プログラムの部分移送に基づく遠隔実行機構とその知的インタフェースへの応用

Remote Execution by Partial Program Transfer and its Application to Intelligent Interfaces

倉林 則之[†] 湯淺 太一[‡] 小宮 常康[‡]
Noriyuki Kurabayashi Taiichi Yuasa Tsuneyasu Komiya

概 要

本論文では，Java VM 上で動作する Scheme 処理系ぶぶにおける，遠隔実行機構の導入とその応用について述べる．プログラムの部分移送を伴う遠隔実行は，ソケット通信や遠隔手続き呼び出しとともに開放型分散環境におけるシステムプログラミングをサポートするための言語機構として利用できるが，ここでは特に，プログラムの移動が本質的な役割を果たす，移動可能な知的インタフェースに焦点を当てて，遠隔実行の応用について議論する．我々の遠隔実行機構は，指定された部分を本体とする関数閉包のコピーによって実現されている．遠隔実行は同期型で行なわれるが，この間，接続は維持する必要がない．関数閉包の環境は実行結果とともに呼び出し側に返される．このため，必要な部分のみを効率的に移送できるだけでなく，通信環境の変化に対しても頑強である．具体的な応用例として，URL を選択して Web ブラウザを起動するアプリケーションを作成した．このアプリケーションは，閲覧した URL と時刻を事例として学習し，過去の最も近い時刻に閲覧された URL を候補として推薦する，予測インタフェースを実現している．事例はサーバに蓄積され，GUI を実装しているクライアント上で学習機構が遠隔実行される．新たに追加された事例をサーバに保存することで，クライアント間で学習結果を共有することが可能となる．このアプリケーションの実装を通して，我々の遠隔実行機構の有効性を示す．

1 はじめに

情報機器のネットワーク化や携帯性の向上によって，情報システムの利用形態が大きく変化している．システムには時間や場所の制約を受けない柔軟性や自由度が求められ，状況に依存した適切なサービスの提供や各種のユーザ支援のために移動性を持つことが必要になると予想される．Bharat ら [3] は，インタフェースを持つシステムの移動が有用な応用として，ユビキタスコンピューティング，電子秘書，対話型エージェントなどを挙げている．特に，ユビキタスコンピューティングは実世界指向インタフェース [13] とも関連が深く，その適用範囲は広い．ユーザ支援の観点から，端末が遍在する環境においてインタフェースを持つシステムが移動することには，次のような利点がある．

- 任意の端末を動的に個人化することが可能となり，端末を持ち歩く必要がなくなる
- その場所にある計算資源を有効に活用できる
- 常に同一のインタフェースを用いてシステムと対話するため，端末の操作感に一貫性を持たせることができる

一方で，システムの高機能化や多機能化に起因する使用性（usability）の低下も指摘されており，主観的な複雑さや操作の煩雑さを軽減するとともに，ユーザの多様性にも対応できる使い勝手の良いインタフェースが望まれている．知的インタフェースは，状況やタスクに適合した積極的な操作支援によって使用性を向上するインタフェースであり，このような要求を満足しうるものである．例えば，知的インタフェースの一種である予測インタフェース [6, 15] では，ユーザの次の操作を予測することで操作効率を改善する．また，多くの知的インタフェースでは，学習によってユーザ特性を獲得して支援に利用することが多く，継続的な使用によってユーザに適応することが可能である．

本研究では，端末が遍在する環境において，移動性を持つ知的インタフェースを実現するための基盤として，Java VM 上で動作する Scheme 処理系ぶぶに遠隔実行機構を実現した．これは，Scheme プログラムの指定した部分のみを遠隔ホスト上で実行する部分移送に基づいており，実行状態を保存したままで転送されたプログラムを遠隔実行するものである．また，具体的な応用例として，予測インタフェースを実現した Web ブラウザのランチャーを作成することで，本研究の遠隔実行機構を用いて知的インタフェースの移動性を実現できることを確認した．なお以降では，遠隔実行のために転送されるプログラムを**部分プログラム**と呼ぶ．

