記 号 処 理 59－2
（1991. 3. 11）

Yy-server における入力機構について

田中啓介，古坂孝史，井田昌之
青山学院大学
情報科学研究センター 研究教育開発室

Common Lisp 用のウィンドウツールキット *YyonX* は，*Yy-server* と *Yy-client* の二つのプロセスで負荷分散を行ないつつ，ウィンドウ操作環境を提供する．キーボードやマウスを使ったユーザからの入力に対しても，*Yy-server* と *Yy-client* で分担して処理を行なう．本論文では，特に *Yy-server* での機能を中心として，*YyonX* における入力処理機構について述べる．

入力処理機構のうちキーボード入力に対応する処理については，X server からイベントとして送られてくるキー入力の情報を，*Yy-server* が解釈し，そのデータの画面へのエコーバックを行なう．この個々の情報は，アプリケーション側である *Yy-client* には送られない．但し，ある特定のキーが押下された場合には，それまでの入力情報を *Yy-client* にも通知する．*Yy-client* は，このキーの設定をアプリケーションに合わせて自由に設定できる．また，*Yy-server* は，キーに応じて入力編集コマンドを起動する機能を内部に保持している．さらに，外部のかな漢字変換サーバと通信を行なうことで，かな漢字変換機能を実現している．

An Input Handling Mechanism on *Yy-server*

Keisuke TANAKA, Takashi KOSAKA, Masayuki IDA
Computer Science Research Lab., Information Science Research Center,
Aoyama Gakuin University,
4-4-25 Shibuya, Shibuya-ku,
Tokyo, 150, Japan.

YyonX, which is a window tool kit for Common Lisp users, is designed based on a server-client model, and *Yy-server* and *Yy-client* share a load to provide a computing environment on windows. On windows, an input mechanism of a system receives some events from a keyboard or a pointing device. On *YyonX*, the input mechanism is divided into two parts; one is on *Yy-server* and another is on *Yy-client*. This paper describes the input handling mechanism on *YyonX*, and especially on functions for *Yy-server*.

For keyboard events, *Yy-server* receives events from X server, and echoes the data to the display. Each event from X server is processed only by *Yy-server* and is not send to *Yy-client*. *Yy-client*, or a Yy application, can designate several special keys. When one of these keys is pressed by a user, *Yy-server* sends the contents of its input buffer to *Yy-client*. The input mechanism on *Yy-server*, has an input-editing facility and KANA-Kanji conversion facility. The KANA-Kanji conversion is implemented on the communication between *Yy-server* and a server for a KANA-Kanji conversion.

1 はじめに

YyonX では，サーバ/クライアントモデルに基づいたGUI機能の提供を行なっている[1, 2]．クライアント部分 (*Yy-client*) は CLOS により記述されたアプリケーションを含んでいる．また，サーバ部分 (*Yy-server*) は X server と直接通信を行ない，X Window System の扱うことのできる資源を利用したウィンドウ利用環境を提供する．

Yy-server は *Yy-client* に対して「テリトリ」と呼ばれる領域に基づいたインタフェースを提供している．テリトリは，画面入出力の単位となる論理的には矩形の記憶領域である．テリトリには，画面上にマップされる状態とそうでない状態が存在する．画面上にマップされたテリトリは，X Window System の枠のない Window に対応する．また，この状態のテリトリは，X Window System におけるユーザからの入力をテリトリへの入力として受けつける．画面上にマップされていてもいなくても，テリトリへの文字・グラフィック出力は同様に可能である．*Yy-client* は，このようなテリトリを操作することで，ウィンドウ環境を提供する．このため，*YyonX* を使用するアプリケーションはベースとなるウィンドウシステムからは独立しており，また，X Window System 以外のウィンドウシステムの上で動作する *Yy* Window Tool Kit 上でも同様に動作が可能である．

