システムソフトウェアと
オペレーティング・システム 90−15
（２００２．６．２８）

リモートエディタの日本語ターミナルへの応用

宮里 忍 琉球大学理工学研究科情報工学専攻
河野 真治 琉球大学, 科学技術振興事業団さきがけ研究 21

概要

我々が提案しているリモートエディティング・プロトコル (REP) は、テキスト編集に特化したネットワーク・プロトコルである。サーバおよびクライアントに存在するテキストバッファを操作するコマンド列によって構成されたプロトコルは、ネットワークの遅延、切断に強く、巨大なテキストファイルの編集を可能にする。いままで、pico, Emacs, Mac OS X の TextEdit などに REP を応用し、その有効性を示して来た。ここでは、 REP が単なるファイル編集機能だけでなく、複数のアプリケーションを結ぶプロトコルとして使用できることを示す。実際に、kterm 上にリモートエディタサーバとしての機能を与え、TextEdit と接続し、REP をサポートした TextEdit 上で日本語ターミナル機能を実現することができる。

Application of Remote Editing Protocol on Japanese Terminal Emulator

Shinobu Miyazato Specialty of Information Engineering, University of the Ryukyus.
Shinji Kono Information Engineering, University of the Ryukyus, Japan Science and Technology Corporation

Abstract

We introduce Remote Editing Protocol (REP) which is a special network protocol for text editing. REP is a command sequence on text buffers in clients and servers, which has good resistance on network delay or network failure, and it is good for large text editing. We have been implemented REP on pico, Emacs and Mac OS X's TextEdit. Here we show that REP is not mere remote text editing, but it is also a communicating protocol among applications. We present a combination of kterm (Terminal Program on X Window) and TextEdit. The result is a Japanese Terminal Emulator on Mac OS X.

我々が提案しているリモートエディティング・プロトコル (REP) は、テキスト編集に特化したネットワーク・プロトコルである。サーバおよびクライアントに存在するテキストバッファを操作するコマンド列によって構成されたプロトコルは、ネットワークの遅延、切断に強く、巨大なテキストファイルの編集を可能にする。いままで、pico, Emacs, Mac OS X の TextEdit などに REP を応用し、その有効性を示して来た。ここでは、 REP が単なるファイル編集機能だけでなく、複数のアプリケーションを結ぶプロトコルとして使用できることを示す。

実際に、 kterm にリモートエディタサーバとしての機能を与え、TextEdit と接続し、REP をサポートした TextEdit 上で日本語ターミナル機能を実現することができる。

1　リモートエディタについて

ここでは、リモートエディタについて説明し、通常のエディタとの違いを示す。

リモートエディタでは、従来のエディタの機能を、クライアントとサーバの二つに分割し、互いに通信することでデータの編集・管理を行う。サーバ側では主にファイルと編集バッファの管理を行う。クライアント側は、サーバに接続し編集を行う。

開いたファイルのバッファをサーバ側で管理することにより、NFS のように場合によっては大きなデータを送らなければならない、といったことがなくなる。また、クライアントにバッファを持つことにより、ネットワークの切断が起きても、クライアントのバッファを用いて編集作業を行い、ネットワークが回復したところで変更箇所をサーバに送るようにすることで、編集作業の中断を最小限におさえることができる。

また、通常のエディタでは、編集されたファイルを他のクライアントが編集を行うには、そのファイルのアクセス権限の変更や、一旦自分のカレントディレクトリにコピーして、そのファイルを編集するなどの作業が必要になる。

リモートエディタでは、すべてのファイルをサーバで管理することにより、他のクライアントが編集したファイルに対して追加編集を行うといったことができる。そのため、多人数でのプログラムの作成や、書類の編集などが容易になることが期待できる。