以下に，論文の構成を示す．2 節では，知的インタフェースの移動性について考察する．3 節では，ベー

[†]エイ・ティ・アール環境適応通信研究所
ATR Adaptive Communications Research Labs.
[‡]京都大学大学院 情報学研究科
Graduate School of Informatics, Kyoto University

スシステムである**ぶぶ**の概要を述べる．4 節および 5 節では，本研究で実現した遠隔実行機構とその実装について説明する．応用例として作成した知的インタフェースの詳細については 6 節で述べる．7 節では考察を行ない，8 節で関連研究について触れる．最後に，9 節でまとめる．

2　知的インタフェースの移動性

本研究では，端末が遍在する環境を想定している．ここで端末はネットワークに接続可能（ただし常時接続とは限らない）であり，複数のユーザによって共有されるものとする．このような環境において，学習型の知的インタフェースに移動性を実現するには，次の点に考慮しなければならない．

端末特性の違い 特性が異なる端末間の移動を実現するには，特定の端末に依存しない振る舞いの定義が必要である．

永続性 状況やタスクに応じて適切なユーザ支援を行なうには継続的な学習が重要であり，獲得したユーザ特性情報は永続的に保持しなければならない．

計算資源の有効利用 端末を使用する機会を他のユーザにも提供するために，サービスの利用が終了したら計算資源を直ちに解放すべきである．

転送量の抑制 携帯端末などの小型機器を利用できるように，移動の範囲を指定できる必要がある．

インタフェースの表現は端末特性に依存することから，その低水準の実装は端末上に配置するのが適当である．一方，学習により獲得したユーザ特性情報は学習機構によって管理されるべきものである．そこで本研究では，知的インタフェースを学習機構とインタフェースの実装に分離して扱う．移動性を持つのは学習機構のみである．端末はネットワークに常時接続されるとは限らないため，学習機構はサーバ上に用意して任意の端末からアクセス可能とする．すなわち，ユーザからの要求によって，学習機構をユーザが使用している端末へ転送して実行することで，知的インタフェースの移動性を実現する．また，サービスの利用が終了（端末上で学習機構の実行が終了）した時点で，更新されたユーザ特性情報をサーバ側に書き戻すことで，ユーザ特性の永続性を保証する．学習機構は値を返すため，実行形態は遠隔実行となる．

3　ぶぶの概要

ぶぶ [12] は Java VM 上で動作する Scheme 処理系であり，次のような特徴を持つ．

- IEEE P1178[7] 準拠

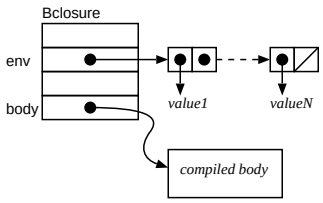

図 1: 関数閉包の構造

- Java との完全な言語透過性
  - 例外処理機構
  - オブジェクトシステム
- 遠隔オブジェクト
- Common Lisp ライクなマクロ

また，**ぶぶ**はバイトコードインタプリタ方式を採用しており，ユーザ定義関数は図 1 に示す関数閉包（`Bclosure`；Byte code closure）にコンパイルされる．図において，`env` は関数閉包の環境，`body` は関数本体のバイトコードを指す．関数閉包の環境はリスト構造で表現される．

例外処理機構およびオブジェクトシステムは Java インタフェースを用いて実装されており，Java クラスへの透過的なアクセスを実現している．遠隔オブジェクトは ORB（Object Request Broker）[11, 17] に相当する機能を提供しており，遠隔ホスト上でのオブジェクト生成と遠隔参照が可能である．

