Linuxプラットフォームにおける視覚障害者のアクセシビリティ、支援技術の動向——なぜ、点字ベースのユーザインターフェイスは欠かせないのか

南谷和範
学習院大学、Debian-jp プロジェクト、筑波大学付属盲学校非常勤講師
(minatani@debian.or.jp)

目 次

1 始めに

1.1 本稿の目的と構成

本稿の目的は Linux プラットフォームにおける視覚障害者のアクセシビリティ、支援技術の最近の動向を概観し、合わせてオープンソースにおけるこれら技術の意味付けを探る一助とすることにある。その際、特に注目すべきものとして、Linux プラットフォームにおけるアクセシビリティ

の根幹を担うスクリーンリーダ BRLTTY の展開、開発の方向性を解説する。合わせて GNOME GUI プラットフォームにおける支援技術の現状を補論として取り上げる。とはいうものの、視覚障害者のアクセシビリティは一方で重要な問題として認識されてはきたが、具体的な事項に立ち入った一般的な解説は、特に Unix 関連の議論の場では殆どなされてこなかったように見うけられる。そのため、本稿では Linux プラットフォームあるいは計算機からも離れるようなトピックであっても、トータルな説明を行う上で不可欠と思われる論点についてはいくらか立ち入った解説を行った。具体的には点字ピンディスプレイのメカニズム、点字の文字体系や各言語・分野におけるヴァリアントについて、かなり説明を行った。視覚障害者のアクセシビリティを論ずる際、音声を用いたインターフェイスについて語られることが多い。[1] なるほど、音声を用いたインターフェイスは基盤ソフトウェアがそろってきたこともあり、安価に実現できる有効な手段である。しかし、それは決して点字ピンディスプレイを用いた文字ベースのインターフェイスが役割をおえたことを意味しない。むしろ、ある意味では点字ピンディスプレイを用いた文字ベースのインターフェイスは原理的に音声出力インターフェイスに優越する。理由は簡単である。情報の提示の主導権が点字ピンディスプレイを用いた文字ベースのインターフェイスではユーザの側にある。音声出力では主導権はあくまで計算機の側にある。いくら音声出力のタイミングや速度、発声の工夫を行ったところで、ユーザ (視覚障害者) の側が半無意識的に自由に読み方を選択できる文字ベースのインターフェイスに比べれば、情報理解に際しての拘束は厳しい。こうした不利はリニアに文章を読み進むことができないような局面において顕著になる。なるほど、Emacspeak はこうした不利を軽減するための工夫を数多く取り込んでいるし、単純な文章を短時間で把握するという用途であれば、音声出力によるインターフェイスの方が有利である。とはいうものの上に示したユーザ側の主導権の有無という根本的な問題は解決しがたいものとして残る。この事実のみをもって、本稿のサブタイトルに示した問い—-「なぜ、点字ベースのユーザインターフェイスは欠かせないのか」—-への解答と言ってしまってもよいほどである。

ところが、これまで点字の特質やその計算機との親和性まで踏み込んで、点字ピンディスプレイを用いたインターフェイスを扱ったトータルな論述はなされることがほとんどなかった。この主題を本格的に論ずるには、点字の表記法を十分理解した上で、それを計算機上の文字情報を表現するものとして用いる場合の特性を把握しておく必要がある。さらに日本語を取り扱うことを考慮するのであれば、後述する日本語点字体系の特殊な位置という論点が加わる。点字に対する包括的な知識を要求されるこの問題は、これまで正面から論じられることがなかった。非力な筆者ではあるが、BRLTTY という優秀なソフトウェアの紹介を通じて、点字ピンディスプレイを用いた文字ベースのインターフェイスの有効性を論じてみたい。この先行する論述の欠如という状況にかんがみ、表題からいくらか脱線する筆者の議論も最初の車輪の発明だと思ってご寛恕いただけるとありがたい。また、(批判的見解も含め) 今後の諸氏の論考のリソースとなれば幸いである。

本稿の構成は以下のとおりである。本論にあたる 2 章では、Linux プラットフォームにおけるアクセシビリティの根幹を担う BRLTTY の最近の展開、開発の方向性を解説する。その過程で BRLTTY の機能に即しつつ、点字表記法の重要なポイントを解説する。点字表記法の体系的な詳述は本カンファレンスの趣旨から完全に逸脱するものであり、今回は差し控える。

次に目下の問題である GUI プラットフォームの支援技術として Gnopernicus を中心とする GNOME GUI プラットフォームの一連の開発動向を 3 章において補論として略述する。

---

[1] Linux プラットフォームにおける音声を用いたアクセシビリティの試みとしては、emacspeak がまず第 1 にあげられるべきであろう。また、emacspeak を含む音声インターフェイスについての論文としては渡辺、井上両氏による BEP(Bilingual Emacspeak Plathome) についての解説が包括的な説明を提供している。両氏を中心に発表されたペーパーは複数あるが、`http://lc.linux.or.jp/lc2002/papers/watanabe0920h.pdf` がもっとも包括的な資料である。本稿執筆にあたっても、両氏を中心とする BEP 関係者からご助言をいただいた。

BRLTTY, Gnopernicus いずれについても多言語対応、特に日本語プラットフォームへの対応可能性について考察する。

1.2 用語についての若干のコメント

アクセシビリティ、支援技術という言葉は、近年、障害者の社会参加を特にテクノロジーとの関連で問題にされる場合に用いられる言葉であり、若干の解説を加えておきたい。(用語の問題にあまり興味のない読者はこの項は読み飛ばしていただいても差し支えない。)

最初に結論を述べるのであれば、筆者の見る限りこれらの言葉について確固とした定義はいまだ定まっていない。むしろ、その都度文脈に従って定義され、使われている印象が強い。また、障害者の積極的な活動をサポートするという (少なくとも表向きは) 批判の余地のない目標と関連付けられるアクセシビリティ、支援技術といった言葉は、分析概念であるとともに、目的をもった活動のスローガンとしても多用され、そのことが意味の焦点をぼやかすことにもなっているように思われる。

アクセシビリティという言葉については、それが基本的にはアクセシブル、アクセス可能であるという語の派生であることが重要である。ある一定の障害を有する人間がアクセスできる、利用可能である特質・状態をアクセシビリティと呼ぶことができよう。無論、この概念の適用は計算機プラットフォームに限られない。(たとえば公共建築物の利用可能性。) 他方、このアクセス可能性という発想は「障害者に最も適した計算機プラットフォーム (あるいは建築物の設計)」という発想とは一線を画すものであることにも注意が必要である。つまり、アクセシビリティという概念は、あくまでアクセス可能であることを目標とし、その達成で満足するものであって、ある意味では次善のソリューションとも言える。この論点が特に計算機プラットフォームの文脈で重要なのは、アクセシビリティとユーザビリティは本質的に違う概念であるという点である。つまり、ユーザビリティという概念をユーザの使いやすさをストレートに追求するものと見るのであれば、アクセス可能であることで一応満足するアクセシビリティの発想とは基本的に異なることとなる。こうした相違の実例は後に GNOME GUI についての章で具体的に言及することとなる。