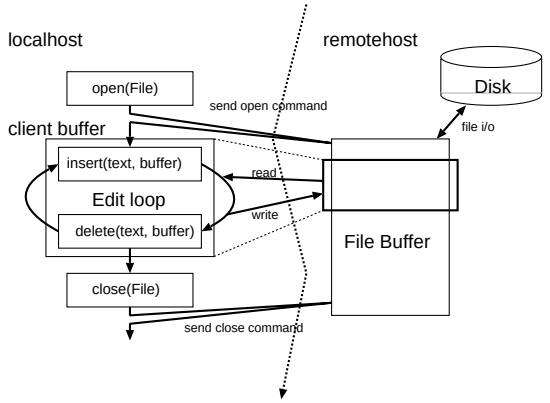

図 1: リモートエディタでのファイル編集

1.1　テキストの編集に必要な要素

テキストエディタでファイルを開いた場合、ファイルの内容は全て、メモリ上にバッファリングされる。このうち、ユーザが編集の際に必要な情報は、画面に表示する分のデータである。

テキストエディタでの編集において、ユーザの直接入力による変更は、画面に表示されている部分にしか行われない。また、テキストエディタには、入力文字列の検索・置換機能が備わっていることが多い。文字列の置換も文書編集機能の一部ではあるが、これはユーザ側には見えない部分での編集操作である。

つまり、ネットワークを介した、エディタによる編集では、クライアント側で行うバッファリングは、画面表示分だけ行うのが最適であると考えられる。

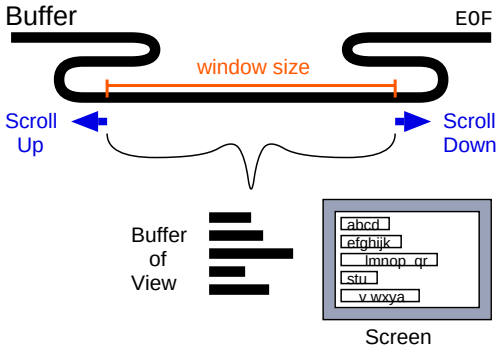

図 2: リモートエディタにおけるデータ転送

2　REP について

リモートエディタ間は REP を用いて接続される。ここれでは、REP について述べる。

2.1　パケット形式

クライアントからのパケットは、2 バイトのコマンド識別子に、Client ID、Offset 値、テキスト値を引数とする形である。リモートエディタでは Offset が通信の単位となる。

編集コマンド	**Client ID**	Offset値	テキスト値

図 3: パケット形式

2.2 編集コマンド

次に編集コマンドを示す。ここでのサーバとは、編集対象となるバッファを持つエディタ、クライアントとは、そのバッファへ接続し編集を行うエディタのことである。

編集コマンドは主に open、save、close、read、write、update からなる。

- open : サーバ側のファイルの読み込みを行う。引数としてファイル名を与える。サーバ側でファイルが開かれるとクライアントへ、そのバッファ名を、コマンド識別子を付加して返す。また、同時に次の read を行う。
- save : サーバエディタ側のバッファを実際にファイルに書き出す。ファイルはサーバ側のホストに書き出される。
- close : 編集バッファを破棄する。サーバはクライアントとのコネクションを切断する。
- read : クライアントで open コマンドが実行されたとき、またはエディタの行移動などで、画面が更新される際に、必要なテキストデータをサーバへ要求する。引数として、要求するテキストのオフセット値を与える。サーバはこのコマンドを受けると、オフセット値から適当な長さのテキストを、コマンド識別子を付加してクライアントへ返す。同時にサーバでは、クライアントへ送ったテキストに対応するバッファのオフセット値を記憶する。
- write : サーバのバッファに対し、クライアント側で行った変更を加える。引数として、テキストデータおよびそのオフセット値を与える。サーバはこのコマンドを受けると、サーバ側のバッファにおけるオフセット値と、受け取ったデータに含まれる引数のオフセット値を元に、現在のバッファ領域のテキストを受け取ったデータに置換する。そしてサーバ