YyonX ではウィンドウツールキットの機能をサーバとクライアントに分離し，かつその間にプロセス間通信を必要としている．そのため，プロセス間通信のタイミング及び通信量，あるいは，サーバとクライアントの役割分担は *YyonX* 全体の性能を決定する重要な要素である．特に入力処理，すなわち「ユーザによるキーボードやマウスからの入力に対して，適切な応答を返すための処理」における性能の良さは，良いユーザインタフェースを提供するための第一の条件である．

YyonX では，特にキーボード入力処理に注目し，その応答性能を高めるための機能分散を *Yy-server* と *Yy-client* の間に行なった．さらに，この入力処理機能の中で，*Yy-server* が *Yy-client* 以外のプロセスとの通信を通じてかな漢字変換を可能としている．以下では，*Yy-server* に行なわれた入力処理の概要と，かな漢字変換を可能とするための拡張について述べる．

2 *YyonX* における入力処理

2.1 ウィンドウツールキットにおける入力処理

ウィンドウツールキットにおける入力とは，アプリケーションの提供する画面上のウィンドウに対して，ユーザが入力用のデバイスであるマウスやキーボードを用いて行なう指示を指す．

一般的に，ウィンドウに対する入力としては，単発的な入力と連続的な入力が存在する．前者は，マウスボタンのクリックや指示位置の移動，割り込み等の単発的なキーの押下が含まれる．この入力は，それを起因として何かしらの処理がアプリケーション中で行なわれる．

一方後者は，ファイル名のような文字列を指定するための一連のキーの押下を意味する．この入力は，リターンキーなどのアプリケーションによって特別な意味を与えられたキーの押下や，あるいはアプリケーションの自発的な判断によって終了する．そして，一連の操作の結果がアプリケーションに渡され解釈される．入力の終了までのシーケンスはアプリケーションにとっては意味もなく，アプリケーションがその途中経過を知る必要はない．この形式の入力では，ユーザの入力操作を補助するために，押下したキーの内容が画面にエコーバックされることが多い．

これらの入力をウィンドウツールキットにおいて処理することを考えると，単発的な入力に対しては，ユーザの入力行動からアプリケーションにおいて対応した動作が開始されるまでの時間遅延が小さいことが望まれる．一方，連続的な入力に対しては，ユーザの入力行動に対する画面上での反応の早さが要求される．

2.2 *YyonX* における入力処理

YyonX ではウィンドウツールキットとして，単発的な入力と連続的な入力に対応している．アプリケーションは状況に応じて，単発的な入力を受けたり，連続的な入力を受けたり出来る．単発的な入力はイベントとして扱われ，アプリケーションでそのイベントを処理するための関数を定義すればそれが自動的に呼び出される．この種の入力としては，マウスの移動，ボタンの押下・開放，割り込みキーなどの特殊キーの押下がある．また，

連続的な入力はキーボードからの文字列入力を想定し，アプリケーションから `read` 等の関数を陽に呼び出すことで利用できる．

ところで，*YyonX* は X Window System 上で動作するウィンドウツールキットであるために，X Window System における入力が *YyonX* における入力となる．X Window System においてアプリケーションに対する入力はすべて「イベント」という形式で提供される．単発的な入力と連続的な入力は区別されず，すべて単発的な入力となる．そこで，連続的な入力は，ウィンドウツールキットやアプリケーションで適切な入力機構をもって対応しなくてはならない．したがって，*YyonX* では，1) X Window System のイベントを *YyonX* のイベントとしてアプリケーションに伝える機構，2) `read` 等の呼び出しにともない，一連のキー入力を読み出し結果を返す機構，という二種類の入力機構を持つことになる．

2.3 Lisp の入力機構としての特殊性

YyonX は Common Lisp 処理系を主たる対象としたウィンドウツールキットであり，入力にも Common Lisp の持つ会話的な性格を反映した処理が必要となる．その主な点としては，1) ')' に対応する '(', 対応する '"' などをユーザに知らせる，2) ')' の押下に対して，自動的にリストの終了を判断し，評価を開始する，3) スペースなどの押下に対して Completion を行なう，などが挙げられる．

