ヒューマンインタフェース 61－6
（1995. 7. 20）

An Interface Toolkit with Dynamic Selectable Modality

川合 史朗　相田 仁　齊藤 忠夫

東京大学 工学部

概要　計算機の使用形態が多様化するに従って、固定したモダリティしか利用しないアプリケーションは、身体的障害や使用環境によってそのモードが使えないユーザにとっては役にたたなくなるという問題が出てきた。最適なインタフェースはユーザの環境によって左右され、しかも計算機の使用中に変化することが有り得る。本稿では、アプリケーション側ではインタフェースの意味を記述し、インタフェースを表現する部分を分離することで、ダイナミックにモダリティの切り替えを可能にしたツールキットについて述べる。

キーワード コンピュータ・ヒューマン・インタフェース、アシスティブテクノロジー、マルチモーダル・インタフェース

Shiro Kawai　Hitoshi Aida　Tadao Saito

Faculty of Engineering, The University of Tokyo

Abstract　It is a problem that people who have physical disabilities or who are in limited environment cannot get the benefits of fancy multimodal features of new applications which don't provide alternative interaction method. In fact, the most suitable interface differs for each user, and can be changed while he or she is using the computer. In this paper we describe a new user interface toolkit library which allows to change modalities dynamically, by separating representation of interface from semantics of interface.

Keywords　Computer-Human Interface, Assistive Technology, Multimodal Interface

1 はじめに

計算機－ユーザ間のコミュニケーションに、文字情報だけでなく画像情報や音声情報を用いることは普通に行なわれるようになってきている。それらの様々なモダリティを利用したアプリケーションは、テキストデータのみを頼りとしていた時代のアプリケーションに比べ、操作性、作業効率、学習のしやすさ等の面で格段に優れている。

同時に、計算機が人々の日常生活のなかで道具として使われることが多くなってきたため、「誰にでも」「いつでも」使えるアプリケーションがますます求められるようになってきている。

しかしその一方で、情報が多くのコミュニケーションチャネルに分散して伝達されるようになってくると、そのうちひとつでも利用できないコミュニケーションチャネルがあるユーザはアプリケーション全体を利用できなくなるという問題が生じてきた[7]。キャラクタベースのアプリケーションならば、全盲の人でも画面に表示されたキャラクタを音声や点字ディスプレイに出力する付加機器によって利用することができたが、ビットマップで出力されるグラフィックベースのアプリケーションではそのような方法は使えない。健常者であっても、例えば電車で移動しながらアプリケーションを使う場合、周囲の騒音レベルによっては聴覚情報が充分に伝達されないケースや、端末の能力差、情報ソースと端末間の通信帯域幅の制限などからもとの情報の提示がそのままでは不可能なケースなどが考えられる[6]。

また、ユーザが複数の計算機がある環境で作業をすることも一般的になってきており、デスクトップでの作業の続きを携帯端末で行うといったような、ダイナミックにインタフェースの環境を切り替えることも求められてきていると考えられる。

本稿では、これらの問題点を踏まえて、さまざまなモダリティを柔軟に切り替えることができるようにするためのツールキット fruit (Flexible and Rearrangeable User-Interface Toolkit) の構成について述べる。

2 要求される機能

2.1 ユーザの使用環境とインタフェース

計算機の使用形態が多様化するに従い、どのようなインタフェースが最適かということは一概には言えなくなってきている。健常者がデスクトップで作業する場合には快適なインタフェースであっても、次にあげるような要因によっては最適なインタフェースにはならない場合がある (図 1)。