支援技術 (assistive technology) とは障害者がなにかを積極的に行う場合にその助力となるテクノロジーを指す語と言える。大方の場合 (ハード、ソフト両者を含む) 計算機プラットフォームのプロダクトあるいはある程度高度なテクノロジー (もう少しありていに言えばローテクでない技術) を用いた器具を示す。

なお、しばしば用いられる障害者の「社会参加」という言葉についても何らかの解説を加えておくべきかもしれないが、アクセシビリティ・支援技術と比較しても圧倒的に慣用的に用いられるいわば玉虫色の言葉なので踏み込んだ説明は控える。あえて言えば、障害者が他人と能動的に関わり合うこと、あるいは創造的な行為に携わることをイメージしていると思われる。

2 BRLTTY の開発動向

BRLTTY は Linux プラットフォームで確固たる地位を確保した点じピンディスプレイ ((refreshable) braille display) スクリーンリーダである。オフィシャルウェブサイトは`http://mielke.cc/brltty/`。現在のメンテナは Dave Mielke である。開発スタイルとしては、Mielke を中心とする開発者グループがパッチを受け付け、それを適宜マージしていく方式を採用している。本稿では現行執筆時の最新版である 3.5pre2 の実装を基準に記述する。

ファーストリリースは 1995 年夏に ftp://ftp.sunsite.unc.edu にて行われた。

2.1 点字ピンディスプレイと点字の概略

BRLTTY の解説の前に、点字ピンディスプレイの構造と点字表記法の簡単な説明をしておく。点字ピンディスプレイは、ピンを上下に駆動させることによって点字を表示するディスプレイである。[2] 点字は原則縦 3 点横 2 点の合計 6 点で構成される「ます」(セル) を単位にして文字を表現する。1 セルは六つの点の有無で表現されることから 2 の 6 乗で 64 パターンのキャラクタが表現可能である。計算機プラットフォームで利用される英語点字の規格 NABCC(North American Braille Computer Code) では、6 点で構成される基本部に加え、セルを縦 4 点横 2 点に拡張し、8bit256 パターンの表現を可能にしている。このベースとなる 6 点部の NABCC のパターン一覧は http://www.kgs-jpn.co.jp/dev/NABCC.html で閲覧できる。(8 点に拡張する場合アルファベット大文字は追加された 2 点のうちの一つ 7 の点を付記することで小文字との区別がなされる。) 8 点に拡張された NABCC コードは BRLTTY に含まれる BrailleTables/text.NABCC.tbl を Programs/tbl2hex でダンプすることで 16 進ダンプの形ではあるが一応確認できる。BRLTTY 3.4.1 までは tbl2txt というプログラム (のソース) が付属しており、これを用いて tbl を可読性の高いテキスト形式にダンプできたが、現在 3.5pre2 にはこのユーティリティは含まれない。ただし、旧ヴァージョンの tbl2txt で 3.5pre2 に含まれる text.NABCC.tbl(text.*.tbl) をダンプすることはできる。

8 点への拡張により、コントロールコードまでを含む ASCII コードは全て NABCC で表現可能となる。NABCC を踏まえ、点字ピンディスプレイの大半は縦 4 横 2 にピンを配置したセルをユニットとしてそれを複数個組み合わせてディスプレイを実現している。しかしながら、このセルユニットは駆動系を多用する精密機械であり、また需要がさほど大きい製品ではないので、高価である。点字ピンディスプレイヴェンダに供給される際のセルユニット単体の価格は明らかではないが、点字ピンディスプレイとして商品化された際の一個あたりの単価は 10,000 円をくだらない。また、アームを用いてピンを上下させるという構造上、縦列の集積は難しい。[3] こうした事情が重なる結果、たとえば 640*480 ドットの VGA 相当、つまり 80 桁 25 行の点字ピンディスプレイが製品化されたというようなことはない。コストと利便性のトレードオフとして、40 桁ないし 80 桁一行分を表示する点字ピンディスプレイが一般的である。(一部に 2 行を同時に表示できるものもある。)

点字ピンディスプレイは普通シリアル (あるいは USB) 経由で計算機から制御される。

2.2 BRLTTY のメカニズム

BRLTTY は、上に示したような構造になっている点字ピンディスプレイを Linux から制御し、画面 (モニタ) への出力内容を随時点字ピンディスプレイに表示させるソフトウェア (デーモンプロセス) である。動作の流れとしては、/dev/vcsa から現在の画面の表示内容を取得し、画面上の ASCII キャラクタに対応する点字を表示するためのコードをシリアル経由で点字ピンディスプレイに送信する。/dev/vcsa の機能、出力のフォーマットについては vcsa(4) および vcs(4) のマニュアルペー

[2] 本稿では、英語の (refreshable) braille display に対応する日本語として「点字ピンディスプレイ」を用いる。英語で refreshable が付されるのには、braille display(点字ディスプレイ) では常時書き換えが可能であるという特質が伝わりにくいためである。意訳ではあるが、日本語でもピンの駆動という機構を反映し、慣用されてきた「点字ピンディスプレイ」の語を用いる。

[3] 日本の KGS 社はセルユニットの研究開発において、世界をリードする企業であり、大手点字ピンディスプレイメーカーにセルユニットを供給している。同社のセルユニット製品の概要は http://www.kgs-jpn.co.jp/b_sc.html にて閲覧できる。縦列集積の困難は、この仕様から明らかであろう。

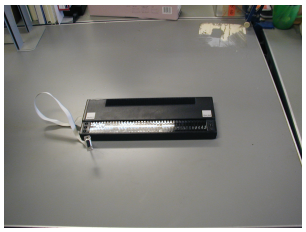

図 1: 筆者所有の BAUM 社の Vario40 点字ピンディスプレイ

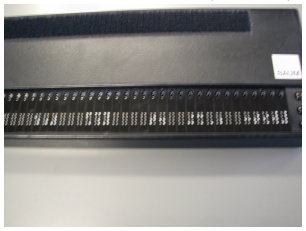

図 2: 点字ピンの部分にフォーカスしてみた。全体的にぼやけてしまったが、浮きあがって点字を表現しているピンが分かるだろうか。

ジを参照していただきたい。/dev/vcsa を情報源として用いることから分かるように BRLTTY が点字ピンディスプレイに送信する画面情報は、カーネルが直接管理するいわゆるコンソールへの文字出力に限られる。同じ文字情報であっても X Window System 上で起動された xterm などの出力には感知しない。

一方、GNU screen にパッチを当て、その共有メモリから画面出力を読み出し、点字ピンディスプレイに表示することもできる。/dev/vcsa は Linux 固有の機構であり、/dev/vcsa が利用できない他の UNIX ではこの Gnu screen との組み合わせというアプローチで BRLTTY を使うことができる。現在 Solaris と OpenBSD が正式にサポートされている。(ドキュメントには言及が見られないが HP-UX をサポートするコードも含まれている。) ただし、ユーザプロセスである Gnu screen に依存するため、ブートアップ時のメッセージなどは閲覧できない。また、画面表示を確認せずにログインして GNU screen を起動する作業を行わなくてはならない。これは看過しがたいデメリットである。