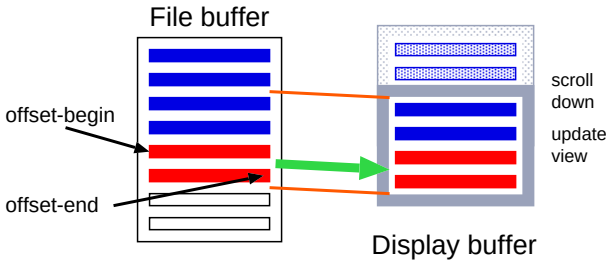

図 4: read コマンド

は、クライアントに対して、新しいオフセット値を返し、そのオフセット値を記憶する。

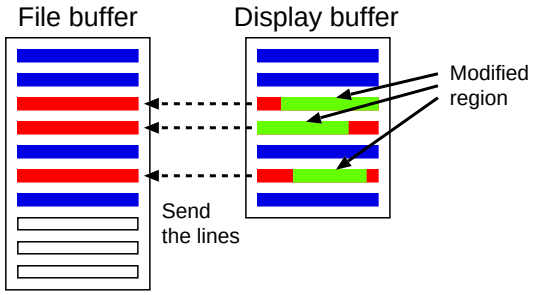

図 5: write コマンド

- update : 複数ユーザによる単一バッファに対しての同時編集時に、サーバ側のバッファに対して行われた変更を、他のクライアントエディタに反映させる。このコマンドは、あるクライアントによって編集された Offset 値とそのテキストを引数として受け取り、それを同じバッファを編集中の他のクライアントエディタへ送る。クライアントはこのコマンドを受け取ると、指定された Offset 値のテキストを受け取ったテキストデータに置換えする。

2.3 クライアント・サーバ型リモートエディタ

ここで紹介する実装は、一つのエディタをサーバとし、そこへ複数のクライアントエディタを接続する。ファイルの入出力および管理はサーバ側で行う。

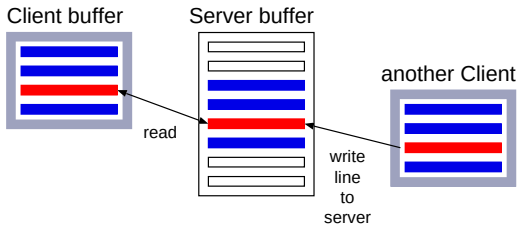

図 6: update コマンド

2.4 TextEdit おける実装

TextEdit は Mac OS X に付属しているエディタで、通常のプレインテキストエディタの他に RTF(RichTextFormat)、HTML エディタとして使うことの出来る多機能なエディタである。

2.4.1 リモートエディタとしての TextEdit

TextEdit にはリモートエディタとしての機能と TextEdit 本来の機能を繋ぐ TextEditAdditions 、REP メッセージ送受信を行う communicator、パケットを展開処理するための encoder-decoder を実装した。(図 7)

- TextEditAdditions: 本来の TextEdit の機能とリモートエディタの機能を接続する。TextEdit からのイベントをとらえ、encoder-decoder を通して接続先に伝える。また、encoder-decoder から受け取る処理も実際はここで処理される。TextEdit 固有の処理はここで行われる。

- communicator: TCP/IP を用いたメッセージ送受信インターフェースであり、ソケットの管理を行い、エディタ間のデータストリームを提供する。また、REP メッセージを encoder-decoder に渡すと共に、encoder-decoder から受け取るメッセージを接続先へ送る。

- encoder-decoder: REP メッセージを展開処理し、TextEdit のエディタ部分へ相当する処理を渡す。また、REP メッセージを生成し communicator に渡す。

