国際化端末エミュレータの開発

関場 治朗 <jir@yamato.ibm.com>, 鷲澤 正英 <washi@jp.ibm.com>
日本アイ・ビー・エム株式会社 ソフトウェア開発研究所

平成 14 年 8 月 12 日

概要: Linux 上では, 限られた言語を扱うことができる端末エミュレータが各言語や地域ごとに存在するが, POSIX locale model に基づいて国際化された端末エミュレータは存在しない. そこで Linux 上で動作し, BiDirectional 言語を含む多言語表示を可能とする国際化された端末エミュレータを開発した. 本稿では, その概要を述べる.

1 はじめに

Linux をはじめとするフリーソフトウェアに対し, 国際化/多言語化の要求が高まっている [1]. しかし, Character User Interface である端末エミュレータは, POSIX locale model[2] に基づいて国際化された portable な実装は存在しない. X Window System 上では mlterm[3] や xfree86 xterm[4] などが多国語で利用できるようになってきているが, これらは POSIX locale model に基づいてはいない. 我々は X11R6.5.1 に含まれる xterm に対して POSIX locale model に基づく国際化パッチを作ってきたが [5], BiDi 拡張や速度などの問題を抱えている.

また, コンソール環境においては日本語表示を中心とした jfbterm[6] や日本語/中国語/韓国語などのアジア言語に特化し, 描画スピードを重視した zhcon[7] など, 多くの端末プログラムが開発されてきているが, それらは限定したコードセットのみを扱い, 地域化の範囲を超えていない.

さらに, メニューバーなどを付加し Usability を改良した, aterm, eterm, wterm, gnome-term, konsole 等の端末エミュレータが多数存在する.

端末エミュレータの Capability はほぼ同等なものであるにも関わらず, この様に多数の端末エミュレータが存在するため, 個々の端末エミュレータの国際化は現実的ではない.

そこで, 端末エミュレータを特定の環境に依存しないモデル部分と環境に依存するユーザインターフェース部分を分離した, 国際化端末エミュレータのモデル設計と実装を行った. 今回, そのモデルを利用し BiDi(Bi Directional) 言語を含めた多言語の表示に対応した X Window System および Linux のフレームバッファ用の国際化端末エミュレータを開発した. 本稿ではモデルの設計とモデルの実装を利用したターミナルエミュレータの概要を述べる.

2 ソフトウェアの国際化

ソフトウェアの国際化とは, プログラムを言語や地域をあらわすロケールに依存した部分と依存していない部分に切り分ける手法である. ロケールに依存したものとして, ユーザが読むメッセージ, 日付や数字のフォーマット, 通貨記号, コードセットなどがある. また, ロケールに依存しないものとして, そのプログラムのアルゴリズム自身などがあげられる. 国際化されたプログラムはロケールに依存した地域化部分を各ロケール毎に用意することで, 一つのバイナリを使って全てのロケールで地域化されたプログラムが利用できることになる.

図 1 に国際化ソフトウェアのイメージ図を表 1 に国際化と地域化の比較を示す. 表 1 は参考文献 [8] から引用した. Internationalization, すなわち国際化は”I” と”N” の間に 18 文字あることから I18N と省略して表記される. 同様に, 地域化 (Localization) は L10N と表記されることがある.

表 1: Merit and Demerit of I18N/L10N

	利点	欠点
I18N	• 国際化部分が共通 • ロケール依存の追加・修正が最小 • 機能拡張が容易	• 実行速度の低下, サイズの肥大化 • 特定のロケールへの要求が通りにくい
L10N	• 小規模の場合修正は比較的容易 • オリジナルに影響がない	• 各ロケールのリリースの遅れ • 新しいバージョン毎に修正が必要 • ソースコードの不統一

