各ユーザはそれぞれがアプリケーションを実行した時にリソースの値を設定するdefaults file を、ホーム･ディレクトリの中に作成することができる。ユーザごとのこのファイルはアプリケーションまたはシステムの設定値よりも優先する。

~/.Xdefaults の内容

```
*background :   red
*foreground :   blue
```

Chapter 4

Speech WorkBench

Version 4

Speech WorkBench Version4(以下wb4 と略す)は前章にて述べてきたX11 及びDECwindows の機能を生かし存分に利用して作成されている。よってこのアプリケーションを利用するユーザは、X11を使い慣れたプログラマ並みとまではいかなくても、ある程度X11 またはXToolkit に関する知識を持っていたほうが有利である。特に第3章の内容などは、カスタマイズなどを個別に行う時に、X11 の内在する機能を十分に生かすことができ、インターフェイスの幅を広げるであろう。

では、このwb4 の実際の使い方を見ていくことにする。

4.1 起動方法

起動のためのコマンドは、

```
% wb4
```

である。この実行形式のファイルは、

```
/usr/local/bin
```

に置かれているので、ユーザはこのディレクトリをサーチ･パスに入れておく必要がある。この実行時に3章の内容が生かされるのである。例えば、wb4 の横幅、高さそれぞれ1000ピクセル、300ピクセル、X座標、Y座標をそれぞれ10、100と指定するときは、コマンド行にて

```
% wb4 -geometry 1000x300+10+100
```

と指定する。またwb4 を任意のディスプレイに表示するときは、環境変数DISPLAY をsetenv コマンドにて設定するという方法もあるが、起動時にオプション -display の引数として指定することも可能である。コマンドは以下のようになる。

```
% wb4 -display atrv07:0
```

ここでの'atrv07:0' であるが、atrv07: までがサーバ名であり、それにつづく0 という数字はそのサーバにおけるディスプレイ･ナンバーを意

味している。通常各ワーク・ステーションにはディスプレイは1つしか接続されていないので、デフォルト値の0という数字が自動的に設定される。よってデフォルト値をわざわざ指定することはなく

% setenv DISPLAY atrv07:

というように、省略することも可能である。また、ウィンドウのフォアグラウンド、バックグラウンドの色をこのコマンド行にて設定する時には、

% wb4 -foreground red -background blue

と指定することも可能である。ここで注意して頂きたいことは、今これらのオプションをそれぞれ別々に設定する例を示したが、これらのオプションを同時に設定することも勿論可能である。

wb4を起動すると、まず指定したディスプレイにウィンドウを表示する。このウィンドウの最上部には、IconifyButton, TitleBar, LoweringButton, ResizeButtonの4つが左から右の順に並んでいるのがわかる。このそれぞれの機能は、以下の通りである。

IconifyButton	そのウィンドウをアイコン化する為のボタン
TitleBar	そのウィンドウをスクリーン上で移動する為のバー
LoweringButton	そのウィンドウを下に隠されているウィンドウの更に下に追いやるためのボタン
ResizeButton	そのウィンドウのサイズを変更するためのボタン

そしてこれらの下には、wb4に対するオペレーションのためのメニュー・バーが表示される。このメニュー・バーの中に、'File', 'Label', 'Speech', 'Zoom', 'Analyze', 'View'といったメニュー・ボタンがそれぞれ作成される。これらは、

File　音声ファイルに対して、読み込み、書き込みの操作を行なうためのセレクション･ボックスを表示する。また、このセッションを終了するためのボタンもマネージする。

Label　ラベルファイルを読み込むためのセレクション･ボックスを表示する。

Speech　読み込んだ音声ファイルをD/A変換することにより、その内容をスピーカから出力する。

Zoom　波形上で、指定された範囲を拡張、縮小または標準の縮尺に設定する。

Analyze　LPCSpectrumslice、RunningSpectrumを表示するために行う計算、Spectrogram表示のための計算を実行する。

View　音声データ、計算後のデータを基に、ラベル、時間軸スケール、ログ･パワー、ランニング･スペクトラム、FFTスライス、LPCスライス、スペクトログラムを表示または削除する。

4.2.1　音声データ･ファイルのロード ('File' ボタン)

wb4 は、音声ファイルをロードするということが、操作の始まりである。この一連の操作は、

① 'File' ボタンをクリック(マウスの左ボタンをプレス)し、ドラッグ(プレスしたままマウス･カーソルを移動)しながら'Wave Load' に合わせリリース(プレス状態を解除)する。そうすると、ファイル･セレクション･ボックスが表示される。