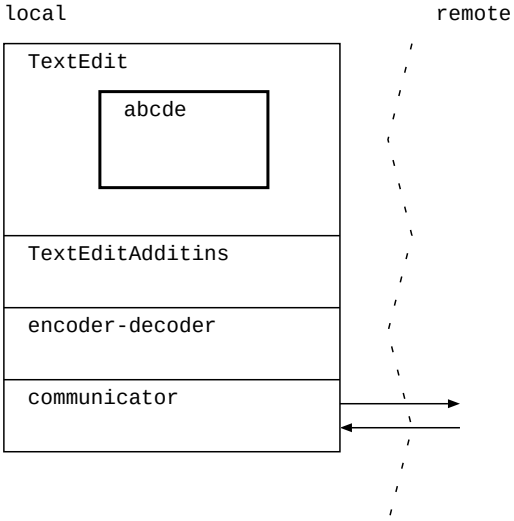

図 7: TextEdit における実装

3 REP による日本語端末の実現

端末エミュレータは、キーボードによる入力と画面への出力をグラフィックス画面上でシミュレーションするプログラムである。Unix は、シェルと呼ばれるコマンドインタプリタを中心としたユーザインタフェースを持っているので、日本語端末は重要なアプリケーションの一つである。日本語端末の機能としては、以下のようなものがある。

- キーボードやマウスの入力を端末アプリケーションに伝える

- 出力された文字を画面に表示する

- vt100 エミュレーションと呼ばれる画面操作

- リアルタイム入力を Unix API を通じてアプリケーションに伝える

この他に、フォントの管理や、対応するグラフィックスの機能に関する処理が必要となる。

ここでは、kterm という X Window 用の日本語端末を変更し、REP の機能を付け加え、TextEdit と通信することにより、TextEdit を日本語端末として使うこととする。TextEdit は、画面表示などの機能を既に実装しているので、REP を使う場合には、ウィンドウシステムなどに依存する部分を実装する必要はない。

Unix では入出力のリアルタイム処理は、termios と呼ばれる POSIX で規定された API を使用する。また、キーボードの入力の一部は、割込み処理に相当する signal を生成する必要がある。これらの機能は、pseudo tty と呼ばれる疑似端末エミュレータにより実現されている。この部分は、kterm に実装さている部分をそのまま用いることが出来る。(図 8)

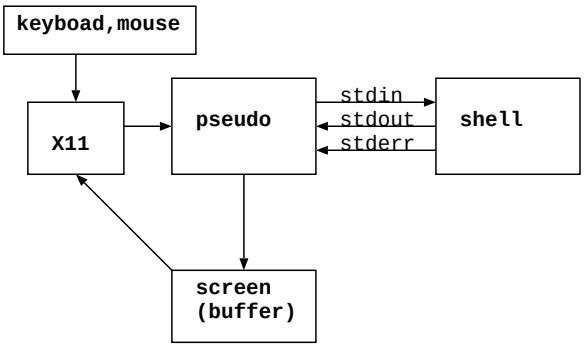

図 8: kterm の処理の流れ

kterm での vt100 エミュレーションは、screen と呼ばれる画面の状態を格納するバッファに対して行われる。screen に対する変更は、テキスト編集と同等なので、この操作を REP のプロトコルに変更して、TextEdit に伝えることになる。

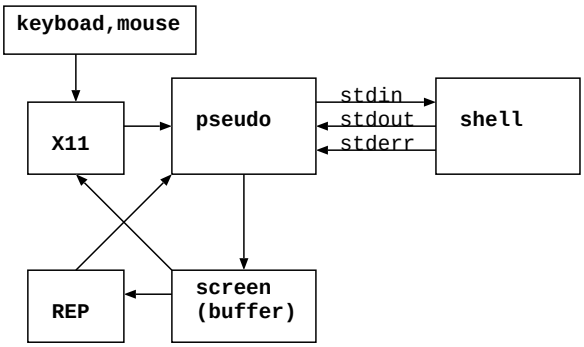

図 9: kterm への REP の実装

図 9 の REP としている部分は communicator,encoder-decoder で構成される。それぞれは以下のような処理を行う。

- encoder-decoder: kterm のスクリーンが更新された際に、その差分を REP に encode する。また、クライアントから渡される REP メッセージを decode し、kterm に処理を反映する。
- communicator: TCP/IP を用いたメッセージ送受信インターフェースであり、クライアントの接続の受け付け、ソケットの管理を行い、エディタ間のデータストリームを提供する。また、REP メッセージを encoder-decoder に渡すと共に、encoder-decoder から受け取るメッセージを接続先へ送る。