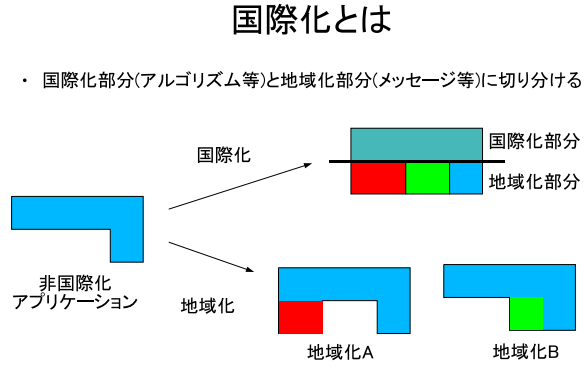

図 1: I18N and L10N

2.1 POSIX Locale Model

POSIX Locale Model とは, ANSI C で各国語サポートを行うために導入された概念で, 国際化プログラムの根本となるモデルである. 一つのバイナリプログラムが実行環境によって動作を変えることによって, 各ロケールに合った処理を行うことを可能にしている. ANSI C は国際規格である ISO C になり, ロケールの概念は POSIX に引き継がれた.

プログラムは setlocale(3) でそのプログラムのロケールを設定し, POSIX で定められた国際化 API を利用することで各ロケールの処理を抽象的に扱うことが可能となる.

2.2 UTF-8(Unicode) と国際化

プログラムの扱うコードセットを UTF-8 とすることで国際化ができるのでは? ロケールモデルも, もう必要ないのでは? という期待や疑問があるが, これは大きな間違いである. 既に述べたように国際化という観点からすればコードセットはロケールに依存した多くの処理の一つにすぎない. メッセージなどを実行環境によって切り換えるためにロケールは必要である.

また, UTF-8 は現在利用しているコードセットと完全に互換性があるわけではない. プログラムの内部コードを UTF-8 にし, 他のコードセットを利用する場合外部フィルタによって UTF-8 に変換すれば, EUC-JP などのコードセットも正しく利用できると思われがちだが, 場合によっては問題が生じる. 典型的な問題は EUC-JP と UTF-8 でロシア語などの文字幅が異なるといった問題である. さらに, 将来に渡って UTF-8 が今後制定される全てのコードセットをサポートできるとは限らない.

一方 国際化されたプログラムはコードセットを

ロケールに依存した部分として切り離している. このため国際化されたプログラムでは UTF-8 をコードセットとして利用するロケールを用意することで, UTF-8 を他のコードセットと同様に正しく扱うことができる. 国際化プログラムからすれば, UTF-8 はたまたま全てのロケールで利用できるコードセットの一つにすぎないわけである. そのため今後新しい コードセットをサポートしたい場合, そのコードセットを利用したロケールを作ることでプログラムは変更することなくそのコードセットを扱うことができる.

これは デバイスドライバの考え方と同じである. アプリケーションはデバイスを直接コントロールせずに, read/write などのシステムコールを通し, デバイスドライバを利用してデバイスにアクセスする. コードセットをデバイスと考えると, ロケールはデバイスドライバであり, プログラムは API を通しロケールを利用してコードセットを扱う. UTF-8 は USB デバイスかもしれないが, それが全てではない.

3 既存端末エミュレータの問題点

端末エミュレータプログラムは 1 章で取り上げたほかにも多数の実装が存在するが, これらの多数の端末エミュレータは端末エミュレータとしての機能に大きな差はなく, 特定の言語処理を付け加えた地域化, 外観やユーザビリティの拡張, そして実行環境 (X/fb/vga 等) の差異などにとどまっている.

これらからユーザが様々な外観やユーザビリティを求めているのではないかと推測できる. 実際, xterm の地域化である kterm 以外にも日本語が使える rxvt や kterm の拡張パッチなどが存在しており, 個々にメンテナンスされている. 端末エミュレータを国際化することで, ユーザが利用したい言語を扱うことができるようにはなる. しかし, 前述のようにユーザは様々な外観やユーザビリティを求めるため, ある言語を利用できることだけではユーザの要求を満たせない.

ところが, 既存の端末エミュレータは端末エミュレータの機能部分と外観などの部分が密接に関連しており, 外観やユーザビリティの拡張のみを行いたい場合に, 端末エミュレータの機能としての部分を再利用することが難しい. このため X 環境では主に xterm からの派生として xterm のソースを元にし, 端末エミュレータ機能部分はそのまま利用して, 外観部分を独自に変更している場合が多い. この場合, 端末エミュレータ部分の機能は同じコードであるにも関わらず, 個々のプログラムソースコードとして分散してしまっている. また, コンソール環境などでは再利用が難しく, スクラッチから作ることを強いられてしまう.