② ファイル･セレクション･ボックスの'File Filter' にて、

/MAU/DAT/D0/MAU*.NEW

の様にディレクトリ指定をし、ファイル名にマスクをかけ、'Filter' ボタンをクリックしてフィルタに通す。

③ フィルタのかけられたファイル名が一覧となって、ボックス内のスクロール･ウィンドウ内に表示される。

④ 目的のファイル名の上で一度クリックを行い、ファイル名を反転さす。そして'OK' ボタンをクリックしてそのファイルを選択する (Figure 4-2-1-1)。また他の方法として、ファイル名の上でダブル･クリックを行うというのがある。この方法を使用すれば、'OK' ボタンを使用しなくても良い。

このフィルタに通すという方法以外に、直接'Selection' にてファイル名を入力するという方法もある。

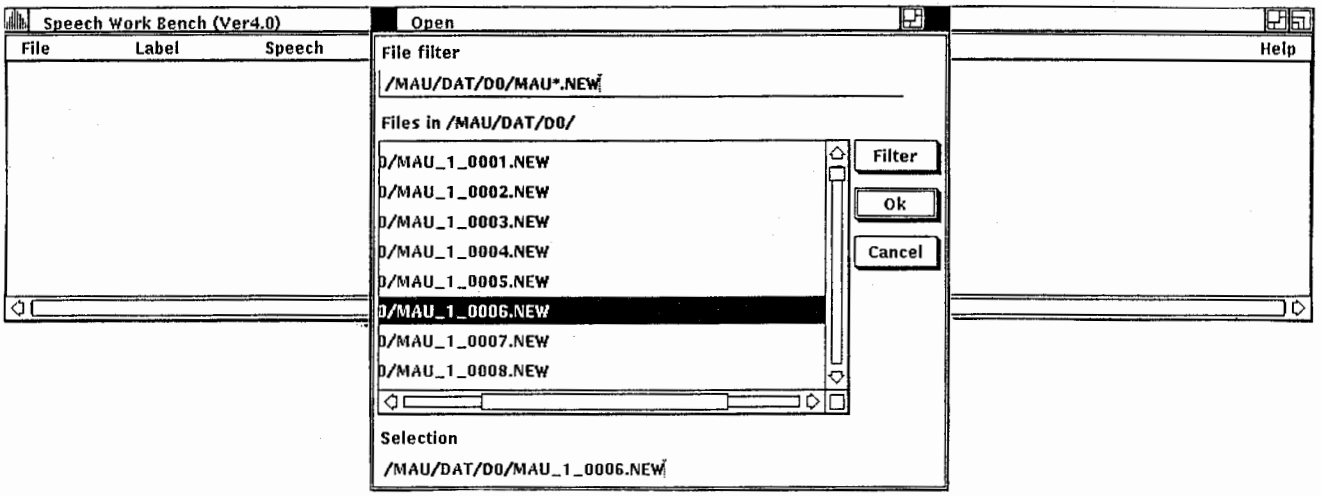
Speech Work Bench (Ver4.0)
File
Label
Speech
Help
Open
File filter
/MAU/DAT/D0/MAU*.NEW
Files in /MAU/DAT/D0/
D/MAU_1_0001.NEW
D/MAU_1_0002.NEW
D/MAU_1_0003.NEW
D/MAU_1_0004.NEW
D/MAU_1_0005.NEW
D/MAU_1_0006.NEW
D/MAU_1_0007.NEW
D/MAU_1_0008.NEW
Filter
Ok
Cancel
Selection
/MAU/DAT/D0/MAU_1_0006.NEW

Figure 4-2-1-1

4.2.2　音声データ･ファイルのセーブ ('File' ボタン)

音声ファイルをロードし、そのデータに対して編集操作を行えば、データをセーブする必要が生じてくる。この操作は以下のようにして行う。

① 'File' ボタンをクリックし、ドラッグしたまま'Save Whole as...' または'Save Selection as...' に合わせリリースする(Figure 4-2-2-1)。そうするとファイル･セレクション･ボックスが表示され、編集されたデータの内容の、全体または一部を指定したファイルにセーブすることができる。
② ファイルの選択は、4.2.1 と同様である。

データの全体、または一部というような選択は、波形表示ウィンドウ上でマウス･カーソルのクリックとドラッグの組合せで行う領域指定のことである(Figure 4-2-2-2)。これら音声データ編集機能の詳細は、Chapter 2のWave Widget を参考にして頂きたい。