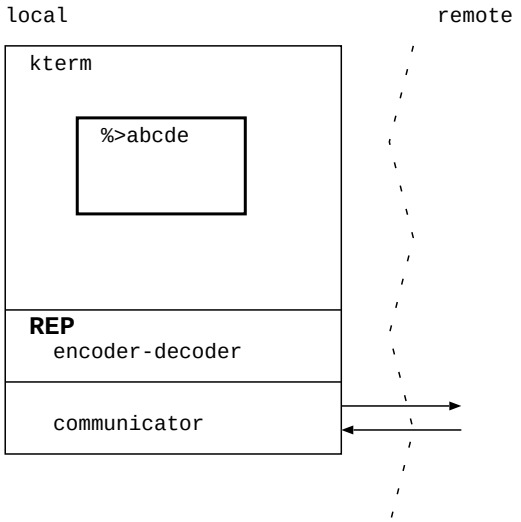

図 10: kterm におけるリモートエディタサーバ

リモートエディタサーバ kterm に対して、リモートエディタクライアント TextEdit を用いて接続し、TextEdit から kterm のターミナル機能を使用する。

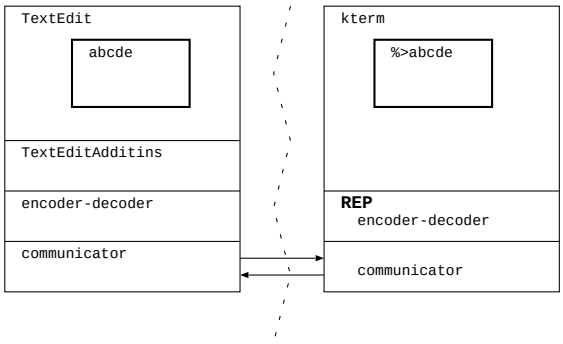

図 11: TextEdit と kterm の接続

kterm では、クライアントによって開かれるスクリーン全体のバッファとバッファ名、及びクライアントへ送信したバッファ領域へのポインタが管理される。リモートエディタクライアント TextEdit と、リモートエディタサーバ kterm を接続し、リモート編集を行う際の一般的な流れを以下のような手順となる。

1. (kterm) 起動
2. (TextEdit)kterm に接続要求
3. (kterm)TextEdit との接続を確立
4. (TextEdit)open コマンドを kterm に送信
5. (kterm)open コマンドが示すファイルに対応付けされたバッファを用意する
6. (TextEdit)read コマンドを kterm に送信
7. (kterm)read コマンドに対応する write コマンドを TextEdit に送信
8. (TextEdit) 編集が行われた際は、write コマンドを kterm に送信
9. (kterm)write コマンドに対応する処理をバッファに行い、場合によって結果を TextEdit に返す
10. (TextEdit)close コマンドを kterm に送信
11. (kterm) バッファを破棄し、TextEdit との接続を切断

4 ユーザ入力の扱い

ユーザの入力は、TextEdit と kterm という二つのアプリケーションがあるので、二通りの入力を行うことが可能である。

kterm で入力を受け取る場合は、X Window が必須であるが、これは、Mac OS X 上では、余計な Window システムを必要とするので、欠点となる。しかし、この場合は、TextEdit 側での入力を伝える必要がないので、実装としては容易になる。

TextEdit 上で入力を受ける場合は、さらに、以下の選択肢がある。

- TextEdit での編集したテキストの変更を REP 経由で送出する
- TextEdit に直接入力を受け、REP 経由で入力を送出する

最初の方法では、リアルタイムの入力を直接に実現するわけではないので、行単位の入力を実現することになる。この方法では、エコーバックのない入力や、割込みなどを、直接に入力することは出来ない。つまり、Emacs や vi などのリアルタイムのキー入力を取り扱うようなアプリケーションを使用することは出来ない。しかし、TextEdit 上での編集はスムースであり、より REP に向いた実現方法であると言える。