基本動作としてはこの

- 画面表示内容の取得
- 点字ピンディスプレイへの情報の送出

という作業をリアルタイムで繰り返すことになる。(点字ピンディスプレイのリフレッシュレートの限界から現実的には一秒間に数回の書き換えとなる。)

ただし、すでに述べたようにピンディスプレイが一度に表示できるキャラクタはせいぜい 80 が限界であり、計算機の画面上の文字情報を一度に全て表示することはできない。そこで、モニタ上のフォーカス領域を動かし、点字ピンディスプレイが点字で表示する領域をシフトさせることが必須となる。こうした要求に応えるために点字ピンディスプレイにはカーソルキーに模した一群のキーが搭載されている。ユーザ (視覚障害者) はこのキー群を操作し、画面のフォーカス領域[4] を移動させ、点字で表示させる領域を変更する。これらのキーが点字ピンディスプレイの側にあること

[4] フォーカス領域というのは、あくまで観念であり、BRLTTY が画面描画を操作してフォーカス領域を画面上に示しているわけではない。

は、点字を指で読むことを勘案するならヒューマン・インターフェイス的にも都合がよい。このオペレーションを実現するために BRLTTY は点字ピンディスプレイがシリアル経由で送信してくるキーコードを監視し、それぞれのキーに対応するフォーカス領域のシフトを行う。

BRLTTY の動作は上に示した二通りの作業に概括できる。BRLTTY が大きな信頼を勝ち得たのは、ファーストリリースの時点で実用的には十分な完成度を達成していたことが大きく寄与している。列挙するのであれば

1. それまで MS-DOS プラットフォームで集積されてきたプロプライエタリなスクリーンリーダのノウハウを忠実に継承し、ユーザ (視覚障害者) に UI への順応という形で強いるハードルが極めて低かった。他にも痒いところに手が届いたオリジナルな UI の工夫が見られた。たとえば、点字表示位置のフォーカスが最上行あるいは最下行にある時に、より上の行あるいは下の行にフォーカスを移動させようとするとビープで警告するようになっている。

2. 提供したピンディスプレイのドライバ部が高度な安定性を実現していた。これは突如の点字ピンディスプレイへの出力停止・操作不能というようなユーザに巨大なフラストレーションを与える事態を回避することになった。

3. /dev/vcsa からの単純な読み出しとかれたシリアルでの通信という設計により、ハードウェア要求が極めて低くすんだ。筆者が確認した範囲では CPU Intel 486 25MHz、メモリ 8MB のプラットフォームでも BRLTTY 導入による負荷の上昇は観察されず、BRLTTY のレスポンスにも顕著な低下は観察されなかった。

という点があげられよう。2,3 については中心的開発者であった Nicolas Pitre が Linux kernel を ARM プロセッサにポーティングした人物であり、[5] 想定される問題にセンシティブであったことが大きなアドバンテージとして働いたと思われる。

2.3 点字ピンディスプレイドライバの増大とドライバのモジュール化

計算機と点字ピンディスプレイの間の通信プロトコルには、点字ピンディスプレイメーカー各社の間で互換性は原則なく、ドライバで個別対応が必要である。そのため、ファーストリリース後の BRLTTY の開発は対応する点字ピンディスプレイの拡大を中心に進められた。この作業は基本的に未対応の点字ピンディスプレイの所有者、利用者が個人で書いたドライバコードをコントリビュートする形態で行われている。現在 18 種類のプロトコルがサポートされている。BAUM 社の VARIO シリーズは複数のプロトコルをサポートしており、実質的には 17 種の製品群をサポートしていることとなる。これら 17 種にはそれぞれ搭載するセルの数や機能でかなりの数のヴァリアントが存在し、実際にサポートしているハードウェアの種類は 50 前後と想像される。

イニシャルリリースに含まれていたドライバ部のコードは非常に見通しのよい設計になっており、後のドライバ開発は概してこれを雛型にしている。

一方、対応する点字ピンディスプレイのドライバの増大とともに、それらを包括的に扱う枠組みが必要となってきた。元来、BRLTTY はコンパイル時に Makefile でコンパイルする点字ピンディスプレイのドライバを指定し、そのコードを含むワンバイナリとして BRLTTY をコンパイルするようになっていた。このアプローチは鉄のような安定性を確保するという観点からは有効ではあったが、汎用性という点では大きな制約となる。特にユーザ (視覚障害者) の間には独力でディストリビューションをインストールできる環境を確保したいという要求が根強くある。一つのバイナリ

[5] 少し古い情報ではあるが `http://www.cam.org/ nico/cv.en.html` 参照。

では一つの点字ピンディスプレイにしか対応できないという状態ではそれぞれのピンディスプレイ用のインストーライメージを個別に準備しなくてはならない。結局のところ筆者を含めユーザ (視覚障害者) が個別に一般に配布されているインストーライメージを改造し、自分の所有する点字ピンディスプレイに対応した BRLTTY バイナリを含むインストーライメージを作成するという状態がしばらく続いた。

こうした状況を踏まえ、version 2.3 へのバージョンアップの際に点字ピンディスプレイのドライバを共有ライブラリとしてロードするスタイルに変更された。さらに、USB 接続の点字ピンディスプレイの増大にかんがみ、シリアルポーとと USB ポートを統一的に扱う設計がヴァージョン 3.4 以後導入されている。この結果、一つのバイナリセットで複数の点字ピンディスプレイへの対応が可能となった。ドライバモジュールは BRLTTY の起動時に-b (ドライバ名)、あるいは–braille-driver=(ドライバ名) という形で指定する。一方、ブート時の kernel parameter としてドライバ名を指定し、カーネル起動後、BRLTTY がその値を参照してロードするドライバモジュールを決定することもできる。これは上で触れたディストリビューションのインストーラへの要求が反映された機能である。ただし、安定性への配慮から、特定のドライバを選択し、それをスタティックリンクしたワンバイナリとしてビルドすることも引き続き可能である。

ドライバの一括共有ライブラリ化は BRLTTY 内でのファンクショナリティの切り分けの明確化でもあった。もともと、こうした切り分けはワンバイナリの時代からはっきり意識されていたが、ドライバのモジュール化で名実ともに達成された。つまりドライバモジュールはハードウェアとの通信、やり取りを担当し、本体部が/dev/vcsa からの読み出し、と共通化された方式でのモジュールとのやり取りを行う。本体部は一般性の高いものとして独立し、音声出力モジュールへの対応にもつながった。

2.4 ヨーロッパ諸語の点字への対応

同時にヨーロッパ諸語の点字への対応も順次行われた。周知のとおり、ヨーロッパ諸語には英語アルファベットにアクセント記号を付した文字が多数存在し、それは計算機用の文字コードにも反映されている。元来、各地域の点字は、その地域の視覚障害者が自己の必要に従ってその地域の言語の文字に合わせた点字を発案し、採用されてきたという経緯がある。英語アルファベットや数字というヨーロッパ諸語で押しなべて採用されている文字については、おおよそ点字でも共通のパターンが採用されているが、各言語特有の文字については各々独自に制定されてきた。[6] 一方、上述のように点字では原則 1 セル 64 パターンしか表現できず、ヨーロッパ諸語の間でも一つのパターンに言語によってはそれぞれ別の文字が割り当てられるという事態が生まれた。一部の記号類についてはある言語では点字パターンが定義されているが、他の言語では定義されていないという状況も見うけられる。他方、近年の計算機の重要度の増大にかんがみ、これらの国々でも計算機での利用を考慮した (縦 4 点、横 2 点の 8 点に拡張された) 計算機用点字パターンが独自に考案され、随時採用されている。