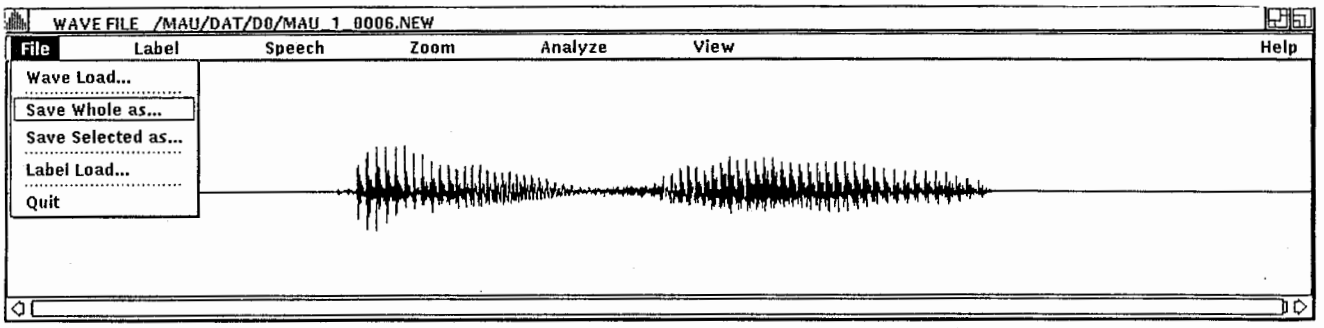

Figure 4-2-2-1

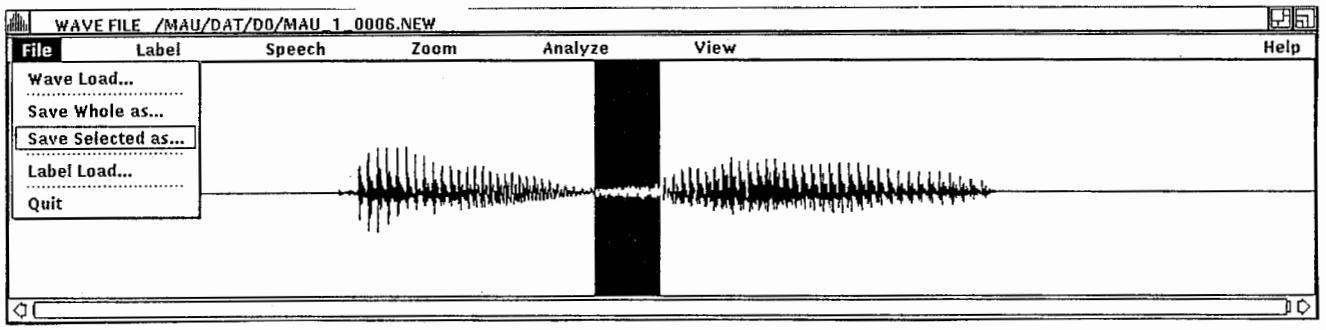

Figure 4-2-2-2

4.2.3　　ラベル･ファイルのロード

('File' ボタン、'Label' ボタン)

ラベル･ファイルのロードは、4.2.1、4.2.2 と同様の操作で実行される。このファイルを読み込まないことには勿論、音声ファイルに対応したラベル表示は不可能である。

この操作で注意しなければいけないことは、音声ファイルとラベル･ファイルが同じものでなければならないということである。例えば音声ファイル

/FFS/DAT/D0/FFS__1__0123.NEW

を読み込んだのであれば、当然ラベル･ファイルは

/FFS/LBL/D0/FFS__1__0123.LB

を読み込まなければならない。現在どの音声ファイルが読み込まれているかを知る方法であるが、これは音声ファイルの選択と同時にタイトル･バーにその名前がストアされるので、それを参考にして頂きたい。

4.2.4　　D/A 変換

('Speech' ボタン)

音声データの内容を実際にアナログ信号に変換し、スピーカを通してその音声を聴き確かめることができる。

これは'Speech' ボタンをクリックした時のメニューで実行する。これにはモードが3つ用意されており、その時点での、①音声ファイルの全体の内容、②範囲指定されている部分の内容、それと、③ウィンドウに表示されている部分の内容の何れかのモードで変換することができる。この3タイプの違いは、

① 範囲指定に関係なく、その時点でのデータ格納バッファの内容をすべて変換する。

② マウス･カーソルのクリック&ドラッグによる範囲指定された部分(Figure 4-2-4-1)のみを変換する。

③ その時点で、ウィンドウに表示されている部分のみの変換モードである。次のセクションにて説明をするが、wb4 には'Zoom' というメニューがあり、拡大/縮小機能が備わっている。そしてまた、スクロールという機能も備わっており、これらの機能によってウィンドウ上に表示されるデータは毎回違ってくる。このような時にこのモードを使用して、その都度表示されている部分のみを変換することは大変有用ではなかろうかと思われる。