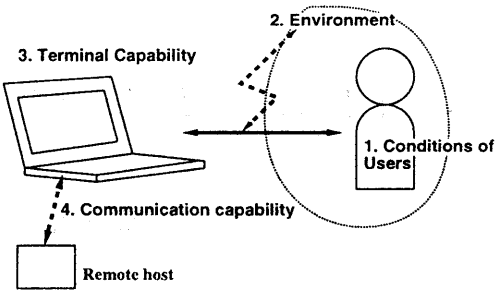

図 1: インタフェースに影響を与えるファクタ

1. ユーザの身体的状況
   - 現在のインタフェースは、ユーザは上肢が自由であることと視力を持っていることを前提としている。したがって、視覚に障害 (弱視、全盲) のある場合は通常のアプリケーションの出力を受け取ることが困難であり、また、上肢に障害をもつ場合は入力が困難である。
   - 聴覚障害、発声・発話の障害は、現在のアプリケーションではあまり問題になることはないが、音響入出力を扱うアプリケーションが増えて来ると問題になり得る。
   - Virtual Reality がインタフェースに採り入れられた場合、運動機能の障害を持つユーザは健常者と同じようにはインタラクトできない。
2. (主として一時的な) 使用環境に関するもの
   - 騒音のある環境や、逆に音を出せない状況では、音響入出力が使えない場合があるであろう。
   - 振動のある環境や、不安定な状態での使用、あるいは手がふさがっている状況では、キーボードやポインティングデバイスが通常と同じようには使えないであろう。
3. 端末のケーパビリティ
   - 限られた画面：精細な GUI やビットマップイメージの表示ができない可能性がある。
   - 音声入出力等のデバイスが備わっていない場合もある。
   - 限られた計算能力：動画やマルチメディアデータ等は、インタラクションの関係のない範囲で代替表現を用いるほうが良い場合も考えられる。

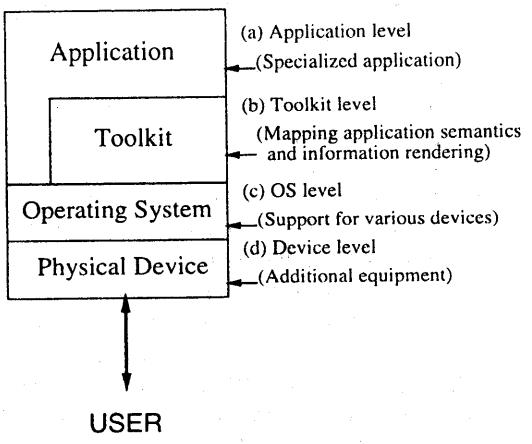

図 2: 代替コミュニケーションを用意するレベル

4. 通信路のケーパビリティ

- データソースが遠隔地にある場合、通信路容量の不足により情報量の大きいデータの扱いに障害が出る。

2.2 アプローチの検討

前節で述べたようなさまざまな状況に対応するためには、いくつかのインタラクションの方法を状況に応じて切り替えられるとよい。

どのレベルでその切り替えをサポートするかという選択にはいくつか考えられるが、整理すると図2のようになる。どのアプローチも、特に障害者対応の分野ではこれまで研究されて来ているが、それぞれ一長一短がある。

アプリケーションレベル

盲人用の音声ワードプロセッサ等では、音声チャネルを直接コントロールするコードを一般のアプリケーション中に埋め込んだり、初めから専用のアプリケーションとして開発されたものがいくつかある。このアプローチは、ユーザからの要請にもっとも適したインタフェースが設計できる、従来のパーソナルコンピュータの能力的制限をカバーできる等の利点があるが、元となるアプリケーションのバージョンアップやハードウェアの進歩に対応しづらい等、汎用性に欠ける。

OS レベル、デバイスレベル

OS レベルで複数のモダリティに対応することができれば、アプリケーションから独立した汎用的な選択機構が実現できる。既に製品化されたものもあり [9]、また入力に困難がある場合にはデバイスドライバレベルでの対応で充分な場合も多い [8, 10, 11]。この段階での対応の研究は数多く、優れた実績をあげている。