これらの処理は，特に連続的な入力で必要となるが，ここでは，ユーザとの会話性能だけでなく，Lisp による入力の文脈の理解とそれに対する適切な表示処理が必要である．

3 *YyonX* Version1.0 における入力処理

図 1に *YyonX* Version1.0 における *Yy-server* の構造を示す．*Yy-server* と *Yy-client* の間には，a) コマンドやそれに対する戻り値を返すための通信路，b) 入力イベントを通知するための通信路，の二種類の通信路が存在する．

X Window System から *Yy-server* に通知されるイベントは，1) キーの押下・開放に関するイベント，2) マウスに関するイベント，3) ウィンドウの構成変化に関するイベント，がある．各イベントは X Window System 上のウィンドウに関連付けられて，発生・通知される．*Yy-server* は，このイベントの内容を解析し，対応するテリトリに振り分ける．三種のイベントの中で，ウィンドウの構成変化に関するイベントについては，テリトリ管理部で内部的に処理する．*Yy-server* は，テリトリの状態を変化したり，画面上のウィンドウの内容を再描画したりする．これらの処理は *Yy-client*, すなわちアプリケーションからは見えない部分で行なわれ，*Yy-server* と *Yy-client* との間で必要以上の通信を発生させない．

一方，キーやマウスに関するイベントは，対応するテリトリに割り振られたのちに，その内容がそのまま *Yy-client* に *Yy* におけるイベントとして通知される．すなわち，*Yy-server* だけでみると，すべての入力は単発的な入力であり，連続的な入力は存在しない．連続的な入力は，*Yy-server* からキーに関するイベントを受けた *Yy-client* が処理を行なう．

具体的には，*Yy-client* は，連続的な入力の処理を行なう場合には，1) *Yy-server* からの入力イベントを受けとり，2) その内容をウィンドウにエコーバックするため，*Yy-server* に対して，文字表示の命令を発行し，3) 文字列を保存するための内部のバッファを更新する，という手順をとる．

この方式では，入力処理を行なう中心が Lisp プロセスである *Yy-client* そのものにあることから，'(' と ')' の対応をとったり，Completion を行なったりという Lisp 言語依存の機能を実現しやすい．また，ユーザからの入力の受付，画面の状態，*Yy-client* の内部状態が完全に同期しており，アプリケーションの想定する画面状態が *Yy-server* と X server で保証されている．

しかし，反面，入力エコーバックに関しては，常に 1) X server から *Yy-server* への入力イベント通知，2) *Yy-server* から *Yy-client* への入力イベントの通知，3) *Yy-client* から *Yy-server* への文字出力コマンドの発行，4) *Yy-server* から X server への文字出力コマンドの発行，という四種類の通信を必要とする．このため，入力応答時間，すなわちキーを押下してから画面にその文字が表示されるまでの時間が，[5] による実測値では 103.74 msec となり，実用に向かない値となってしまう．