4　遠隔実行機構

4.1　実行形態

本研究では，部分プログラムが遠隔実行される場所を**プレース**と呼び，**ぶぶ**処理系と区別している．遠隔実行は同期型で行なわれ，実行中は呼び出し側はブロックされる[1]．すなわち，遠隔実行は**ぶぶ**–プレース間の制御の移動で表現され，通常の式と同様に扱うことができる．部分プログラムの局所変数は定義時の環境を参照するが，自由変数はプレース環境を参照する．従って，移動先（プレース）の情報を積極的に利用するような部分プログラムも容易に記述できる．

4.2　プレース記述子

プレースは，それが動作しているホスト名と，部分プログラムや実行結果の転送に使用されるポート番号によって一意に識別できる．プログラム中では，これらのリストである**プレース記述子**（*place descriptor*）

(*hostname port#*)

を使ってプレースを特定する．

[1] **ぶぶ**は future ベースの並行処理機構を提供しており，これを用いることで**ぶぶ**とプレースの並行動作も可能である．

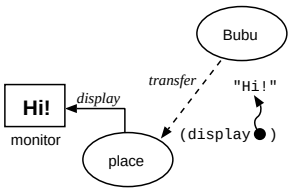

図 2: 遠隔実行の例

4.3 遠隔実行のための形式

遠隔実行のための形式は次の 2 種類である．

- (**rexec** *pdesc func*) **関数**
  プレース記述子 *pdesc* で指定されたプレース上で *func* を実行する．*func* は無引数の関数でなければならない．*func* 本体の自由変数はプレース環境を参照する．*func* の返す値が **rexec** 関数の値となる．

- (**remote** *pdesc body*) **マクロ**
  プレース記述子 *pdesc* で指定されたプレース上で *body* を実行する．*body* はフォームの並びである．*body* の自由変数はプレース環境を参照する．*body* の最後のフォームの値が **remote** マクロの値となる．

例えば，

```
(let ((msg "Hi!"))
  (remote ("robin" 8140)
    (display msg)))
```

は，プレース記述子 **("robin" 8140)** で指定されるプレース上で式 **(display msg)** を実行する．局所変数 **msg** は文字列**"Hi!"**に束縛されており，プレース上には「**Hi!**」と表示される．ここで **display** は自由変数であり，プレース環境を参照する．このため，実際の表示形態はプレースにおける **display** 関数の実装に依存しており，図 2 に示すようになる．また，この式は

```
(let ((msg "Hi!"))
  (rexec (list "robin" 8140)
    (lambda () (display msg))))
```

と展開される．

5 実装

5.1 部分プログラムの転送

rexec 関数は，第 1 引数のプレース記述子で指定されたプレースに，第 2 引数で与えられた関数閉包のコピーを転送することで遠隔実行を行なう．関数閉包は図 1 に示した構造を持つ．**rexec** 関数によって転送される部分プログラムはすべて関数閉包であり，

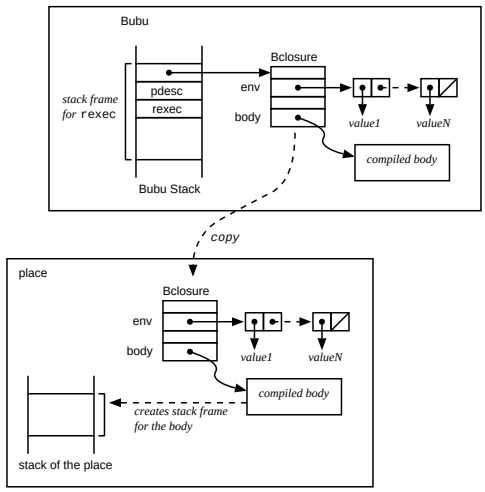

図 3: **rexec** 関数の実行

これが転送の単位となる．遠隔実行のためにスタックのコピーは行なわれず，プレース側で実行時にこの関数閉包を呼び出すためのスタックフレームを構築する（図 3）．従って，部分プログラムは，本体を実行するためにプレースのスタックを使用する．