しかし、よりユーザに対して使いやすいインタフェースにするには、アプリケーションの実行コンテキストを考慮することが不可欠である (例えば、編集中の文章は文字どおり読み上げられることが必要であるが、操作メニューの項目についてはより簡潔な表現を用いて良い)。特に、グラフィカルインタフェースを用いたアプリケーションの出力のセマンティクスを従来の OS の段階で捕捉することは非常に困難であり、OS の API としてより高度な表現を採用するか、ツールキットレベルで対応する必要があると考えられる。

ツールキットレベル

原理的には、図3のようにグラフィカルユーザインタフェース (GUI)、キャラクタベースユーザインタフェース (CUI)[1]、音響的ユーザインタフェース (AUI)[2]等のツールキットとアプリケーション間のインタフェースを一様にしておけば、ユーザの状況や使用環境に応じて必要なモダリティを切り替えて使える、selectable multimodal interface が実現できる。

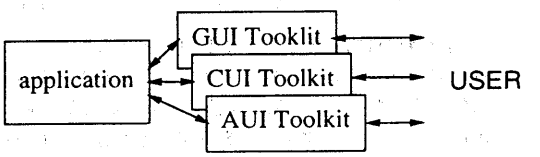

図 3: ツールキットベースアプローチ

また、特に移動端末からリモートホストのアプリケーションを使うケースを考えると、複数の異なる OS をまたがって情報をやりとりする必要がでてくる。そのようなケースも汎用的にサポートするにはツールキットベースのアプローチが有効であると考えられる。

[1]本稿では、キャラクタ端末のフルスクリーンで自由にカーソル移動を行うインタフェースの形態を CUI、UNIX のシェルのように行単位の入出力を行うものを Line oriented UI と呼ぶことにする。

[2]本稿では、AUI は auditory output と keyboard input の組み合わせを指すものとする。将来的には音声言語による入力も可能になるであろうが、現時点での視覚障害者の計算機使用形態を考えると入力は keyboard と考えるのが妥当であろう。ビンディスプレイによる出力は一種の Line oriented UI と考えられる。

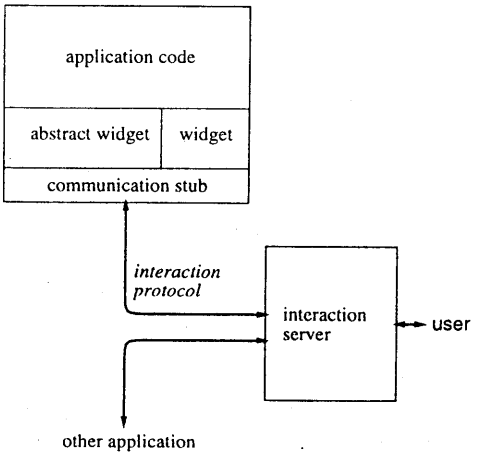

図 4: インタラクションサーバ

3 Fruit のアーキテクチャ

3.1 概要

インタフェースツールキット fruit は、ユーザの環境に合わせてダイナミックにインタフェースのモダリティを切り替えることを可能にするツールキットである。

実行時に自由にモダリティが切り替えられるためには、アプリケーションのインタフェースの意味を表す部分とそれを表現する部分が独立している必要がある。fruit では、インタフェースを表現する部分を interaction server として図 4のように独立させた。

アプリケーションプログラマはインタフェースの部品である widget や abstract widget(後述) の API を用いてプログラムを書き、fruit のライブラリとリンクする。実行時にインタフェースに対するリクエストが生じると、fruit のライブラリ中の communication stub はそのとき有効な interaction server にそのリクエストを送る。それを GUI で表現するのか、AUI で表現するのかは interaction server に任せられる。

また、communication stub は widget によって表現されたインタフェースの状態を保持しており、アプリケーションが動作中に接続中の interaction server が down して、他の interaction server につなぎ替えられた場合には widget の内部状態を通知することでその時点でのインタフェースの状態を再現する。

3.2 Meta-interaction object

GUI のツールキットの場合は、画面に表示される機能単位を widget と呼ばれるオブジェクトで実現するのが普通である。ユーザはマウスなどの物理デバイスを用いて、画面上のボタン (のように見えるもの) に操作を行う。このような、アプリケーションからソフトウェアによってインタラクションの基本的な単位として提示されるものを interaction object と呼ぶことにする。