特にここで問題となるのは，通信そのものに費やされる時間ではなく，通信のための準備時間で

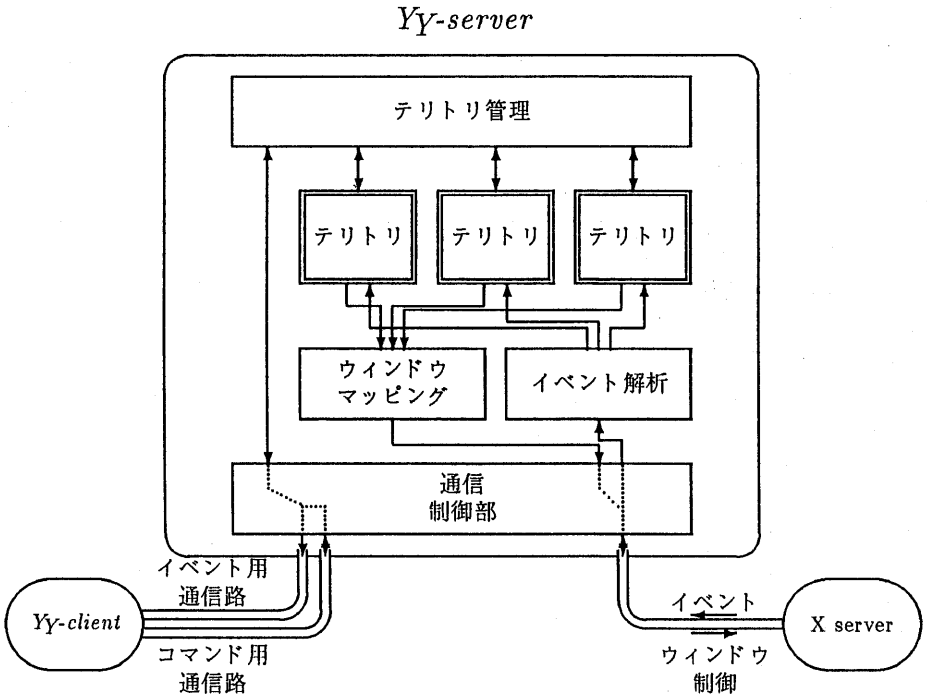

図 1: *YyonX* Version 1.0 における *Yy-server* の構成

あった．特に *Yy-client* が *Yy* でのプロトコルに必要なパケットを生成するための時間が多くの時間を占めることが明らかである [5]．これは，パケットを Lisp の文字列操作機能を用いている実装上の問題と共に，Lisp における文字列操作が必ずしも十分な速度で行なわれることがないことを示している．

4 入力処理における機能分散

4.1 *Yy-server* への Page モードテリトリの導入

YyonX Version 1.0 で採用した入力処理方式では，単発的な入力に対しての性能は，サーバ/クライアント間の通信の遅延に依存して決まる．そのため，Version1.0 の性能を飛躍的に改善する方法は存在しない．しかし，連続的な入力に対する性能は，*Yy-client* で行なっている処理の一部を，*Yy-server* 側で行なうことで改善が期待できる．連続的な入力においては，途中の過程は重要性を持たず，アプリケーションには最終結果だけが渡されればよい．したがって，連続的な入力の処理のほとんどを *Yy-server* 側で行なうことで性能の改善が達成できる．

YyonX Version 1.2 では，この入力処理における性能の向上のために，*Yy-server* に対して拡張を行なった．具体的には，「Page モードテリトリ」と呼ばれる特別なモードを持ったテリトリの設置で対応している．

Page モードテリトリは，*YyonX* の Page 実行モデル [3] に対応した概念であり，ユーザからのキーボード入力を中心として，対イベント反応性を高めたテリトリである．この Page モードテリトリとは別に，簡易アニメーション機能 [4] などの拡張グラフィック機能や背景のないテリトリの重ね合わせ機能 [4] などの複雑な出力機能を中心としてサポートするテリトリとして Viewport モードテリトリを設置した．両者の比較を表 1に示す．単発的な入力処理と Viewport モードテリトリにおける連続的な入力処理に関しては，Version1.2 でも Version1.0 のものをそのまま使用している．そこで，以下では Page モードテリトリにおける連続的な入力処理に対する機構の設計と実現につ

表 1: Page モードテリトリと Viewport モードテリトリ

	Page モードテリトリ	Viewport モードテリトリ
特徴	入力・会話処理中心	出力処理中心
グラフィック機能	基本機能のみ	基本機能と簡易アニメーションや背景のないテリトリの重ね合わせ
単発的な入力	*Yy-server* から *Yy-client* へ通知	*Yy-server* から *Yy-client* へ通知
連続的な入力	*Yy-server* でエコーバック処理，結果が *Yy-client* を通知	*Yy-server* から得る単発的な入力から *Yy-client* が処理