関数閉包の転送は Java の Object Serialization[10] を利用している．**ぶぶ** の関数閉包は通常の Java クラスであり，関数閉包の環境や関数本体のバイトコードへの参照を保存した状態で転送（marshaling）される．オブジェクトはコピーされるが，記号は印字名を転送してプレース側でインターンする．これにより，4.1 節で述べたように，部分プログラムの自由変数がプレース環境を正しく参照する．記号（変数）の値は転送されない．

5.2 関数閉包の環境の更新

部分プログラムで局所変数への代入が行なわれる場合，呼び出し側にもこの変更が反映される必要がある．従って，遠隔実行の終了時に実行結果と関数閉包の環境を呼び出し側に転送し，変更を書き戻さなければならない．

5.3 接続の管理

部分プログラムの転送にはストリーム型ソケットを利用しており，通信には明示的な接続の確立が必要である．ただし，実行に長い時間を要する部分プログラムにおいては，接続の維持は適当でない場合があるため，部分プログラムの転送終了後に一旦接続を切断し，実行終了時に再接続を行なう．また，遠隔実行の時間を制限できるように，タイムアウトを設定可能にしている．接続管理を行なうには，プレース記述子の拡張された形式を用いる．

(*hostname port#* [*timeout* [*keepconn*]])

timeout は遠隔実行のタイムアウトまでの時間（単位はミリ秒）であり，*keepconn* は再接続を行なうか

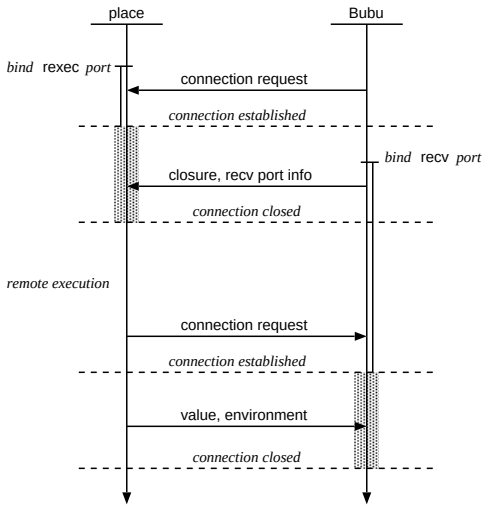

図 4: 再接続手順

どうかの真偽値である．タイムアウトを設定した場合，その時間以内に遠隔実行が終了しないと例外 `bubu.cond.TimeoutException` が発生する．*keepconn* が真ならば再接続処理が行なわれる．

図 4 に再接続手順の詳細を示す．ここで，`rexec` ポートとは部分プログラムを受信するためにプレースが使用するポートであり，`recv` ポートとは遠隔実行の結果を受信するために **ぶぶ** が使用するポートである．二重線の部分は接続要求待ち状態を，灰色の部分は接続が確立した状態を表している．従って，部分プログラムの実行中にプレースをネットワークから切り離すことも可能である．

5.4 評価

実装した遠隔実行機構を評価するために，3 つのプログラムを用いて性能を測定した．動作環境は，100Mbps Ethernet で接続された 2 台の Sun Enterprise 3000（**ぶぶ** 側は UltraSPARC 168MHz，プレース側は UltraSPARC-II 248MHz，OS はどちらも Solaris 2.6），JDK 1.1.7 である．測定に使用したプログラムは次のようなものである（定義は付録 A に示した）．

`empty`	本体が空の関数の実行
`fib`	フィボナッチ数を計算する関数の定義と実行
`trlis`	1,000 個の整数のリストから，正数のみ（500 個ある）を取り出してリストを作成

それぞれの実行時間を表 1（単位はミリ秒）に，データ転送量を表 2 に示す．

`empty` によって遠隔実行のオーバーヘッドが分かる．これは，関数閉包の構造（図 1）の転送や **ぶぶ** – プレース間のポートの準備に要する時間である．`fib` は関数の定義を転送する．これは部分プログラムの本体の局所関数として読み込み時にコンパイルされるため，遠隔実行時にコンパイルは行なわれずオーバーヘッドが小さい．`trlis` は部分プログラムの関数閉包の環境（すなわち局所変数）が長さ 1,000 のリストを持ち，プログラムの実行よりも転送時間が問題となる例である．関数閉包の環境は遠隔実行終了時に書き戻されるため，転送コストが高い．