1 セル 64 パターンという制限は、こうした言語ごとの非互換性の問題に留まらない。数学、物理、計算機などで使われる記号類を総合すれば、文字数が比較的少ない英語であっても、64 パターンに収まりきらないのは火を見るより明らかである。実際、こうした事態は深刻で、現実に英語数学点字、英語計算機点字 (前述の NABCC もその一つ)、英語物理点字というように分野ごとに点

[6] 本文の記述では英語の点字が史上初めて考案されたという誤解を招きかねないので補足しておくが、点字は 1824 年にルイ・ブライユによってフランス語を表現するものとして考案された。その際ブライユはフランス語固有のアクセント記号付きアルファベットをベースとなるアルファベットの点字パターンに規則的な変化を加えたものとして表現した。こうした経緯を考慮するのであれば、英語点字はこのフランス語点字からフランス語固有の文字を削除したものと考えるべきである。

字パターンの体系が独立に存在している。英語数学点字と英語計算機点字の間では −, +, = といった基本的な記号にすら統一は図られていない。よって視覚障害者はそれぞれの点字パターンの体系を学ぶ必要があり、非効率である。そもそも、2 次元平面にいわば自由に作図することでいくらでも文字を作ることのできる普通文字をパターンに厳しい制約のある点字で網羅的に表現するというコンセプトには限界があろう。

この深刻な状況にかんがみ、International Council on English Braille において UEBC(Universal English Braille Code) の策定が行われてきた。これは、英語について、数学、物理、化学、計算機の諸分野の記号を統一しようという試みである。これらの記号を総計すれば、当然 64 パターンという制限をあふれることになる。そこで UEBC では 1 文字を複数のセルを使って表現する方法を採用している。[7] 厳密にはこうしたアプローチはすでに在来の点字で部分的に採用されていたのだが、UEBC ではこうした表現法を包括的に採用している。無論、一文字のパターンとしての表現は冗長となる。発想としては 8bit の ASCII から 32bit の Unicode への転換と一脈通じるところがあると言えよう。ただし、計算機内部の文字コードである Unicode の冗長性は計算機資源の充実 (CPU パワーと記憶容量の飛躍的増大) でカバーできるが、人間 (視覚障害者) が直接読む点字の場合、この冗長性に耐えられるかは未知数といわざるを得ない。ありていに言えば、複数の点字パターンの体系を学び、使い分ける負荷と一つの体系に統一された冗長な点字を読む負荷の選択を迫られているといえる。とはいうものの、これまで放置されてきた点字体系の方言化に疑義を呈したという点で UEBC の策定は重要な一歩であろう。事実、本年 3 月末にカナダで開催された ICEB 総会において、UEBC 制定に基本的なゴーサインが出された。筆者は現在試みに UEBC の計算機点字パターンを出力する BRLTTY 用テーブルを作成している。UEBC についての詳細は `http://www.iceb.org/ubc.html` を参照。

こうした事態に対応すべく BRLTTY では任意の 8bit 文字コードと点字パターンの間の変換テーブルをロードする仕組みが実装されている。これはユーザが自由に定義可能なもので、個人で独自のテーブルをこしらえることもできる。ユーザからのコントリビューションも盛んで、ロシア語なども含め、一バイト文字コードの言語は相当程度サポートされている。

2.5 英語点字略字への対応

次の段階の作業としてヴァージョン 3 へのメジャーヴァージョンアップに際し、英語点字略字のサポートが行われた。英語点字には、点字触読の速度の不利を緩和するため、英語文章中で頻出する文字列を短縮して表記することができる。[8] たとえば、every は e、like は l と記述することが認められている。

略字の規則はかなり複雑なものであるたとえば、gg は略字として括弧マークに当たる点字パターンを用いる。一般に括弧は単語の前 (後) に出現するものであり、gg の略字としてのコーテーションマーク (の点字パターン) は単語の途中にのみ使用が許されている。よって、egg は略字が使えないが eggs は略字を用いて記述できる。さらに、複雑なことに英語点字では開き括弧と閉じ括弧に同じシンボルを割り当てている。単語の先頭にこのシンボルが現れた場合 (として、単語の後に現れた場合) として、単語の途中に現れた場合 gg として読み下すことが読者には要求される。

ただし、この例からも明白なように英語略字は英文の慣習を想定して策定されているので、計算機文書との相性は極めて悪い。にもかかわらず、あえて BRLTTY が英語略字のサポートに労力を

[7] UEBC についての詳細は、International Council on English Braille, “Unified English Braille Code Research Project —— The Reader Rules —— The January 2004 Report of the Objective II Committee” を参照。同レポートでは、英語点字における在来の複数表記体系の併存の問題も的確に論じられている。

[8] 前章で触れた UEBC の冗長性は、英語略字に象徴される触読の速度的不利への配慮には逆行するものである。

割いたのは、むしろユーザ (視覚障害者) の点字表記にたいする思い入れの強さを示しているものといえよう。

BRLTTY では一種のパターン記述言語を導入し、略字の規則を自由に記述し、オリジナルなテーブルを出力できる仕組みを準備している。これは、英語略字にとどまらない大きな可能性を秘めたメカニズムである。事実、このメカニズムを応用して、台湾からのコントリビューションにより BIG5 に対応する点字パターンのサポートも行われている。ただし、筆者は寡聞にして台湾で用いられている点字の現状を知らないため、この変換テーブルがどの程度包括的なものかは分からない。

2.6 BrlAPI 登場

さて、ここまでみてきた柔軟性の拡大、内部構造の抽象化の応用として、BRLTTY の資産を他のプログラムの点字出力を担当するものとして活用する試みがなされている。BRLTTY は上に述べたように随時対応する点字ピンディスプレイの種類を増やしてきており、現在では市場に流通している点字ピンディスプレイの相当数をサポートするにいたっている。おそらく、Windows 用のプロプライエタリなスクリーンリーダでもこれほど多数の製品をサポートしているものは稀であろう。Windows 用のスクリーンリーダにおいて点字ピンディスプレイのサポートが比較的軽視されているのには、

- ソフトウェア音声合成の技術が確立し、ユーザの興味が高価な点字ピンディスプレイについては相対的に減退したこと
- アイコンを多用する Windows GUI では、文字情報をそのまま表現する点字ピンディスプレイの長所がいかせないこと
- スクリーンリーダのベンダはしばしば点字ピンディスプレイのベンダでもあり、自社製品以外の点字ピンディスプレイのサポートには消極的であること