いて述べる．

Page モードテリトリにおける設計の第一点は，高い応答反応を実現することである．キーボードやマウスからの入力においては，ユーザは基本的にエコーバックなどの画面上での反応を頼りに入力を進める．そのため，十分なエコーバック性能が必要である．

また，この入力が，Lisp アプリケーションのための入力であることから，Lisp 言語に依存した処理が入力処理中に可能でなければならない．しかも，高い反応速度を維持しつつもアプリケーションの想定する画面の状態と，実際の画面の間での同期は適切なタイミングでとられなければならない．Page モードテリトリでの入力処理機構は以上のような項目を考慮してなされる必要がある．

4.2 Page モードテリトリでの入力データの扱い

先の要求から，Page モードテリトリでは，入力されたデータを *Yy-server* が *Yy-client* とは独立した独自の判断でエコーバックを行なうこととした．

しかし，*Yy-server* が完全に独立してエコーバックを行なうと，アプリケーションが *Yy-server* の状態や画面の状態をユーザからの入力が終了するまで把握できなくなる．これでは，'(' ')' 対応などの Lisp 言語に依存した処理が困難になる．そこで，*Yy-client* からの指示のもとに適切なタイミングで同期をとることを可能とする．すなわち，クライアントは，a) 入力終了キー，b) 入力中断キー，c) 入力途中経過報告キー，の三種のキーを指定でき，これらのうちのどれかが押下された時点で，a) 入力された結果をクライアントに通知して，入力処理を終了する，b) それまでの入力結果を破棄して，入力処理を終了する，c) それまでに入力された情報をクライアントに通知する，という処理を行なう．

4.3 Page モードテリトリでの入力機構の実現

実現した入力機構を含めた *Yy-server* の構造を図 2に示した．

Yy-client が想定する画面状態を破壊することのないように，Page モードテリトリ上での入力が指示された場合は，*Yy-server* 内に特別なエコーバック用の Page モードテリトリを作成する．このテリトリは *Yy-client* からは制御されず，*Yy-server* によって，入力されたキーの内容をエコーバックする目的でのみ使用される．

ユーザやアプリケーションからの **read** 関数などの呼び出しを受けて，*Yy-client* は *Yy-server* に対し，入力の開始を宣言する．この際，*Yy-client* は，入力の対象となる Page モードテリトリ，入力の開始位置，エコーバックに使用する文字フォントを指定する．また，入力終了，中断，途中経過報告のためのキーも指定する．これらのキーは複数を指定しても構わない．

入力開始を宣言された *Yy-server* は，指定された入力開始位置が含まれるように，入力エコーバックのためのテリトリを生成する．このテリトリは，入力対象となる Page モードテリトリのすべての属性を引き継いだ子のテリトリとなる．X server からみると，これは入力対象となるテリトリに対応した Window の上におかれた Window である．この Window の独立性は X server によって保証される．したがって，Window にどのような表示を行なおうとも，その下の Window，すなわち入