これらの状況から, 個々の端末エミュレータの国際化を行うことだけでは, 新たに別の外観やユーザビリティをもった国際化端末エミュレータを作ることが難しく, 現状の問題を解決できない.

4 デザイン

3 章の問題は既存の端末エミュレータが端末エミュレータ機能部分と外観などの部分が適切に分離されていないことから生じている. 端末エミュレータ部分と外観などの部分を分離することで, 端末エミュレータはさまざまな外観を持つことが可能となり, 国際化された端末エミュレータ機能部分を再利用することができる.

そこで, 端末エミュレータを特定の環境に依存しないモデル部分と, 環境に依存する View 部分を分離した. 図 2 に構成図を示す.

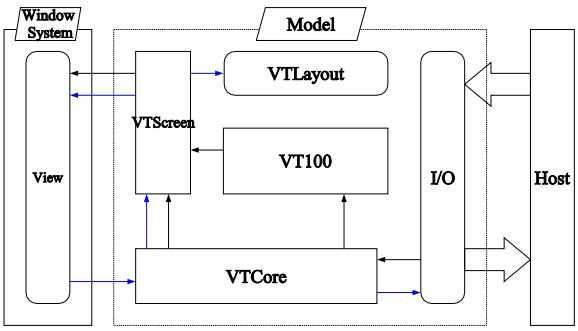

図 2: Design of Terminal Emulater Model

モデル部分は端末エミュレータ自身の機能部分で主にエスケープシーケンスの解釈を行う. この部分は国際化するにあたり, 一文字が多バイトであるようなバイト列の解釈を正しく行う必要が

ある. モデル部分のモジュール群の中はユーザインタフェースと同様に環境依存の I/O や Layout service[1] などは抽象化してあるため, Unix[2] ライクなシステム以外にも容易に移植可能になっている.

View 部分はモデル部分に対し, いくつかの callback 関数 (指定した列/行に対して文字列を描画, 前景/背景色を設定, 等) を設定する. また, ユーザからの入力を接続ホストに送信する為にモデルの機能を利用する.

図 2 の各モジュールは以下のように動作する.

IO: Host に接続する I/O のための インターフェース. 実際には擬似 (pseudo) 端末 や telnet/ssh などのコネクションを実装する. 接続した先の Host からのバイト列 を取得する.

VTCore: 全体を管理するオブジェクト. モデルを利用する場合, VTCore のインターフェースを利用しモデルをコントロールする. VTCore はバイト列から キャラクタ を切り出し, VT100 parser に キャラクタ を渡す. 以後の処理は キャラクタ ベースで行われる. 現在のロケールに不適切なバイト列や Binary を受け取った場合は, 先頭のバイトを切り出し 1 Column, Non Printable な キャラクタとして扱う.

VT100 : VT100 をベースにした有限状態機械. Escape Sequence などの Parser. 入力 キャラクタは Parse され, 現在の状態に従い, 登録された VTScreen の callback 関数を呼び出し, 状態を遷移していく.

VTScreen: 端末画面のモデル. キャラクタの幅などから, 画面のレイアウトを計算し, キャラクタをセル単位に論理的なレイアウトで保存する. キャラクタの合成を考えセルの中に複数のキャラクタを保持する. Display キャラクタがセルを複数占有する場合, 最初のセルに全てのキャラクタを保存し, 他のセルは Display キャラクタが書かれている情報を保持する.

画面の出力は View インターフェースを実装した個々の環境の View オブジェクトに委譲する. Layout が必要な場合は以下の VTLayout を利用して, 適時 Layout 処理を行う.