GUI ツールキットの widget は、GUI の interaction object を表現するには良いモデルである。しかし、他のモダリティでのインタラクションを考えた場合、必ずしも GUI の widget が適切であるとは限らない。

例えば、ジオメトリ情報は auditory interface や line oriented inteface では全く必要ない。一方、GUI ではオブジェクトの状態は画面にいつも表示されているが、auditory interface では、音響情報が一過性のものであるという性質上、ユーザが必要なときにオブジェクトの状態を問い合わせるプロトコルを用意しておかなければならない。

したがって、アプリケーションが様々なモダリティのインタラクションに対応できるようになるためには、widget のような interaction object よりも抽象的で、インタラクションの意味を記述できるようなオブジェクトが必要である。これを meta-interaction object と呼ぶことにする。

Meta-interaction object, interaction object および phisical object の関係を表 1に示す。

3.3 Widget with multimodal enhancement

Interaction object は、fruit でもやはり widget とよぶクラス階層により実現される。アプリケーションから見た API はほとんど従来の GUI widget と変わらない。しかし、その表現形態は interaction server によって異なる。label widget は GUI では普通の label だが、AUI ではそのテキストが読み上げられる。

fruit のアプリケーションでは、アプリケーション側は widget が具体的にどのように表現されるかに関する情報は原則として持っていない。すべては interaction server 任せである。例えば、GUI において pushbutton をクリックするとボタンの色が変わる、といった動作はすべて interaction server 側で行われる。ただし、動作中に interaction server が切り替わった場合にもインタフェースの状態を再現するために、生成された widget とその内部状態、親子関係などはアプリケーション側に記憶されている。

実際に interaction server によって widget がどのよ

Meta-interaction object	インタラクションの意味を表すオブジェクト 例: “アプリケーションを終了させる”, “ファイル名を入力する”
Interaction object	ソフトウェアによって作り出された操作対象 例: Quit, Filename: sample.c
Phisical object	ユーザが直接操作する、あるいは直接目にする／耳にする対象 例: マウス、キーボード

表 1: Meta-interaction object と interaction objet の関係

うに表現されるかは、widget 本体が持つデフォルトの表現形式からだけでなく、その widget がインタラクションにおいてどのような意味を持っているかによっても影響を受ける。そのインタラクションの意味を記述するのが次の abstract widget である。

3.4 Abstract widget

fruit では meta-interaction object を表現するために、abstract widget というクラス階層を用意している。

例えば、アプリケーションを終了させるというインタラクションを考える。GUI では、“Quit” というプッシュボタンをマウスでクリックすることがアプリケーションを終了させるかもしれない。一方、キャラクタオリエンテッドなアプリケーションでは、キーボードから ‘q’ あるいは “quit<RET>” と入力することにより終了するかもしれない。このインタラクションは、どちらもおなじ abstract widget によりマネージされる。

Abstract widget は仮想的なオブジェクトであり、ユーザが目にするのはやはり通常の widget である。ただ、abstract widget はいくつかの widget の集合に対してその意味を与える。例えば、「ファイル名を入力する」という abstract widget は「一行のテキストを入力する」というクラスに属するが、それは通常一つの label widget と一つの entry widget、及びそれらを包含する一つの frame widget により構成される。

GUI ではその abstract widget は表 1の 2 段目にあるように表現されるであろう。CUI では label widget の文字列がプロンプト文字列になり、一行のテキスト入力を待つであろう。また、AUI では label widget の文字列が読み上げられた後、やはり一行のテキスト入力を待つであろう。

Abstract widget によって表されるインタラクションの機能とその実現形態についての例を表 2に示す。

3.5 Interaction Server

Interaction Server はユーザとのインタラクションを引き受ける部分である。どのようなモダリティを用いたインタフェースを実現するか、あるいはそのサーバがどのようなプラットフォームで走るかによって実装方法は異なってくるが、基本的に次の機能を持つ:

- アプリケーションを起動する
- 既に起動されているアプリケーションに接続する
- アプリケーションとユーザ間のコミュニケーションを適当なモダリティにマッピングする

Interaction server は、interaction protocol さえ話せばあとはどのように実現されてもよい。UNIX では一つのプロセスとして走るだろうが、携帯端末では ROM 化して内蔵することも可能であろう。

具体的な interaction server の例は 3.7節に示す。

3.6 代替表現の利用

ビットマップの画像情報などは、GUI 以外のインタフェースではどうしても表しにくい。interaction server でビットマトリックスディスプレイが使える場合はそこに出力することも考えられるが、AUI や CUI では「そこに画像データがある」ということを示すのが精一杯であろう。しかし、画像情報に簡単な説明文章をつけて扱えれば、画像情報が表示できない場合は説明文章を表示するなどの方策がとれる。

そこで、アプリケーションは abstract widget とのインタフェースにおいて、本来扱いたい情報 (原情報と呼ぶ) そのものとその代用表現とをパッケージ化して取り扱うこともできる。これをマルチモーダルデータオブジェクト (MDO) と呼ぶ。MDO を生成するには、原情報に代用表現を付加する (画像データに対しその説明をテキストで付加するといった場合) か、原情報から代用表現を生成するメソッドを付加 (テキストデータに対し、それを音声化するためのメソッドを付加するといった場合) すればよい。

機能	アプリケーションからの提示形式	ユーザからの入力形式
単純なコマンド	[GUI] PushButton [GUI, CUI] メニュー中の 1 項目 [Line, AUI] 必要に応じて available なコマンドをガイド	[GUI] マウスによる選択 [CUI] 特定のキーの入力 [CUI, Line] 特定の綴の入力
状態設定 (Boolean)	[GUI] ToggleButton [CUI] テキストで状態を表示 [Line] 問い合わせに応じて状態を表示 [AUI] 問い合わせに応じてガイド	[GUI] マウスによる選択 [CUI] 特定のキー入力 [Line] 特定の綴の入力
状態設定 (選択)	[GUI] OptionMenu [CUI] メニュー [Line] 問い合わせに応じて状態と available な選択肢を表示 [AUI] 問い合わせに応じて状態と available な選択肢を読み上げ	[GUI] マウスによる選択 [CUI] 特定のキーの入力 [Line] 特定の綴の入力
オブジェクトの選択	[GUI,CUI] available なオブジェクトの表示 (ex. Finder) [Line,AUI] 問い合わせに応じて available なオブジェクトのリストをガイド	[GUI] マウスによる選択 [CUI] 特定のキーの入力 [Line,CUI] 一意な名前等の入力
短いテキストの入力	[GUI,CUI] テキスト入力ウィンドウ (1 行) [CUI,Line] プロンプトの表示 [AUI] 音声によるプロンプト	[GUI,CUI,Line] テキストの入力
複数行テキストの入力	[GUI] テキスト入力ウィンドウ [CUI] テキスト編集ウィンドウ [CUI,Line,AUI] プロンプト	[GUI] テキストの入力, 編集 [CUI] テキスト編集 [Line] ライン編集による入力
2 次元的位置情報の入力	[GUI] 画像表示 [CUI,Line,AUI] (Unavailable)	[GUI] ポインティングデバイス [Line] 座標値の入力
いくつかのオプションを必要とするコマンド	[GUI,CUI] DialogBox [Line,AUI] プロンプト	[GUI,CUI] 個々の項目に入力した後、apply [Line] オプションつきでコマンドを入力

GUI : graphical UI, CUI : character UI, AUI : Auditory UI, Line : line oriented UI