表 1: 実行時間

プログラム	通常の実行	遠隔実行
`empty`	—	144.3
`fib`	10893.3	11353.0 (104%)
`trlis`	614.7	3401.0 (553%)

表 2: 転送量

プログラム	送信 (bytes)	受信 (bytes)
`empty`	1,371	68
`fib`	2,082	175
`trlis`	13,094	17,328

6 応用例

本節では，移動性を持つ簡単な知的インタフェースとその実装について述べ，本研究の遠隔実行の具体的な応用例を示す．このインタフェースを PWB（Predictive Web Bookmark）と呼ぶことにする．PWB は，リストから選択した URL のサイトを表示するための Web ブラウザのランチャーであり，ユーザの閲覧履歴を学習して URL を予測する機能を実現している．図 5 は PWB の画面の例である．PWB の学習手法は URL と閲覧時刻の組からなる事例の記憶[2]に基づいており，現在時刻に最も近い閲覧時刻を持つ事例の URL を閲覧の候補として提示する．

2 節で述べたように，PWB の移動性は学習機構

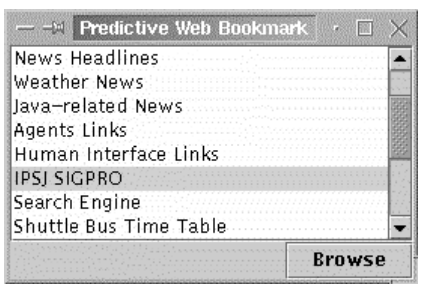

図 5: PWB の画面

[2]事例に基づく学習（IBL; Instance-Based Learning）[2] と呼ばれる．

の遠隔実行によって実現することができる．図 6 に PWB の構成を示す．wrapper および event については，6.1 節で，profile は 6.3 節で述べる．

6.1 インタフェースの具体化と制御

端末上で実装されたインタフェースを利用する場合，学習機構の端末への依存性を排除するために実装の抽象化が必要である．PWB では，インタフェースの具体化（instantiation）および制御を行なう wrapper を端末側に導入することで，学習機構からのインタフェースの統一的な扱いを実現している．

インタフェースの制御はイベント駆動方式で行なう．イベントに対する振る舞いの定義は wrapper に記述されており，wrapper にイベントを送ることで PWB を制御することができる．現在の実装では表 3 に示した 9 種類のイベントを定義しており，図 6 の wrapper と，学習機構（learning engine）およびインタフェース（interface）の間の通信に使用している．例えば，インタフェースは URL を閲覧するために wrapper に対して **browse** イベントを発行する．wrapper の **browse** イベントハンドラは Web ブラウザを起動するとともに，これを事例として学習するために自分自身に対して **learn** イベントを発行する．wrapper の **learn** イベントハンドラは URL と閲覧時刻の組である事例を学習する．以下，本節では，この例に沿って説明する．**learn** イベントハンドラの定義は 6.3 節で与える．

GUI のボタンのクリックにより **browse** イベントを発生するには，ぶぶ オブジェクトシステムを用いて，次のような Listener クラスを定義する（Java AWT による実装の場合）．

```
(defclass BrowseListener
  (:implements ActionListener)
  (:method actionPerformed
           (void (evt ActionEvent))
    ((active-wrapper)
     'browse
     (get-target-url))))
```

`actionPerformed` メソッドは，`active-wrapper` 関数の値である現在の wrapper に対して，閲覧対象となる URL を引数とする **browse** イベントを発行する．