変換時の周波数にはそれぞれ

12kHz　と

20kHz　が用意されている。

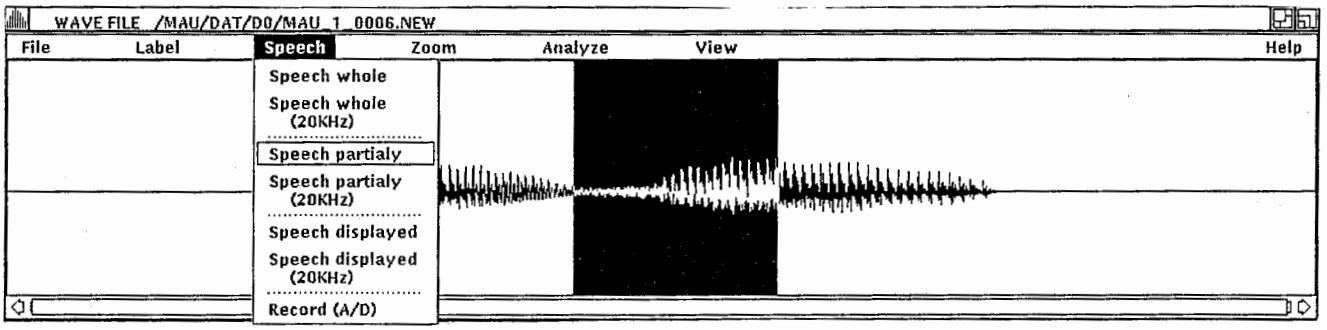
WAVE FILE /MAU/DAT/D0/MAU_1_0006.NEW
File
Label
Speech
Zoom
Analyze
View
Help
Speech whole
Speech whole (20KHz)
Speech partialy
Speech partialy (20KHz)
Speech displayed
Speech displayed (20KHz)
Record (A/D)

Figure 4-2-4-1

4.2.5　　拡大/縮小機能

('Zoom'ボタン)

wb4は前バージョンと違い、音声データのサイズ(時間)に関わらずその内容全てを1つのウィンドウ上に出力する様に設計されている。よって細部を詳しく調べたいというときは、その部分を拡大する必要がある。この機能は、