などの理由が考えられる。

一方、BRLTTY はあくまでコンソール出力の内容を表現することに特化しており、X Window System 等の GUI への対応は考慮してこなかった。このスタンスはユーザ (視覚障害者) に徹底的に安定した画面出力を保証するという観点からは極めて正しかった。とはいうもののユーザ (視覚障害者) の間に GUI プラットフォームへの継続的な興味が存在したことも事実である。事実、X Window System をアクセシブルにする試みは過去にいくつか存在した。Solaris 上の X Window System スクリーンリーダとして開発がアナウンスされた UltraSonic(現在 Web 上でこのソフトウェアについての記述は発見できなかった。) や XVIL `http://portal.beam.ltd.uk/xvil/index.html` が有名なプロジェクトである。しかし、これらはいずれもまとまった成果をあげることなく、終焉したといえる。これらの挫折には X Window System の現代的水準からすれば洗練されているとはいえないアーキテクチャに起因する問題があったと想像される。他方、後に述べる Gnopernicus は実際に視覚障害者が GNOME GUI プラットフォームを利用しうる可能性を具体的な形で提示した。こうした事情にかんがみ、BRLTTY ではその点字ピンディスプレイとの通信を担当する部分を切りだし、他のスクリーンリーダが統一的な API で各種点字ピンディスプレイを扱えるライブラリ libBrlAPI の提供を開始している。

2.7 BRLTTY の日本語への対応

最後に BRLTTY の日本語への対応の可能性について若干触れておきたい。日本語への対応には大きく分けて二つの問題点がある。

第一は BRLTTY が画面出力の読み出しに用いている/dev/vcsa がマルチバイト文字コードに対応していないことである。そのため/dev/vcsa から画面情報を読み出すアプローチを踏襲するなら、まずは kernel の変更が必要となる。ただし、画面上のキャラクタをべたでダンプすることをコンセプトに設計された/dev/vcs, /dev/vcsa の設計思想からはある程度逸脱するともいえ、新たなデバイスファイルの創設が必要となるかもしれない。一方、前注に述べた GNU screen へのパッチを利用する手法もありうる。GNU screen はマルチバイト文字を想定して設計されており、その共有メモリに保持される画面データにもマルチバイト文字を含む情報が適切に反映される。kernel に手を入れるよりは GNU screen からデータを読み出す方がハードルは低い。ただし、BRLTTY の重大な長所であるブートアップの早い段階から画面出力をモニタリングできるという特質は諦めざるをえない。

次に日本語点字を計算機で表現する上での問題がある。実のところ日本語点字には原則漢字がなく、日本語文章は全てかな書きにし、文節毎にスペースを入れて区切ることになっている。(日本語点字の記法としてオフィシャルなものは日本点字委員会編、『日本点字表記法 2001 年版』である。ただし、同表記法の細目については難色を示す人も少なくない。)

蛇足かも知れないが、点字における日本語情報処理を考える上で、避けて通れない問題でもあり、日本語漢字文章と点字の親和性の問題について若干触れておく。日本語点字において漢字を導入しようという試みはいくつか存在し、当事者の活動は活発であるが、試案の域を抜けていない。だが、すでに述べてきたように、一セル 64 パターンしか表現できない点字で数千のオーダーを下らない常用漢字を表現することには根本的な難があろう。実は、そもそも 50 音と数字、句読点といった基本的な記号だけでも 64 パターンはあふれ、2 セルをまとめて用いて 1 文字を表すことは、すでに現在の日本語点字でも多用されている。だが、シンボル、図柄としての含蓄を有する漢字を点の並びである点字に対応付けることには大きな障害があろう。さしずめ JIS コードを直接人間が読み書きする、あるいは T-Code で書き、T-Code のシーケンスを見て文字を判読するようなものといえる。もちろん、これまで提案されてきた日本語点字漢字案もこうした問題には意識的で

- 一文字あたり 3 セルを用いて漢字の音訓読みを反映させる 6 点漢字
- 一セルを縦 4 横 2 の 8 点に拡張し、複数セルを用いて偏・旁と言った構成要素の表現を試みる漢点字

が提案されている。(少しバランスを失している印象もあるが) この二つの漢字案についての紹介を含む日本語点字の簡単な説明は `http://www.yoihari.com/tenji/index.htm` で閲覧できる。

これは 1890 年に日本語点字が制定された際、すでに確立されていたヨーロッパの点字とその筆記具をそのまま活用したことに起因する。あえて例えるなら、1 バイト ASCII 文字の枠組みに準拠した半角かなのみで日本語の記述を行うこととし、欧米の計算機をそのまま輸入し（漢字 ROM やそれに相当するソフトウェアの追加無しに）日本語の記述を可能にしたようなものとも言える。こうした記述から読者はこの決定に強引という印象を受けるかもしれないが、上に示したように、点字で漢字を表現することには根本的な困難があり、かな分かち書きという制定時の決断を誤りとすることも一方的であろう。

また、文章の音読時の音を基本に記述することとなっており、一般のかな書きでは「う」となるところが長音符を用いる場合が多々ある。こうした特殊な記方を計算機でサポートするには、まず

漢字かな混じり文をかな分かち書き文に変換するルーチンが必要となる。この作業においては無論大規模な変換辞書の実装が必須である。さらに、このルーチンにおいては音読読みに準じた点字の表記法にも対応する必要がある。これらの条件を満たすフリーの実装は現在存在しない。なお、決して完全なものではないが、
chasen -F "%a "
によって、この点字の表記法に類する出力を得ることができる。プロプライエタリなものではEXTRA for Unixが漢字かな混じり分から日本語点字への変換をサポートしている。http://www.extra.co.jp/product_2.html#A 参照。なお、容易に想像できるように、どのようなソフトウェアを用いるにしろ、100%の変換精度を保証することはできない。

また、一文字一セルを厳密に維持することのできる英語コンピュータ点字に対して、日本語点字ではこうした一対一の対応関係はまったく維持できない。たとえば、「南」という漢字をかなで読むのであれば、文脈によって「みなみ (音読み、3 文字)、「なん (訓読み、2 文字)」と文字数が変わり、表現に必要なセルの数も変わる。表やインデントされたソースリストを閲覧することを考えれば、一文字一セルを厳守できるか否かが決定的な意味をもつことが分かる。

以上、いくらか悲観的な見解を述べてきたが、こうした問題は実のところ計算機にまつわる問題というよりは、西洋の点字を輸入する形で日本語点字を設計したことがトレードオフとして惹起した問題ともいえ、本格的な吟味にはかなり違った視点が必要であろう。問題を抱えるものであったとしても、日本語点字により視覚障害者は日本語文書の読解が可能であり、BRLTTY のようなスクリーンリーダが日本語に対応することは強く望まれる。

その一方で、Linux 計算機環境の問題に立ち返るのであれば、新世代のインプット・メソッドを標榜するプロジェクト・プロトコルが押しなべてコンソールプラットフォームにおける日本語入力を絶望的に軽視していることには正直戸惑う。その中で野首氏による screen-uim は意欲的な試みといえるのではないか。http://www.daionet.gr.jp/ knok/screen/uim.html 参照。

なお、ここにきてEXTRA for Unixの開発者である石川准氏がBRLTTYの日本語対応版BRLTTY Plusの開発に乗り出し、そのプロトタイプをhttp://fuji.u-shizuoka-ken.ac.jp/ ishikawa/devpgm.htm にて公開したことを報告しておく。この実装においては