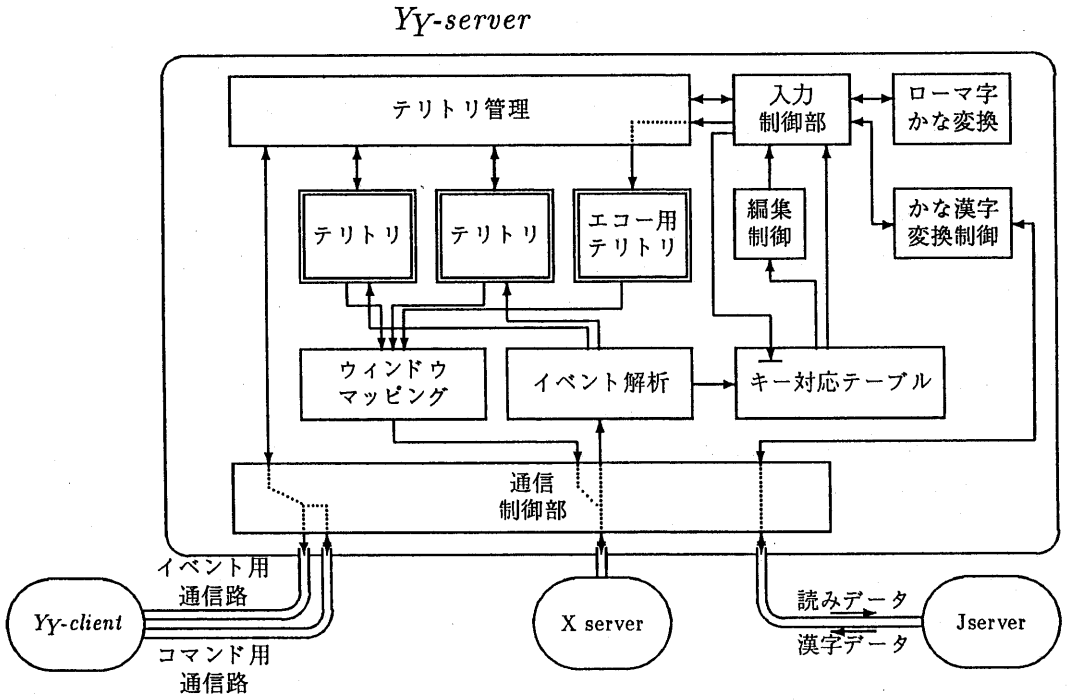

図 2: *YyonX* Version 1.2 における入力処理機構を含む *Yy-server*

力対象のテリトリに対応した Window の内容は破壊されない.

エコーバックのためのテリトリを作成した後, *Yy-server* は, ユーザからの入力された文字列を保存するためのバッファを作成し, ユーザからのキー入力にそなえる.

以後, X Window System によって検出されたキーの押下は *Yy-server* にイベントとして通知される. そこで, *Yy-server* は入力エコーバックのためのテリトリへ, 入力データを表示していく. この際, データ列が長くなり, それまでのエコーバック用のテリトリの大きさでは収まりきらなくなった場合には, このテリトリの拡張が試みられる.

エコーバック用のテリトリが拡張されると, その親, つまり *Yy-client* から入力対象として指定されたテリトリの大きさを越えることがある. これは, 画面に表示された Window がエコーされた文字で一杯になり, それ以上の文字表示のためにはスクロールを必要とするような状態である. *Yy-server* は, この場合にのみ, 入力対象テリトリを適切な大きさに変更し, その位置を変更し, スクロールを行なった状態にする. この変更は, *Yy-server* が行なうが, *Yy-client* の想定する画面状態に変化が生じることから, その変更の内容を *Yy* のイベントとして *Yy-client* に通知する. テリトリの変更自体は *Yy-server* の判断で, *Yy-server* と X server との間の通信によってのみ実現される. したがって, *Yy-client* への通知のための通信が入力処理の性能に影響することはない.

途中経過報告のためのキーの押下時は, そのデータを画面上に表示すると共に, それまでに入力された内容を *Yy-client* に対してイベントの形で通知する. この通知もデータのエコーバックとは独立して行なわれることから入力処理の性能に直接影響しない. *Yy-client* は, この通知を受けて, 入力の途中経過を理解し, 適切な処理を行なうことができる. たとえば, ')' の入力で, リストが終了したために, 入力処理を終了させる, あるいは, スペースキーが押下されたために Completion を行なうなどの処理を行なうことが可能である.