① 拡大したい部分をまずセレクトする。

② 'Zoom'ボタンをクリックし、'Zoom Up'でリリースする(Figure 4-2-5-1)。

③ すると、セレクトされている部分の始点と終点がそれぞれウィンドウの両端に拡大される(Figure 4-2-5-2)。

という手続きで行う。

またその逆で、縮小機能も用意されている。これは、

① 縮小したい大きさの範囲を同じように指定する。この時、ウィンドウの両端がその領域の幅に縮小される(Figure 4-2-5-3)。

② 縮小後の波形データのウィンドウ上での始点は、縮小前のそれと変化はない。

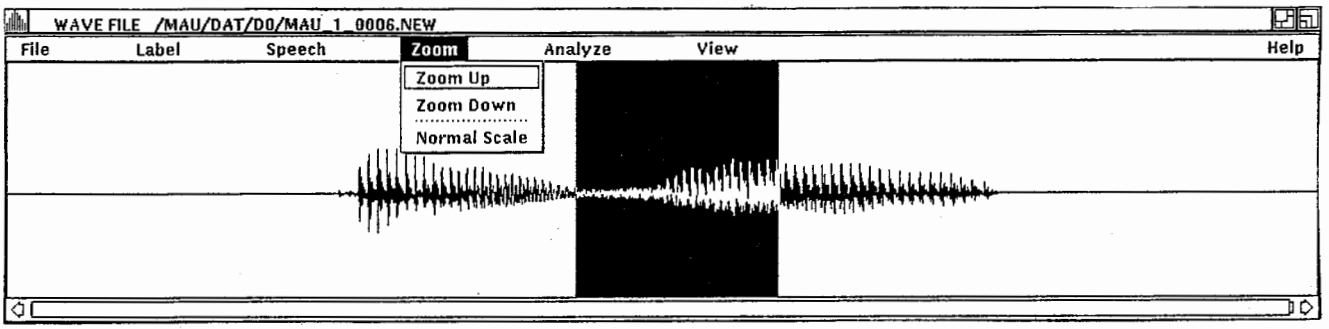

Figure 4-2-5-1

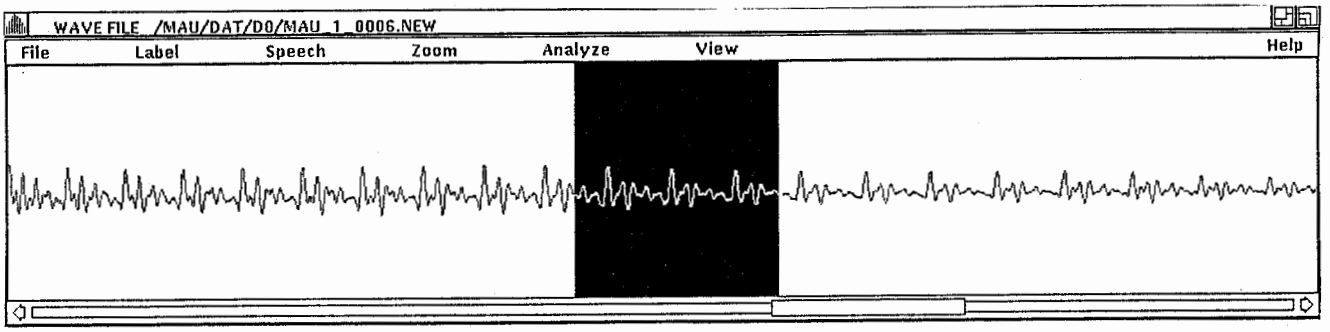

Figure 4-2-5-2

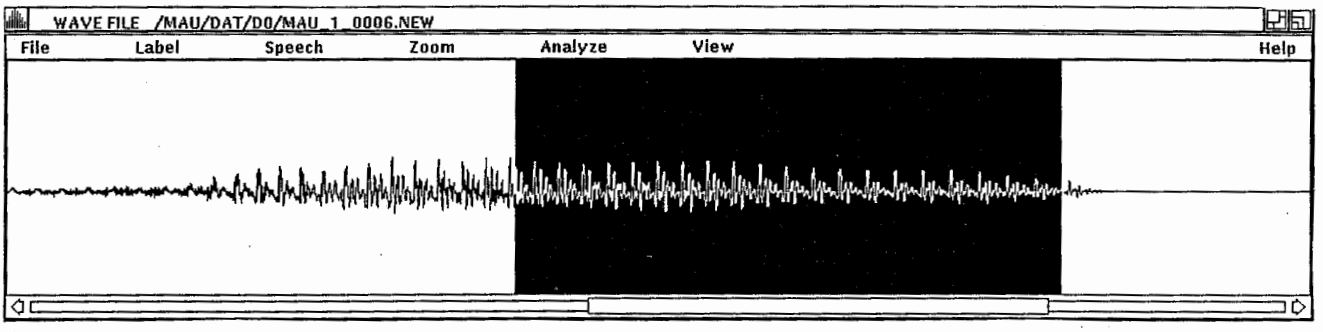
WAVE FILE /MAU/DAT/D0/MAU_1_0006.NEW
File
Label
Speech
Zoom
Analyze
View
Help

Figure 4-2-5-3

4.2.6　　スクロール

(スクロール･バー)

これはこのバージョンから新たに加わった機能であり、DECwindowsのスクロール･バー ウィジェットを使用している。スクロール･バーは、メイン･ウィンドウの下部に表示されている、左右に矢印を伴ったバーのことである。このバーと伸縮するパイプのセットでスクロール･バー･ウィジェットを形成している。

① 波形データの拡大を行えば、このパイプはもとの長さを拡大率分収縮した形で再表示される。バー上でのパイプの位置は、波形データのウィンドウ上での始点が全体のどの部分であるかによる。

② バーの両サイドの矢印をクリックするか、収縮したパイプの外のバーの部分をクリックするか、またはパイプを直接クリックして動かすかの3通りの方法でスクロールされる。

4.2.7　波形データのアナライズ ('Analyze' ボタン)

アナライズ機能は、4.2.8 にて説明するビュー機能を実現するための前準備である。それぞれのビューを表示するためには、この機能を用いてデータ･ファイルを作成してやらなければならない。

ログ･パワー、ランニング･スペクトラム、LPCスペクトラム･スライスなどを表示するそれぞれのWidget は、相関計算後のデータを基に作成されるので、波形データから算出計算された相関計算データが必要となってくる。

また、スペクトログラムWidget は、パワー･スペクトラム･データを基に作成される。

これらの操作は、

① 'Analyze' ボタンをクリックする(Figure 4-2-7-1)。
② ログ･パワー、ランニング･スペクトラム、LPCスペクトラム･スライスを表示するには'AutoCoreration' をセレクトする。
③ スペクトログラムを表示するには、'Spectrum' をセレクトする。

尚、これらの計算をせず、つまりデータを作成せずに、ビュー機能にてWidget を作成しようとすると、'Caution Box' と共にアナライズをするよう要求する。

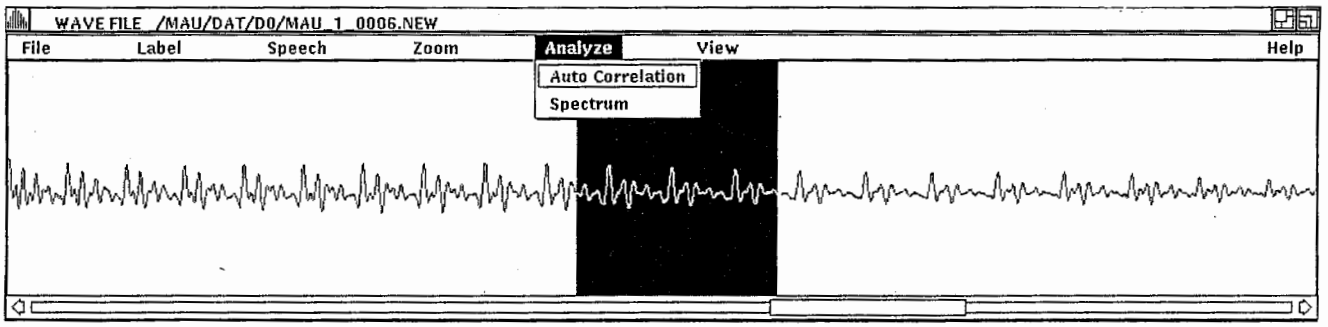
WAVE FILE /MAU/DAT/D0/MAU_1_0006.NEW
File
Label
Speech
Zoom
Analyze
View
Help
Auto Correlation
Spectrum