VTLayout: Complex Text Layout Languages (Arabic/Hebrew などの Bidirectional Language や, Thai などの複雑な Layout が必要な言語) 用の Layout Engine インターフェース. Portable Layout Service(PLS) library や fribidi library[9] などの 外部 Layout Engine を利用するような wrapper. Layout Engine は論理的な キャラクタ 列を視覚的な キャラクタ や Display キャラクタ の列に変換する.

View: 端末の View. 主に, ユーザからの入力と文字の描画を行う. VTScreen から描画関数などが呼ばれる.

5 実装

モデル部分の実装として, Unix システム上に shared library を構築し, X Window System および, Linux frame buffer 上に View 部分を実装した. 実装したプログラムとライブラリは x86/PowerPC/SPARC 上の Linux の X Window System および frame buffer , SPARC 上の Solaris の X Window System で動作を確認した.

図 3 に構成図を示す. fbiterm は Linux Frame Buffer 上で動く端末エミュレータである. libXfont を利用することで,pcf/bdf 形式のフォントを扱えるようになっている. 図中 libiterm が 4 章, 図 2 のモデル部分に相当し, fbiterm が View にあたる部分である.

また, 図 4 は X 上で動作する xiterm の構成図である. 図中 libXiterm が 4 章, 図 2 の View に相当している. libXiterm 部分は Athena Widget となっており, ライブラリとして独立している. このため, xiterm 自身は非常に少ないコード量になっている. 図 5 は xiterm 実行中の画面である.

View とモデルを繋ぐ API は以下の通りであ

[1] アラビア語やヘブライ語、タイ語などの複雑な表記を行う言語のレイアウトを行うライブラリなど

[2] "Unix" は The Open Group の 米国およびその他の国における登録商標

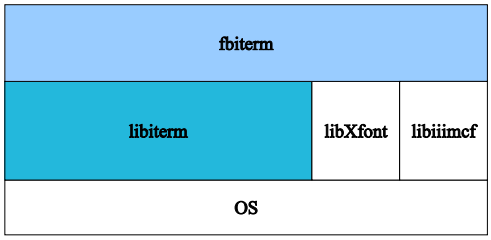

図 3: Structure of FB I18N terminal emulator

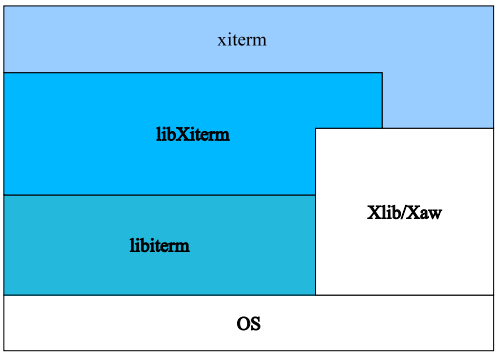

図 4: Structure of X I18N terminal emulator

る．既存の物と異なる外観やユーザビリティをもった端末エミュレータを新たに構築する場合，View としてこれらの関数を実装するだけよい．端末エミュレータの機能はモデルに実装されている．これにより，様々な外観をもつ国際化端末エミュレータの実装が容易になる．

draw_text： 指定した行/列にテキストを描画する

update_cursor_position: カーソルの位置のアップデート

set_rendition: 文字の背景色や前景色，反転，太字，下線などの属性を決める

clear_rect: 行と列で指定した矩形を背景色で塗りつぶす

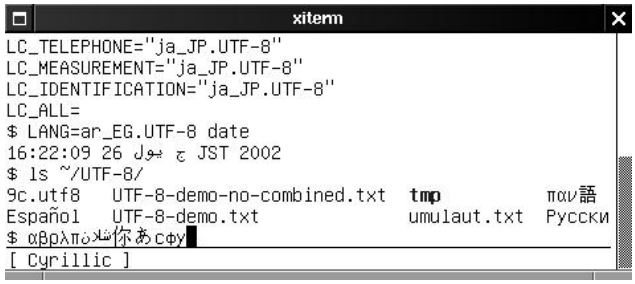

図 5: ScreenShot of X I18N terminal emulator

update_rect: 行と列で指定した矩形の再描画を促す

swap_video: 前景色と背景色を入れ換える

ring: ベルを鳴らす

resize_request: 端末のサイズ変更のリクエスト

notify_osc: Operating System Command．ウィドウのタイトルやフォントを変更する．

update_scrollbar: スクロールバーの位置/サイズを変更する

scroll_view: 画面をスクロールさせる

6 パフォーマンス

端末はホストから文字を受け取って画面に表示を行う．ところが，大量にホストから文字を受け取った場合，X Window System などのイベント通信にコストがかかるシステム上ではパフォーマンスが著しく低下した．このため，このモデルでは以下のようにできる限り描画を遅らせることでパフォーマンスの低下を軽減させた．