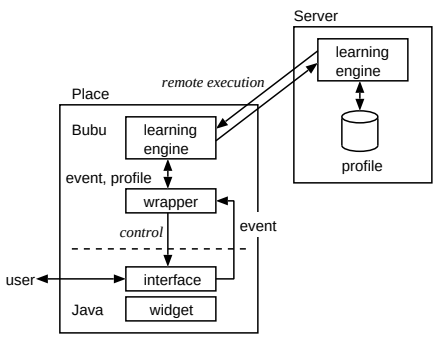

図 6: PWB アプリケーション

表 3: PWB のイベント

イベント	要求される動作
open	セッション開始
close	セッション終了
add	URL の追加
select	URL の選択
target	現在の閲覧候補
predict	URL の予測
learn	事例の学習
browse	ブラウザ起動
finalize	終了処理

Listener が発行する **browse** イベントのハンドラは次のように定義される．

```
(define-handler w 'browse
  (lambda (url)
    (launch-browser url)
    (w 'learn url)))
```

この式は，変数 `w` が参照する wrapper に **browse** イベントハンドラを登録する．ハンドラの本体は，引数の URL を閲覧するために Web ブラウザを起動するとともに，これを学習するために **learn** イベントを自分自身に対して発行する．Web ブラウザの起動は `launch-browser` 関数により行なわれる．

wrapper は，次に示す `make-wrapper` 関数により作成することができる．

```
(define (make-wrapper)
  (let ((pwb (new ListSelector))
        (wrapper
         (lambda (event . args)
           (apply (get-handler event)
                  args))))
    (send pwb addBrowseListener
          (new BrowseListener))
    (define-handler wrapper 'browse
      (lambda (url)
        (launch-browser url)
        (wrapper 'learn url)))
    ;; その他の初期化
    wrapper))
```

`ListSelector` は Java で記述されたクラスであり，wrapper は Scheme からイベントによってこのクラスを利用するために使用される．イベントハンドラは wrapper が管理しており，`get-handler` 関数でこれを検索する．`make-wrapper` 関数はまた，`ListSelector` クラスの Listener として `BrowseListener` を登録する．これにより，PWB の GUI（図 5）のボタンに動作を与える．PWB と

ListSelector クラスの関係については 6.2 節で述べる．

6.2 プレースの実装

プレースではインタフェースおよびその wrapper を実装する．PWB は，図 5 に示した GUI を提供するが，これは ListSelector クラスを wrapper で制御することによって実現している．ListSelector クラスは図 7 に示すクラス構造を持ち，wrapper によって PWBListener および BrowseListener クラスと関連づけられている．つまり，PWB の（静的な）視覚的表現は Java で，（動的な）振る舞いは Scheme で記述している．なお，図 7 に示した JList，DefaultListModel，JButton は，いずれも JFC/Swing のクラスである．

6.3 サーバの実装

サーバでは学習機構を実装する．学習機構は PWB の本体であり，URL のリストおよびユーザの閲覧履歴を管理する．閲覧履歴は URL と閲覧時刻の組からなる事例の集合によって表現されており，これをプロファイル（profile）と呼ぶ．プロファイルは閲覧傾向を表現する，ユーザ特性である．事例間の「距離」は時刻の差によって定義され，最も近い事例を選択することで URL の予測を実現している．

学習機構は，予測や学習の実行とプロファイルの操作に分離することができる．予測は端末上で遠隔実行され，インタフェースの制御を伴う．プロファイル操作は，予測を行なう部分プログラムが参照するプロファイルを管理する．従って，PWB の本体は次のように定義される．

```
(define (make-pwb pdesc sites profile)
  (remote pdesc
    (let ((wrapper (make-wrapper)))
      (add-sites wrapper sites)
      ;; イベントハンドラの定義
      ;; （下記参照）
      (wrapper 'open)
      (event-loop)))
  (update-profile profile))
```