Figure 4-2-7-1

4.2.8　各Widgetの表示

('View'ボタン)

これには、ラベル、スケール、ログ･パワー、ランニング･スペクトラム、FFTスライス、LPCスライス、スペクトログラムを表示するためのメニューが用意されている。この機能を実現するには、

① まず、'View'ボタンをクリックする。

② 'Label'から'Spectrogram'に至るまで、それぞれボタンが現れる(Figure 4-2-8-1)。

③ ここで現れるボタンはトグル･ボタンであり、先程までのプッシュ･ボタンとは違う。

プッシュ･ボタンは、それを選択することにより、何度もその機能実行の動作を行うが、トグル･ボタンはその機能を実行するか、若しくは実行を取りやめるかの動作を交互に行う。この2つの動作を交互に行うということから、トグル･ボタンという名称がついている。

④ 一度ボタンをセレクトすればボタンの左部に、セレクトされているというマークがつく。そしてもう一度セレクトするとそのマークが消える。

⑤セレクトした時に、そのWidget作成のための条件が揃っていなければ、'Caution Box'と共にするべき処理を実行するよう促してくる。たとえば、ラベル表示をしようとするのにラベル･ファイルがロードされていなかった場合や、ランニング･スペクトラム表示の際に相関計算がされていなかった場合などである。

⑥ この様な場合には、その要求通りの操作をするべきである。そして再びWidgetを表示する時には、先程の動作で、トグル･ボタンがセレクトされているはずであるから、もう一度セレクトして、ボタンの左のマークを消してやらなければならない。

⑦　各Widget表示のボタンをセレクトしただけでは実際には目にみえない。メイン･ウィンドウをリサイズ･ボタンにて下方へ拡大してやらなければならない(Figure 4-2-8-2)。

⑧　メイン･ウィンドウをリサイズすることにより、セレクトされた順に各Widgetが表示されていることに気付く。この時点で再度ボタンをセレクトしWidgetの表示を取りやめると、そのWidgetはメイン･ウィンドウから消え去り、もしその下にWidgetが表示されていれば、自動的に上方へ再表示される。