端末は通常ホストから文字を受け取ると現在のカーソル位置に文字をそのまま表示する．この動作の通りに一文字ごとに逐次画面に描画命令を送った場合上記のようなパフォーマンス低下を招く．このため通常は表示文字を受け取り続け，コントロールキャラクタやバッファの終了時点で初めて描画を始める．

本モデルではこれをさらにおし進めコントロールキャラクタなどの文字を受け取っても描画を開始せず，バッファが完全に終了したところで差分

のみを描画するようにしている．この場合, スクロールなどは非常に高速に動作するが, 画面の書き換え頻度が少ないために滑らかなスクロールにはならないという欠点がある．

6.1 パフォーマンス計測

国際化コンソールプログラムと他のコンソール端末プログラムでの描画において, パフォーマンスの観点での比較を行なった．ここでは, あるディレクトリ以下を ls -lR コマンドにより表示させ, リスティングが終了するまでの所要時間を複数回計測し, その平均値を算出し比較した．表示されるファイル数は 2355, ディレクトリ数 254, 全体として 3319 行のリスティングに対する所要時間となる．

表 2: Performance of ‘ls -lR’

I18N console	9.623[s]
zhcon	8.798[s]
jfbterm	39.117[s]
console(frame buffer)	41.735[s]

表 2 に示すように, 国際化コンソールは, jfbterm やフレームバッファ上のコンソールでの描画スピードより早く, パフォーマンスを重視している zhcon と比較しても遜色ない描画スピードになっていることがわかる．

7 まとめと今後の課題

これまで, Linux において POSIX locale model に基づいて国際化された端末エミュレータはなかったが, 今回, X Window System 上および Linux frame buffer 上に国際化された端末エミュレータを開発した．

今後の課題として, 描画の同期アルゴリズムの高速化, GNOME, KDE など様々な Desktop 環境への適用 (図 6), また, コンソールプログラムは多言語入力可能な IM(Input Method) の実装, などがあげられる．

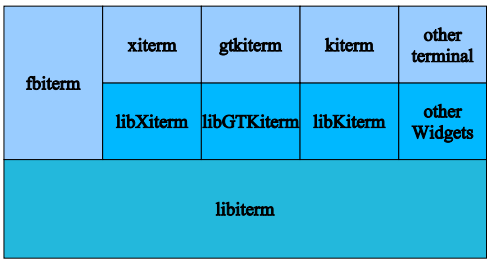

図 6: libiterm future view

謝辞

国際化端末エミュレータの開発にあたり, 助言をしていただいた方々, 特に杉山 昌治氏, 知念 充氏, 稗方 和夫, 長谷川 勇氏に深く感謝いたします．

参考文献

[1] The Free Standards Group Linux Internationalization Initiative (http://www.li18nux.org/)

[2] The Open Group Base Specifications Issue 6 IEEE Std 1003.1-2001

[3] MLTERM (http://mlterm.sourceforge.net/index.html)

[4] XFree86 xterm (http://dickey.his.com/xterm/)

[5] The Free Standards Group Linux Internationalization Initiative/ Advanced Utility Development subgroup (http://www.li18nux.org/ subgroups/utildev/dli18npatch2.html)

[6] JFBTERM/ME (http://www3.justnet.ne.jp/ nmasu/linux/jfbterm/)

[7] zhcon (http://zhcon.gnuchina.org/)

[8] 清兼 義弘, 末廣 陽一: 国際化プログラミング (ISBN4-320-02904-6)

[9] fribidi (http://fribidi.sourceforge.net/)