表 2: 主要なインタラクションとその表現

MDO は interaction server によって、適切なモダリティでのデータの表現に変換される。表3に主なマルチモーダルデータオブジェクトをあげる。

原情報	表現		
	GUI	CUI	AUI
plain text	plain text	plain text	文章読み上げ
character	character	character	文字読み上げ
image	image	説明文字列	説明の読み上げ
audio	説明文字列	説明文字列	audio

表 3: 組み込みマルチモーダルデータオブジェクト

3.7 実装

現在、UNIX および X Window System の環境下でサンプル実装を進めている。その概要を紹介する。

Widget

Widget の API は、Tk[5] に似せたものを用いている。Interaction server 側では実際に Tk を用いることで比較的簡単に GUI server を実現できる。

アプリケーション側にリンクされるライブラリ内では、widget は各スロットの値をただ保持しており、生成時および状態変化があったときにメッセージを interaction server に送る。アプリケーション側では geometry management は一切行わない; Tk の pack コマンドについては、pack コマンド発行時のパラメータを保持しておき、interaction server が切り替わったら同じシーケンスでパラメータをサーバに通知することで表示の一貫性を保つ。

Abstract Widget

Abstract widget はオブジェクト指向の Scheme である STklos[2] を用いて書かれている。オブジェクトの定義には、アプリケーションのためのインタフェースの定義と、interaction server がどのようなルールでその abstract widget を表現するかという定義の両方が記述されている。

図5にクラス階層の一部を示す。

Interaction Server

対話的にアプリケーションの起動やコントロールができる GUI server である fresh (Fruit Evaluation Shell) を実装中である。ユーザからアプリケーション起動の要請があると fresh はアプリケーションを fork し、自らと socket で接続する。

モード指定及び端末のケーパビリティによって GUI, CUI, Line UI, AUI(ただし音声合成装置は外付け) に表現を切り替えられる。また、既に立ち上がっている fruit アプリケーションのいずれかに接続することもできる。

問題点

現在のところ、abstract widget の表現ルールは interaction server 側にハードコードされているため、アプリケーション側で新たな abstract widget を定義したい場合は Interaction server も再構築しなければならないという問題点がある。表現ルールを記述する簡易言語を設計し、実行時にアプリケーション側から interaction server 側にルールを転送してやることができればこの問題は解決すると思われる。

4 関連研究

ユーザの様々な環境からくる要請に、ツールキットレベルで対応している研究例はいくつかある。Mynatt ら[4] は X Windows アプリケーションにおいて、プロトコルレベルでアプリケーションからの情報を抽出することにより、従来のアプリケーションを変更することなく GUI を聴覚情報にマップすることに成功している。また、Blattner, Glinert ら[1, 3] はツールキットに Metawidget という概念を導入することにより時間的に変化する特性を widget に組み込み、視覚表現／音響表現を統一的に扱う事を試みている。

既存のアプリケーションがそのまま使えることは大きな利点である。しかしいずれも、GUI の構成要素をいかにして他のモダリティにマップするかに重点がおかれている。GUI の構成要素の分類はアプリケーションのセマンティクスと必ずしも 1 対 1 に対応しないため、GUI 以外のモダリティではインタラクションが不自然になる場合もあると考えられる。

fruit を用いる場合、既存のアプリケーションは使えないか、少なくとも大幅な書き換えが必要である (現在の実装であれば、既存のアプリケーションが Tk を用いていた場合には書き換えは比較的容易であるが)。ただし、一度作ってしまえば interaction server の入れ換えだけで様々なモダリティに対応できるという利点がある。また、実行時に interaction server だけをリモートの計算機で立ち上げて切り替えることにより、アプリケーションを中断することなくユーザが望む場所に移動することができる。