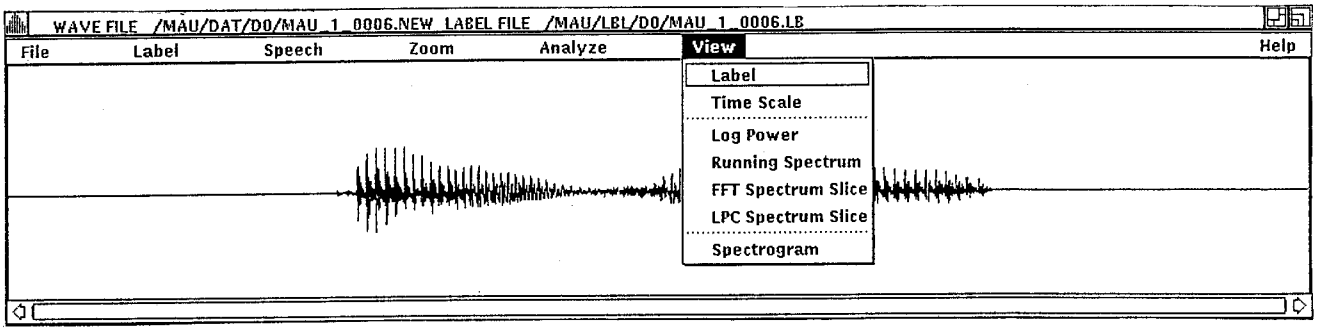

Figure 4-2-8-1

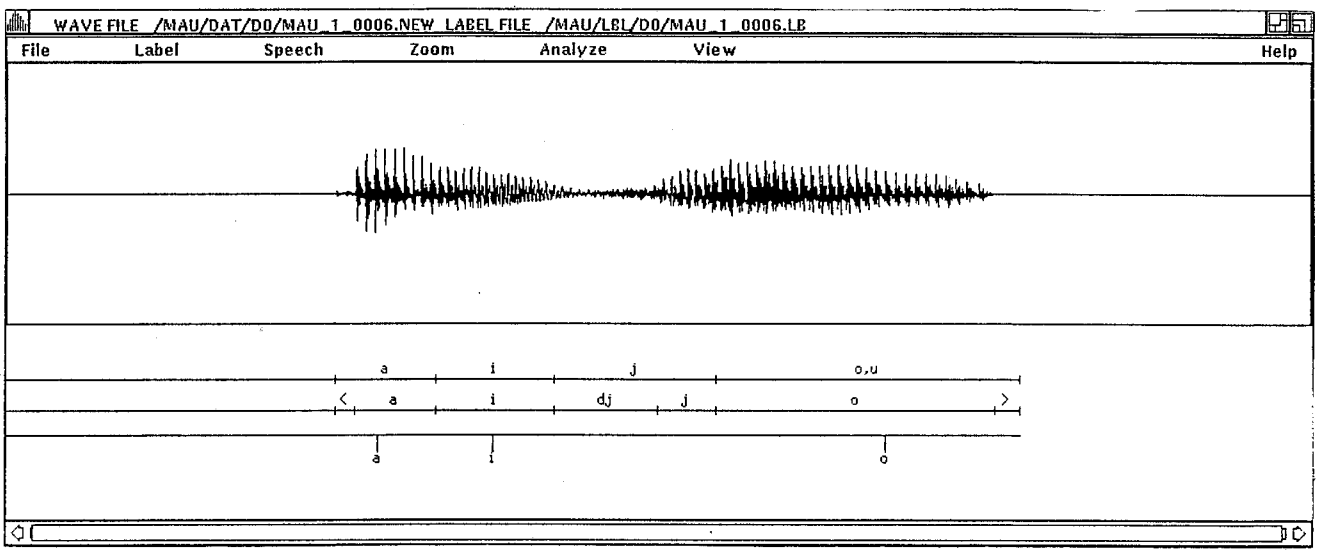

Figure 4-2-8-2

4.2.9　　リサイズ機能

メイン･ウィンドウに表示された、波形データを始めとする各Widgetは、メイン･ウィンドウのリサイズによってそれ自身のサイズも変える様に設計されている。

例えばメイン･ウィンドウのwidthが、1000ピクセルから800ピクセルへと縮小された時、その中に表示されていた各Widgetは、1ピクセル当たりの表示時間を1000/800倍にして表示しようとする。よってメイン･ウィンドウが何ピクセルに拡大/縮小されようともその中に表示されるWidgetの内容は同等のものとなる(Figure 4-2-9-1, Figure 4-2-9-2)。

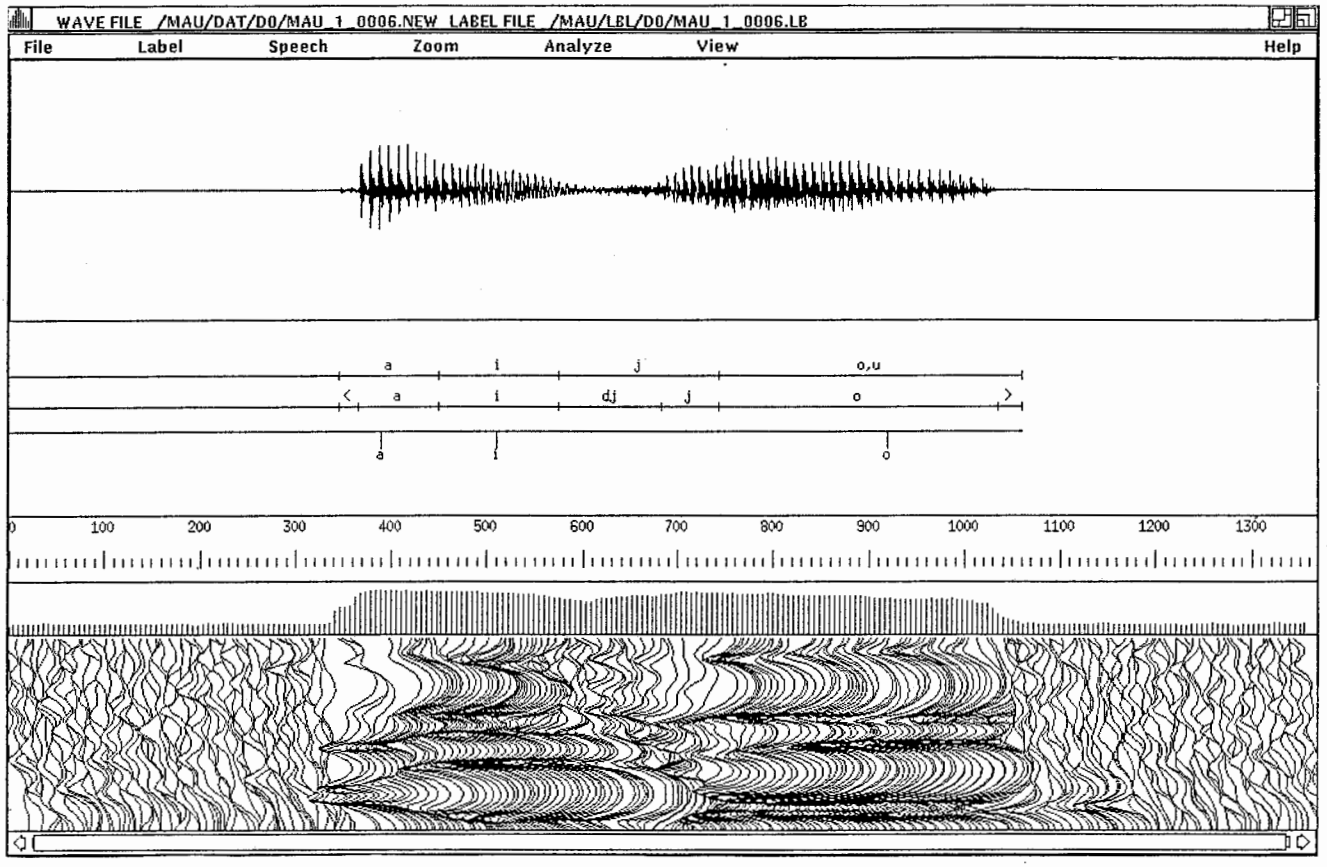
WAVE FILE /MAU/DAT/D0/MAU_1_0006.NEW LABEL FILE /MAU/LBL/D0/MAU_1_0006.LB
File
Label
Speech
Zoom
Analyze
View
Help
a
i
j
o,u
a
i
dj
j
o
a
i
o
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300