TextEdit を変更し、直接入力を REP で送る方法は、TextEdit への出力の反映が、kterm 経由となるために、遅延を生じるという欠点がある。また、TextEdit および、REP に対する若干の変更が必要となる。しかし、この方法の方が、より忠実な日本語端末エミュレーションの実現することができる。

5 考察

REP は、複数のエディタを相互に接続し、遠隔地にあるテキストファイルの効率的な編集を行うことができる。

GUI 上には、様々なアプリケーションがあるが、その多くがなんらかのテキスト入力や、テキストデータを対象としている。日本語端末などは、その典型的なものであるが、グラフィックスエディタなどでも文字入力だけでなく出力ファイルは、DXF や SVG などのテキストデータである。

アプリケーション間を接続するプロトコルは、Windows の DOC や、X Window の ICCCM などがあるが、Window システムに依存しているものである場合が多い。これは、cut and paste のように、GUI に依存したデータをやり取りする機能を持つからである。しかし、XML などでデータをやりとりする手法が互換性などを考えるとすぐれている。そこで、REP をアプリケーション間のプロトコルとして使用することが有効になると考えている。

XML の更新プロトコルとしては、SOAP が知られている。SOAP は、オブジェクトへのメッセージを XML でエンコードしたものであり、XML の変更自体は、オブジェクトが持つメソッドの能力に依存する。REP は、テキスト編集に特化した機能を持ち、SOAP と異なり、プロトコルの持つセマンティクスが明解である。それは、テキスト編集に他ならない。

kterm と TextEdit の接続は、このようなアプリケーション間の接続の一つの例となっている。

5.1 相互接続性

本稿ではkterm と TextEdit の接続を行ったが、これまで本研究ではEmacs,PICO,TextEdit,Sketch(Mac OS X 付属のドローツール),kterm 上でリモートエディタの実装を行ってきた。これらリモートエディタはそれぞれ接続先に関係なく相互に接続し編集を行う事ができる。

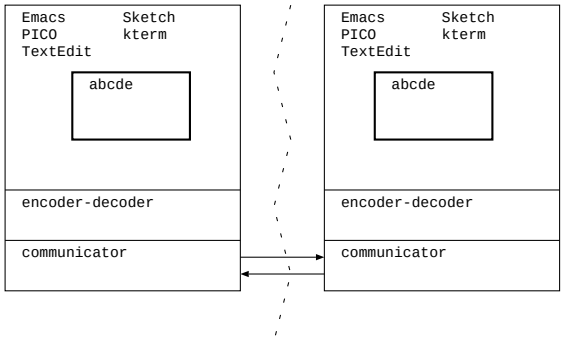

図 12: リモートエディタの特徴と利点

また、接続先によってプログラムを変更する必要も無い。TextEdit の代りに Emacs を用いて kterm に接続しても、TextEdit と同様に kterm のターミナル機能を使用する事ができる。kterm 側の実装は、種々の OS に依存する部分は皆無ではないが、そこには、GUI に依存する実装は必要ない。その部分は、接続するエディタ側に既に実装されている。エディタ側の変更は、比較的容易であり、実際、多数のエディタへの実装を行うことにより、それを示すことができている。

このリモートエディタの特徴は REP が read,write といった「編集に特化」した一歩踏み込んだプロトコルとして設計されているからである。それぞれのアプリケーションは自データ構造に対する read,write を定義すれば良い。これは REP の利点であり、リモートエディタの利点であると考える。

5.2 リモートエディタを実装することとは

リモートエディタは REP を用いてサーバ、クライアント間の通信を行う。そのために REP に依存したいくつかの機能を持つことが必須となる。ここでは、アプリケーションをリモートエディタ化することの必須条件、難しさについて述べる。