- GNU screen からの日本語を含む画面情報の読み出し
- EXTRA For Unix のライブラリを用いた日本語点字の表示

という手法を採用している。ただし、現段階ではあくまでプロトタイプであり、本家 BRLTTY に日本語対応を組み込むテスト段階にある。一方で

- BRLTTY においては比較的軽視されてきた音声出力の積極的活用
- その一環としてクリエイトシステムの「Linux 版日本語音声合成ライブラリ」(通称 Dtalker for Linux) を用いた日本語音声出力
- System V ipc を利用した Gnu screen や日本語フロントエンドプロセッサとの協調のためのインフラ整備。(canfep がサポートされている。)

など、これまでのBRLTTYの方向性には収まりきらない新しい方針を打ち出してもいる。石川准氏の計算機に対するコンセプトは「エディタ中心主義のソフトウエア開発」http://fuji.u-shizuoka-ken.ac.jp/ ishikawa とくに Linux コンソール環境への意味づけについては「7 Windows から Linux へ」を参照されたい。

3 補論—-Gnopernicusを中心としたGNOME GUIのアクセシビリティ

紙数も少なくなってしまったので、ここではGnopernicusを中心としたGNOME GUIプラットフォームのアクセシビリティの概略を補論として紹介しておく。

3.1 開発動向

GNOME GUI [9] におけるアクセシビリティの必要性は、すでに2000年後半にはGNOMEディベロッパー、ユーザ（視覚障害者）の間で意識されていた。gnome.org内のアクセシビリティに関する資料は `http://developer.gnome.org/projects/gap/` から参照できる。

GNOMEはその設計段階でpango, bonoboといった抽象化された形での情報のやり取りを許容するメカニズムを組み込んでいたことが、アクセシビリティの実現に有利に働いた。[10] この点では後述するMSAAを後から追加したWindowsプラットフォームよりも設計思想において優れているともいえ、Sun Microsystemsにおいて長年アクセシビリティを担当してきたPeter Kornは機会がある毎にこの事実を強調している。こうしたメカニズムを活用してGNOME GUIのアクセシビリティ向上の試みがなされることとなった。筆者が参加した2001年3月のCSUNカンファレンス[11] におけるLinux, Unixのアクセシビリティに関するミーティングの時点で、すでに基本的な可能性は示されていた。そこでは、Festival Speech Synthesis System [12] によるソフトウェア音声合成を用いて、GNOMEデスクトップ上の

- アイコンの存在
- ダイアログの存在とその中に表示されるメッセージの読み上げ
- ユーザに選択が要求される項目の存在とその選択肢の数[13]

が可能であることが、部分的なテスト実装による実演を交えつつ示されていた。

その後GNOME GUIのアクセシビリティ向上のための設計、開発作業はSun Microsystemsを中心に進められてきた。Sunのアクセシビリティに対する姿勢については `http://wwws.sun.com/software/cover/2001-082` を参照されたい。メーリングリストにはOwen TylorやHavoc Peningtonといった GNOMEディベロッパの重鎮もしばしば発言している。ここで意図されていたのは主にスクリーンリーダなど

[9] GNOMEについては日本GNOMEユーザ会のYukihiro NAKAI, Akira TAGOH両氏に包括的なご教示をいただいた。記して感謝する。

[10] これに対して、あくまでX serverプロセスに画面描画を行わせることを想定して考案されたXプロトコルはアクセシビリティの観点からは不利であった。X window systemのスクリーンリーダ開発を目指したプロジェクトは押し並べて新たなプロトコルの規定から作業を開始している。

[11] CSUNカンファレンスはCSUN(California State University of Northridgeカリフォルニア州立大学ノースリッジ校) が中心となって毎年3月にロサンゼルスで開催する視覚障害者を含む身体障害者のための支援技術に関するカンファレンスである。そこでは、支援技術の研究・開発に携わっている研究者や企業による発表が行われ、併設された展示会場においては、各社がブースを設け、自社の新製品のデモを行う。世界規模で関係者が集まり、その年の支援技術の動向を占う上でも極めて重要なイヴェントと言える。支援技術の分野ではいわばCEBITやComdexとマイクロプロセッサーフォーラムを合わせたような地位を占めている。

2001年以来、この会場でLinux, Unixのアクセシビリティに関するミーティングが行われている。

[12] Festivalは現在もっとも広く利用されているフリーの英語音声合成ソフトウェアである。`http://www.festvox.org` 参照。

[13] ワープロソフトでファイルをセーブしようとする場合どのファイル形式で保存するかはユーザが任意に選択できるが、その選択肢の数。ある項目に対して選択肢がいくつ存在するかがあらかじめ分かることは見とおしを立てたオペレーションを行う上で重要である。たとえば、Windowsのハードウェアのドライバを手動で選択する場合、ドライバ提供ベンダを選択する項目の選択肢は100を超える。この事実が最初に分かっているかどうかで、同じベンダ選択をするのであっても作業の戦略の立て方が大きく変わる。

がアプリケーションやデスクトップの現在の状態を正攻法で読み出すためのミドルウェアの規定であった。スクリーンリーダはその性質上 OS、デスクトップ環境が意図していなかった機能を追加するソフトウェアであり、しばしばトリッキーな手法を用いることとなる。たとえば、MS-DOS 時代のスクリーンリーダはメモリ上の画面描画領域を定期的にスキャンするのが常套手段であったし、具体的な手法は違うにしろこうしたアプローチは Windows 用スクリーンリーダでも多用されている。Windows においては、Microsoft が MSAA(Microsoft Active Accessibility) を策定しており、これを用いることである程度アプリケーションの情報が取得できる。(MSAA はすぐ後に述べる ATK, AT-SPI が参考にしたものでもある。) だが、Microsoft が考慮に入れていないユーザインターフェイスライブラリを用い (自前で準備する) たりするアプリケーションは少なくない。MSAA を通じて得られる情報のみで包括的なスクリーンリーダを実現することには無理があり、優秀なスクリーンリーダほど上に述べたようなトリッキーなテクニックが駆使されている。しかし、これは一方でレスポンスの低下、OS の不安定化というデメリットも惹起し諸刃の剣である。

GNOME デスクトップ環境における正攻法でアプリケーションが保持している情報を読み出すミドルウェアは ATK(accessibility toolkit)、AT-SPI(service provider interface) として結実することになる。ATK, AT-SPI の登場により、原則 GTK2 アプリケーションに対してはスクリーンリーダの対応が可能となった。とはいうものの、アプリケーション側が積極的に ATK を利用して (リンクして) いる必要がある。

この状況は上に述べた Windows 上での MSAA を基礎とするアクセシビリティに似たところがある。すなわち、そもそも GTK2 を用いない、あるいは GTK2 以外のライブラリも併用するアプリケーションへの対応は不可能、あるいは限定的なものとならざるを得ない。2002 年にスペインで開催された GUADEC(GNOME User and Developer European Conference) において、GNOME GUI のアクセシビリティについて発表を行った Sun の Bill Haneman に対して会場からこの点を問題にする質問が提出されている。Haneman は GNOME プラットフォームにおいては GTK を用いるのが正統なのであって GTK を使わないアプリケーションプログラミングの方に問題がある旨の応答を行ったようである。