Figure 4-2-9-1

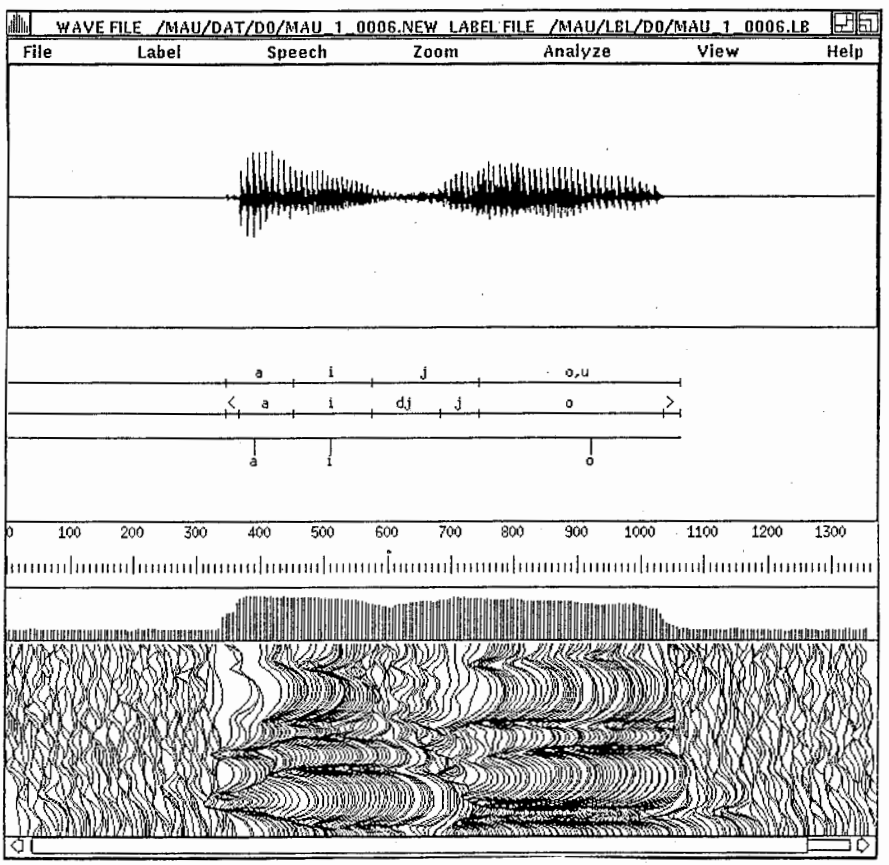
WAVE FILE /MAU/DAT/D0/MAU_1_0006.NEW LABEL FILE /MAU/LBL/D0/MAU_1_0006.LB
File
Label
Speech
Zoom
Analyze
View
Help
a
i
j
o,u
a
i
dj
j
o
a
i
o
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300

Figure 4-2-9-2

参考文献

INTRODUCTION TO THE X WINDOW SYSTEM

Oliver Jones

1989 Prentice-Hall,Inc

ISBN 0-13-499997-5

OSF/Motif Programmer's Guide

OPEN SOFTWARE FOUNDATION

1990 Prentice-Hall,Inc

ISBN 0-13-640525-8