5.2.1 サーバ・クライアント間の通信

サーバエディタ・クライアントエディタ間では REP メッセージを運ぶために通信を行う必要がある。

5.2.2 REP の encode,decode

リモートエディタは「編集が行われた」といったイベントを捕らえ、それを REP メッセージに encode することができなければならない。また、送られて来る REP メッセージを、自身が処理できるように decode することができなければならない。

5.2.3 編集単位の切り分け

リモートエディタは、そのアプリケーションが持つ編集の対象となるデータ構造を編集単位で切り分けできなければならない。この編集単位というのは REP に依存したものであり、一般的にはアプリケーションが持つ編集の対象となるデータ構造を REP の通信単位に切り分けることができなければならない、ということになる。

まとめると、リモートエディタは以下の機能を持たなければならない。

- サーバ・クライアント間の通信
- REP の encode,decode
- REP の通信単位ごとの、自データ構造の切り分け

5.3 REP を用いるアプリケーションの設計

これまでは、既存のアプリケーションに REP を導入し、遠隔地のファイルを編集したり、複数のユーザから異なるユーザインタフェースを持つ機能などを付加してきた。

Unix では、複数のアプリケーションを組み合わせる方法として、stdin, stdout をパイプで接続するという方法が使われている。これは、すぐれた方法だが、大量のデータを取り扱う場合は、そのすべてを転送する必要があると言う欠点がある。また、GUI アプリケーションに対しては適当なインタフェースとは言えない。

GUI アプリケーションでは、そのような簡単な方法はなく、個々のアプリケーションによって、ドラッグ アンド ドロップなどのインタフェースが独自に設定されている。これらのインタフェースは、OS や Window System に依存しており、相互の接続は、NFS や CIFS、あるいは、WWW などのプロトコルを使用しなければならない。

SOAP やアップルスクリプトは、アプリケーション間を結ぶメソッドを提供しているが、その機能は、特別な機能を呼び出すというものであり、Unix のパイプのような、「任意の二つのアプリケーションを接続する」ようなものとは異なる。

REP は、「テキストを編集する」という汎用の機能を提供しており、そのテキストの中身に関して REP が知識を持つ必要はない。したがって、編集機能を提供するというインタフェースを提供するアプリケーションがあれば、そのアプリケーションに REP の機能を付加することは容易である。

また、意図的に、「テキスト編集による機能の制御」を持つアプリケーションを作成することにより、REP によって相互に接続可能なアプリケーション群を作成することができる。例えば、掲示版などは、WWW ページ自体に REP の機能を付加することにより自明に実現可能である。

REP 自体は、OS や Window System に依存しないので、そのようなアプリケーションを作成すれば、システム依存性の低いアプリケーションを作成することが出来る。

参考文献

[1] リモートエディティングプロトコルの Mac OS X のエディタへの応用 宮里忍, 河野真治 SWoPP 2001, July, 2001

[2] リモートエディタの実装とその XML への応用 新垣将史, 河野真治 日本ソフトウェア科学会第 16 回論文集,1999,pp293-296

[3] Emacs 上のリモートエディタ 新垣将史, 河野真治 電子情報通信学会技術研究報告,2000

[4] RemoteEditingProtocol を用いた複数ユーザ編集システム 新垣将史, 河野真治 日本ソフトウェア科学会第 17 回論文修,2000

[5] Apple Developers Connection "Cocoa Developer Documentation" http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html

[6] W3C "Simple Object Access Protocol (SOAP) 1.1" http://www.w3.org/TR/SOAP/

[7] Apple "AppleScript" http://www.apple.com/applescript

[8] David A.Curry 著、アスキー書籍編集部監訳 "UNIX C プログラミング" アスキー 1991.

[9] W.Richard Stevens 著、篠田陽一訳 "UNIX ネットワーキングプログラミング" トッパン 1992.