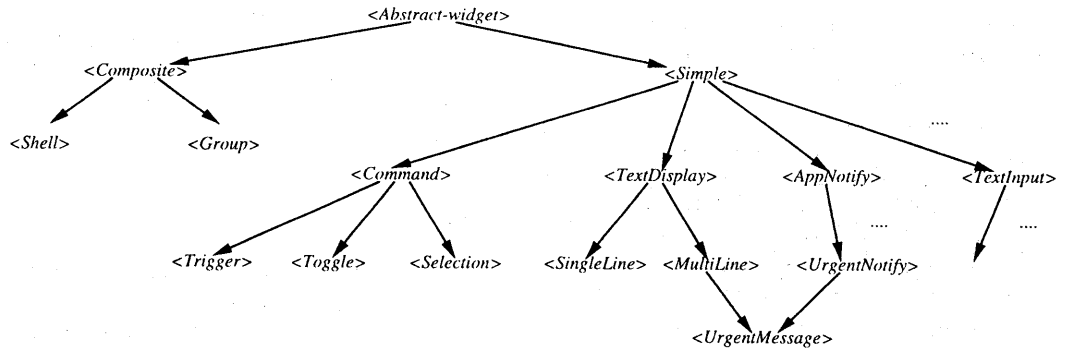

図 5: Abstract Widget のクラス階層

5 おわりに

計算機のインタフェースは様々なモダリティを扱えるようになったが、ユーザの環境によってはそれらすべてが使えるわけではなく、またユーザがアプリケーションを使用中にその環境が変わる場合もある。

本稿ではそのようなインタラクションの形態の変化にダイナミックに対応できる、汎用的なツールキットを紹介した。

単体での分かりやすさ、使いやすさを追求したアプリケーションが次々に開発され、またユーザインタフェースの技術も急速に進歩している。それ自体は歓迎すべきことであるが、ともすれば標準的でないインタフェースを要求するユーザは見過ごされがちである。せっかくインタフェースの手段が広がったのであるから、これまで計算機とインタラクトすることが難しい環境にいたユーザにも容易に使えるようなアプリケーションが数多く生産されることを期待している。

参考文献

[1] M. M. Blattner, E. P. Glinert, J. A. Jorge, and G. R. Ormsby. Metawidgets: Towards a Theory of Multimodal Interface Design. In *Proc. COMPSAC '92*, pages 115–120. IEEE Computer Society Press, Sept. 1992.

[2] E. Gallesio. STklos : A Scheme Object Oriented System Dealing with the Tk toolkit. In *Proceedings of X'hibision 94 conference*, 1994.

[3] E. P. Glinert, R. L. Kline, G. R. Ormsby, and G. B. Wise. UnWindows: Bringing Multimedia Computing to Users with Disabilities . In *Proceedings of the IISF/ACMJ International Symposium on Computers as Our Better Partners*, pages 34–42. World Scientific, Mar. 1994.

[4] E. D. Mynatt and W. K. Edwards. Mapping GUIs to Auditory Interfaces. In *Proceedings of the 5th Annual Symposium on User Interface Software and Technology (UIST '92)*, pages 61–70, Nov. 1992.

[5] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[6] K. Sakamura. TRON Enableware: Empowering Dsiabled People on the BTRON Environment. In *Proceedings of the IISF/ACMJ International Symposium on Computers as Our Better Partners*, pages 26–33. World Scientific, Mar. 1994.

[7] G. C. Vanderheiden. Building Disability Access Directly into Next-Generation Information and Transaction Systems. In *Proceedings of the IISF/ACMJ International Symposium on Computers as Our Better Partners*, pages 2–6. World Scientific, Mar. 1994.

[8] 小山 智史. スキャン方式による肢体不自由者用キーボードエミュレーションプログラムの開発. 信学技報, HC93-31:49–55, Sept. 1993.

[9] 小山 智史, 野島 秀夫, and 太田 茂. 盲人のためのコンピュータインタフェース —OS-TALK—. 電子情報通信学会創立 *70* 周年記念総合全国大会, S13-11, 1987.

[10] 太田 茂. 電子機器アクセシビリティ実現への世界的取り組み. 電子情報通信学会誌, 72(10):1169–1176, Oct. 1989.

[11] 太田 茂. 障害者の可能性を広げるコンピュータ. 中央法規出版, 1990.