ATK, AT-SPI は視覚障害者のみのアクセシビリティを意識したものではない。たとえば、入力デバイスを抽象化し、キーボードやマウスに限定されない多様なデバイスによるデスクトップの操作が可能になるような仕組みも取り入れられている。これは上肢、下肢に障害を有する人のための支援技術である。視力にある程度の障害はあるが、画面を見て計算機を操作するいわゆる弱視者のために、画面を拡大するソフトウェア gnome-mag も開発されている。

3.2 Gnopernicus と GUI スクリーンリーダにおけるユーザビリティの問題

ミドルウェアの開発と前後して点字ピンディスプレイの製造販売業者である独 Baum 社が GNOME GUI 用スクリーンリーダ Gnopernicus の開発意思をアナウンスした。BAUM 社は点字ピンディスプレイの設計開発では大きな実績を有する。点字ピンディスプレイの画期的な小型化を実現した Vario シリーズ (掲載した写真のピンディスプレイ) で近年の点字ピンディスプレイ業界をリードする存在である。ATK, AT-SPI といったミドルウェアを Sun や GNOME ディベロッパが担当し、ユーザインターフェイスに関るスクリーンリーダを支援技術企業である BAUM が分担したのは正解であろう。BAUM 社の Gnopernicus に関する情報は `http://www.baum.ro/Gnopernicus.html` にて閲覧できる。

2002 年秋にはアルファ版ソース tarball が公開されている。ただし、Gnopernicus は GNOME2 を想定し開発されており、当時いまだ開発段階にあった GNOME2 において、それ以上にアルファ

ステイトであった Gnopernicus を動かすことに成功したユーザ（視覚障害者）は殆ど皆無であったようである。GNOME のヴァージョンが正式に 2 シリーズとなり、2003 年には Gnopernicus もユーザ（視覚障害者）によるテストがなんとか可能な段階に入った。Debian には Mario Lang [14] による Gnopernicus のパッケージが提供されており、適切に設定すれば festival を用いた音声合成を体験することができる。Gnopernicus は音声合成バックエンドとしては festival 以外に IBM Viavoice(すでに配布終了)、freetts(フリーではるが、java を用いるため Gnopernicus の Debian パッケージは対応していない。) などに対応する。現在はいまだアルファテストの段階にあり、

- Gnopernicus は起動時に festival バックエンドを起動するが Gnopernicus を終了しても festival のバックエンドプロセスは残る
- Festival のレスポンスがあまりよくなく、時に音声合成ができなくなる

といった深刻な問題を抱えている。また、GNOME デスクトップは普通 X window system を前提に動作するものであり、ユーザ (視覚障害者) は画面表示内容の確認なしで、なんとか GNOME デスクトップと Gnopernicus の起動までを手動で行う必要がある。

なお、点字ピンディスプレイでは自社製品である Vario シリーズにはネイティブ対応し、他の製品には前述の BrlAPI を通じて対応している。

Gnopernicus ではアプリケーションへの個別対応がかなり行われている。ATK, AT-SPI で gtk2 アプリケーションは対応できると言っても話はそれほど単純ではなく、スクリーンリーダの側が動いているアプリケーションを特定し、それぞれに見合った情報の能動的読み出しを行う。gedit についてはこうした個別対応がかなりしっかり行われており、相当実用的に利用できる。とはいうものの gedit が GNOME アプリケーションの中ではもっとも単純な部類に入り、gedit ですら個別対応が必要となるという厳しい見方もできる。Emacs, vi を使うから gedit はいらないというような実も蓋もない議論はひとまず棚上げにするにしても、GNOME GUI のアクセシビリティは企図についたばかりといえよう。gnome-terminal(端末エミュレータ), epiphany(ウェブブラウザ) も相当使える印象を持った。個別対応という思考はある意味でアクセシビリティを超えてユーザビリティの領域に踏み込むものでもある。同じスクリーンリーダといっても、こうした個別対応を駆使する発想は、画面の表示内容を忠実にダンプすることをコンセプトとする BRLTTY とは決定的に異なる。すでにかなりの実績がある Windows 用のスクリーンリーダにおいても (GUI の) 画面を忠実に反映するアプローチは衰退し、アプリケーションへの個別対応を駆使しつつ、GUI を音声あるいは点字ピンディスプレイ[15] でどのように分かりやすく表現するかが勝負所となっている。GUI 用のスクリーンリーダではアクセシビリティの一環としてユーザビリティへの配慮が不可欠となろう。アプリケーションへの個別対応とユーザビリティへの着目は emacspeak にも一脈通じるものがある。emacspeak は簡単に言ってしまえば、

- emacs バッファの読み上げを行う汎用関数や音声合成プロセスと通信を行う関数から構成されるミドルウェア部
- 各種 elisp パッケージの特定機能の実行時に付随的に実行される advice 群からなる個別対応部

の二つの部分からできている。よって emacspeak は膨大な個別対応の集積ともいえる。(事実、最近の emacspeak のアップデートは対応する elisp パッケージの増加が中心である。)

[14]Mario Lang は他に BRLTTY の Debian メンテナでもあり、FSG(FREE STANDARDS GROUP) Accessibility Workgroup `http://www.a11y.org` の評議員でもある。FSG Accessibility Workgroup は立ち上げからまだ日が浅く今後の動向は予測できない。評議員を見る限り多言語化、国際化への意識が希薄と思われるのが気がかりである。

[15]画面の忠実な反映という問題設定が成立しがたい GUI のアクセシビリティにおいては、文字情報の忠実な再現を売りとする点字ピンディスプレイのアドバンテージはかなり減殺される。

BRLTTY のような画面描画忠実反映形と Gnopernicus に代表される GUI スクリーンリーダや emacspeak のような個別対応型、いずれのアプローチが優れているかというのは、おそらくケース・バイ・ケースであろう。OS を使いこなすという観点からは画面を忠実に反映する包括的なスクリーンリーダは必須であり、クリティカルな局面で BRLTTY は必ず必要になる。他方、高いユーザビリティによって高度の生産性を実現することが望まれる文書入力などの作業では痒い所に手が届く個別対応が歓迎される。つまり、適材適所ということであり、事実大半のユーザ（視覚障害者）は BRLTTY と emacspeak を併用しているようである。

3.3 Gnopernicus の日本語対応の可能性

現状の Gnopernicus はバックエンドとして英語音声合成ソフトウェアしかサポートしていないため、日本語環境には対応しない。さて Gnoprnicus の日本語対応の難易度であるが、私見では絶望的に困難ということはないと考えられる。日本語音声合成エンジンは概して漢字かな混じり文を渡せば、それを発音に変換し、音声合成をするので、スクリーンリーダ側で処理してやる手間は英語環境とあまり変わらない。GNOME プラットフォームが相対的には多言語化、国際化をはっきりと意識した設計になっていることも追い風となろう。MSAA 導入当時の Windows プラットフォームでは英語版と日本語版の間で API に決定的な格差があり、日本語 Windows のアクセシビリティが欧米に大きく水をあけられる原因の一つとなった。GNOME プラットフォームでは少なくともソース・バイナリは共通であり、こうした格差はいくらか緩和される見こみがある。問題は

- ATK, AT-SPI がどこまで多言語化、国際化にセンシティブか
- インプットメソッドなどマルチバイと文字コード圏固有のソフトウェア群がどこまでフォローできるか