Yy-server は途中経過を通知した後も入力処理を続行する. この処理は, ユーザからの入力終了や入力中断キーの押下, あるいは *Yy-client* による入力処理終了の指示まで続けられる. *Yy-server*

は，ユーザにより入力終了キーが押下された場合，最初に *Yy-client* によって指示された入力対象テリトリに最終結果を表示して，エコーバック用テリトリを破壊する．これにより，入力対象テリトリには最終結果だけが表示され，その表示に伴う変更だけがなされる．その後，*Yy-server* は入力結果を *Yy-server* に対して，イベントの形で通知して入力処理を終了する．

4.4 入力編集機能

図2に示すように，*Yy-server* は入力編集用のキーテーブルを保持している．これにより，行頭・行末へのカーソル移動や，行消去・文字消去などの行単位での編集を可能とする機能を提供している．

これらの編集による画面上の変化は，すべて入力エコーバック用のテリトリの上の変更によるものである．そのため，複雑な編集作業を行なったとしても，入力対象となっているテリトリの内容を破壊することはなく，したがって，アプリケーションの想定する画面の状態を変更することはない．

現在，入力機構において標準的に提供している編集機能を表2に示した．キーの割り当ては，*Yy-server* 利用者が自由に設定できる．したがって，*Yy-client* のアプリケーションによらず，Page モードテリトリ上での入力は，一定のキー操作で編集が可能である．

5 入力機構へのかな漢字変換システムの組み込み

5.1 入力編集機能の拡張としてのかな漢字変換

YyonX の Page モードテリトリでの入力処理は，*Yy-client* に対して，フロントエンドプロセッサの役割を果たしている．このフロントエンドプロセッサは編集能力をも備えている．したがって，この編集機能を拡張することで，かな漢字変換の機能を入力処理機構に組み込むことができる．すなわち，編集キーのテーブルを，状態に応じて変更することで，ローマ字かな変換と，かな漢字変換の処理，およびその途中状態の表示が可能である．

入力のエコーバックが，すべて入力エコーバック用のテリトリを用いていることから，変換途中での画面表示の変更はすべてのこのテリトリ上で行なわれる．したがって，*Yy-client* の想定する画面が変換作業によって破壊されることはない．

表 2: *Yy-server* における入力編集機能

機能名 (標準キー割り当て)	機能
`beginning-of-line` (Ctrl-A)	行の先頭へカーソルを移動
`backward-char` (Ctrl-B)	前の文字へカーソルを移動
`delete-char` (Ctrl-D)	一文字消去
`end-of-line` (Ctrl-E)	行末へカーソルを移動
`forward-char` (Ctrl-F)	次の文字へカーソルを移動
`delete-backward-char` (Ctrl-H)	前の文字の消去
`kill-line` (Ctrl-U)	行消去

5.2 かな漢字変換サーバとの協調

かな漢字変換の辞書や変換アルゴリズムは *Yy-server* 内に組み込むのではなく，外部プロセスとして設定する．

このような方式をとった理由として，a) インタフェースが変化しなければ，*YyonX* の処理系とは独立して，かな漢字変換系を更新することができること，b) 他の非 *Yy* アプリケーションとの，辞書や頻度情報の共有が可能であり，変換アルゴリズムや使用感覚を一致させることができること，c) ネットワーク上での辞書や頻度情報の共有が可能であること，などがあった．

5.3 実現

図2に示すように，入力制御部は必要に応じて，キー対応テーブルを変更できる．これを操作して，a) 通常入力・編集モード，b) ローマ字かな変換モード，c) かな漢字変換モード，の三状態を切替える．今回は，かな漢字変換のための外部プロセスとして Wnn Jserver[7] を使った．

ローマ字かな変換モードにあっては，入力制御部はローマ字かな変換部を使用することで，ユーザが入力したローマ字列をかな文字列に変換する．具体的には，これは Wnn rom_kan ライブラリの関数呼び出しによって行なわれる．変換結果によって，入力エコーバック用テリトリの内容が変更され，画面上では適切な変換結果の表示がなされる．