といった点であろう。ただし、このいくらか楽観的な見通しはあくまで憶測であることをことわっておく。

むしろ、日本語対応で深刻なのは基盤ソフトウェアの方である。現状フリーの日本語音声合成エンジンは存在しない。現在 Linux 上で実用に耐える音声合成エンジンはクリエイトシステム社のプロプライエタリなソフトウェア Dtalker [16] のみである。同音声合成エンジンの完成度は決して低くはない。が、フリーのデスクトップ環境を実現するという GNOME デスクトップの目的からすれば、プロプライエタリなソフトウェアに全面的に依拠してしまうことは問題無しとは言いがたい。[17]

3.4 アクセシビリティの社会的意味づけの必要

他方、（欧米にしろ日本にしろ）はたしてユーザ（視覚障害者）の間に GNOME GUI を利用したいというインセンティブがどれほどあるかは、実は未知数である。PC-Unix を活用している視

[16] `http://www.createsystem.co.jp` BEP(Bilingual Emacspeak Plathome) もこれを用いている。

[17] なお、音声合成エンジンとは言いがたいが Galatea `http://hil.t.u-tokyo.ac.jp/ galatea/` は興味深いプロジェクトである。同プロジェクトは、はじめ IPA 関連のプロジェクトが頻繁に採用する原子力などでの利用を禁止するライセンスを採用していた。基盤技術として有望な同ソフトウェアの可能性にかんがみ、筆者が作者にライセンスの変更を相談したところ快諾いただき、現在のライセンスとなっている。現ライセンスが OSD(OpenSource Definition `http://www.opensource.org/docs/definition.php`)、DFSG(Debian Free Software Guideline `http://www.debian.org/social_contract#guidelines`) に厳密に適合するかどうかは難しい問題ではあるが、フリーの基盤技術が渇望されている分野に積極的な対応をしていただいたことは感謝に耐えない。一方、音声合成に限らず基盤ソフトウェアの開発の支援に積極的な IPA には、一ユーザとしてぜひとも現在のライセンスのスタンスの再考を願いたい。

覚障害者には (Windows で失った)MS-DOS で享受していたユーザインターフェイスを取り戻したいという人間が多い。そして、PC-Unix プラットフォームのコンソールは dos 時代に享受していた以上のオペレーション環境を与えてくれた。視覚障害者にとって desktopfree な環境は否定しがたい魅力がある。こうした意識については The command line interface - ideal for blind users. `http://www.eklhad.net/cli.html` を参照。そのため、仮に Gnopernicus が Windows 用のスクリーンリーダが実現しているクオリティのアクセシビリティ、ユーザビリティを実現したとしても、GNOME プラットフォームを喜んで使うかどうかはどうしても疑いが残る。実際、GNOME GUI のアクセシビリティを主導してきたのは Sun microsystems に代表される企業やディベロッパの側であった。ここには、GUI としての GNOME プラットフォームをハックする足がかりが視覚障害者にはない（なかった）という理由もあるが、彼らに GUI への渇望がなかったのも事実である。ただし、現状 GUI 環境でしかできない重要な作業も存在する

- オフィススウィートの活用
  - 特定形式のワープロデータの取り扱い、オフィス文書の読み書き。
  - 表計算の活用
  - プレゼンテーション（これは重要）
- JavaScript, Java, Flash を含む Web コンテンツの閲覧
- GUI を利用したアプリケーションの活用。商用のシステム管理ツールやバックアップツールの大半は GUI のクライアントソフトウェアのみをバンドルしており、企業でエンジニアとして勤務する視覚障害者にとっては、これらのソフトウェアが利用できるか否かは死活問題である。

は看過しがたい例である。

それではなぜ企業が主導して GNOME GUI のアクセシビリティ向上の努力をしてきたかというと、（企業側の社会的責任の関心の他に）アメリカ合衆国の政策がある。合衆国は連邦リハビリテーション法 508 条により、合衆国政府が調達する物品はアクセシブルなものに限ると規定した。周知のように合衆国政府は大量の物品を民間から調達しており、この市場を失うことは企業にとって死活問題である。この規定により合衆国政府に LInux デスクトップシステムを納入するにはアクセシビリティが現実的な問題となった。これは自社の OS Solaris に GNOME GUI の採用を決定していた Sun microsystems にとっては特に重大な問題であった。同社の GNOME GUI アクセシビリティへの積極的なコミットメントの背景にはこうした事情があったことを意識しておく必要があろう。一方、Gnopernicus が最低限の成果をあげつつある今、周囲の関心の低さも手伝って、GNOME プラットフォームのアクセシビリティ開発インセンティブは以前に比して明らかに減退しているようである。[18]

4 総論

本稿では、BRLTTY の機能概要と点字体系の特質を中心に論じ、合わせて Gnopernicus の簡単な紹介を試みた。BRLTTY の開発の歴史は

[18]連邦リハビリテーション法 508 条の実施の細目については、いまだ十分調査できていないが、必ずしも自社のプロダクトのアクセシビリティを自社で責任をもって確保する必要はないようである。Microsoft は Windows について MSAA を規定し、提供してはいるが、自社で Windows 用のスクリーンリーダを提供してこなかった。Windows 2000 以後 narrator という簡易スクリーンリーダがバンドルされるようになったが、実用的ではない。

- 最初期に核となる機能を決定的な完成度でリリースし、
- 対応するデバイス・原語を拡充し、多様な環境に対応し、
- ソフトウェアの内部の構成を整理し、汎用性を拡大する

というものであった。このプロセスは成功するオープンソースソフトウェアプロジェクトにしばしば見られるものである。一方、この解説を通じて計算機データを点字で表現することについてある程度の理解を得ていただけれたのであれば、筆者としては嬉しい。すなわち、

- 点字は限られたパターンで普通文字の表現を試みるものであり、表現能力にはシビアな限界があり、
- NABCC, UEBC あるいは英語略字に代表される多用な創意工夫がなされている
- 漢字文章をかな書きで表現する日本語点字は、かなり特殊な位置にあり、計算機上での処理には固有の困難が存在する

という点を確認しておきたい。

最後になったが、オープンソースソフトウェアとしての BRLTTY および Linux プラットフォームについて最低限のコメントを付け加えておく。MS-DOS, Windows のスクリーンリーダではトリッキーな手法が常套手段になっていることについてはすでに述べた。こうした正攻法とはいえない手法を多用することはあまり誉められた話ではないが、MS-DOS, Windows というプラットフォームを想定するならやむをえないことも事実である。それに対して、オープンソースソフトウェアを基本とする Linux プラットフォームでは、トリッキーな手法を多用する必然性はない。なぜなら、os を含む当該ソフトウェアに対するパッチを作成し、プロプライエタリな環境ではトリッキーな手法でしか行えなかった情報の読み出しを、正攻法でできるようにすることが十分可能だからである。BRLTTY が用意した GNU screen に対するパッチはこの好例であろう。gtk2 アプリであれば ATK に対応させ、リンクすることで GNOPERNICUS での利用が可能になるのも同様である。オープンソースはアクセシビリティにおいて原理的に有利な条件であることを強調しておきたい。