また，かな漢字変換は Jserver とのプロセス間通信を通じて達成される．入力処理部は，それまでにかな文字列に変換されたものを漢字の読みとして Jserver に送り，その漢字表現を得る．そして，その漢字表現を，かな文字にかえて表示する．いくつかの候補が存在する場合は，標準的にはスペースキーの押下で次候補を表示する．また，文節の切れ目の変更等も可能である．これらの変換に関する機能は，キー対応テーブルに登録され，ユーザのキーの押下に対応して呼び出される．したがって，変換のためのキーの割り当ては，入力編集のキーの割り当てと同様に自由な設定が可能である．

かな漢字変換のための Wnn は Version 4.0.3 を用いた．全体としての使用感覚は，GNU-Emacs の日本語対応版 NEmacs のかな漢字変換インタフェースである Egg に近いものに設定してある．

6 おわりに

本入力機構を，*YyonX* Version1.2 において実現した．この機構の導入により，入力反応速度は，それまでの 103.74 msec から，実測値で 14.8 msec に改善された．この値は，[3] で設定された，目標値を満足するものであり，その意味で実用的であると判断できる．

この入力機構の特徴として，以下の二つの点が挙げられる．

- エコーバックは X server と *Yy-server* の間の通信によってのみ実現される．すなわち，その性能は *Yy-client* である Lisp プロセスの直接的な影響を受けない
- 入力の途中経過や最終結果の *Yy-client* への報告はキーの押下に同期して行われる．このキーは *Yy-client* によって任意のキーに設定できる．そのため，クライアントは入力時に押下されたキーに対応した処理を自由に発行することができる．実際に，*YyonX* においては，')' キー押下時のリストの終了の検出を可能としている．

この方式が，実用性を持つことは，サーバ/クライアント間の適切な負荷分散の有効性を示している．しかも，*Yy-server* が編集機能を持ち，そのためのテリトリを管理することからアプリケーションからの独立性を保ったまま自由な編集を可能とする．この機能はアプリケーション独立であり，異なる *Yy* のアプリケーションで共通に使用できる．

しかし，サーバ/クライアントモデルは，プロセス間通信を必要とする点において，通信のオーバーヘッドという避けられない問題を抱えている．したがって，今後，もっと効率的なプロセス間通信の方式を検討する必要がある．高速伝送媒体の出現により，通信速度では満足する性能を示すものの，通信のための前処理，後処理には改善の余地がみられる．*YyonX* でも特に Lisp プロセス側のパケット作成，パケット解析のための処理時間は無視できない程度の大きさになっている．この点についての改善が必要である．また，この実装上の課題として，Lisp における標準プロセス間通信インタフェースの構築が残されている．

参考文献

[1] Masayuki Ida, et.al.: "An Overview of *Yy* and *YyonX* – A CLOS based Window Tool Kit and its implementation –," Proc. of Europal'90, pp. 245-252. Mar. 1990.

[2] 井田昌之，他: *YyonX*, 情報処理学会第 40 回全国大会, 3J-2, 3J-3, 3J-4, Mar. 1990.

[3] 井田昌之，他: "*YyonX* 実行モデル (1) – 機能分散に基づく設計 –", 情報処理学会第 41 回全国大会, Sept. 1990.

[4] 太田幸雄，他: "*Yy* へのカラーグラフィック機能の組み込みについて", 情報処理学会第 60 回記号処理研究会, Jan. 1991.

[5] 古坂孝史，他: "*YyonX* 実行モデル (2) – *Yy-client* の出力の解析 –", 情報処理学会第 41 回全国大会, Sept. 1990.

[6] 田中啓介，他: "*YyonX* 実行モデル (3) – ウィンドウ入出力機構 –", 情報処理学会第 41 回全国大会, Sept. 1990.

[7] 鈴木隆，他: "Wnn 日本語入力システムのかな漢字変換について", 日本ソフトウェア科学会第 4 回全国大会, D-1-2, 1987.

[8] 戸村哲，他: "Nemacs 組み込み型日本語入力システム「たまご」について", 第 2 回 GMW+Wnn ユーザー連絡会資料, 1988.