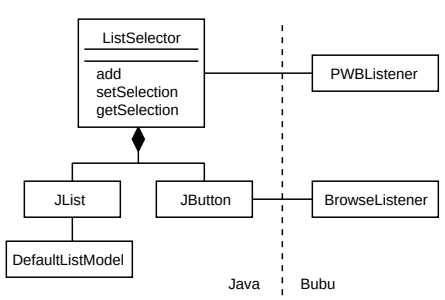

図 7: GUI のクラス図

remote 式の本体では wrapper を作成し，add-sites 関数によって URL を登録している．また，予測や学習を実行するために，イベントハンドラの定義も行なう．これらは本節の後半で述べる．遠隔実行終了後，update-profile 関数によりプロファイルを更新（操作）する．ここで，変数 profile により参照されるプロファイルは，新しい事例を学習するために遠隔実行中に変更される可能性がある．5.2 節で述べたように，profile は remote 式により作成される関数閉包にクローズされるため，update-profile 関数は変更されたプロファイルを正しく参照することができる．

予測および学習は，それぞれ PWB の開始時および終了時に実行するため，イベントハンドラとして実装する（定義は make-pwb 関数の remote 式の本体に記述する）．

```
(define-handler wrapper 'predict
  (lambda ()
    (wrapper 'select
     (nearest-case profile))))
(define-handler wrapper 'learn
  (lambda (url)
    (let* ((time (get-time))
           (c (make-case url time)))
      (set! profile
            (cons c profile)))))
```

predict イベントハンドラは，nearest-case 関数を用いて最も近い事例を検索し，その URL をユーザに提示するために **select** イベントを発行する．URL の提示はそれを GUI 上で選択状態にすることで行なう．**learn** イベントハンドラは新しい事例を学習する．学習は make-case 関数を用いて現在時刻と URL の組から新しい事例を作成し，変数 profile の先頭に追加することにより行なう．

6.4 視覚的表現の変更

ユーザが複数の端末を使い分ける場合，端末の種類は異なっていることが多い．特に，表示領域に大きな制限を持つ携帯端末等の機器においては，GUI の利用は現実的とはいえず，代替的なインタフェース方式が望まれる．本節では，インタフェースをテキスト形式に置き換えたプレースの実装例を示す．

```
(define-handler wrapper 'open
  (lambda ()
    (wrapper 'listing)))
(define-handler wrapper 'input
  (lambda ()
    (let* ((num (user-input "URL#>"))
           (url (nth-url num)))
      (wrapper 'browse url)
      (wrapper 'learn url))))
```

ここでは，URL のリストをテキスト形式で出力するイベント **listing**（定義は省略する）と，番号で URL

図 8: テキスト形式の表現

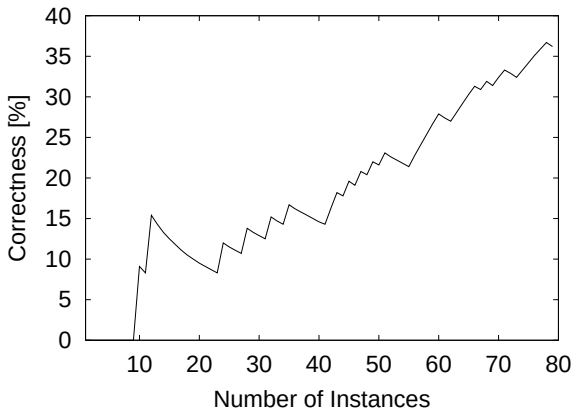

図 9: PWB の学習曲線

を選択するためのイベント **input** を新たに導入している．

イベントループは次の駆動ループで実現する．

```
(define (event-loop)
  (let ((wrapper (active-wrapper)))
    (cond (wrapper
           (wrapper 'predict)
           (wrapper 'input)
           (event-loop)))))
```

図 8 は，テキスト形式のインタフェースを実装した PWB の画面例である．

6.5 PWB の評価

6.5.1 操作効率

PWB は，URL の予測によってサイトの選択における操作効率の向上を図っている．ここでは，キー打鍵レベルでタスク遂行時間を推定するための評価モデルである KLM（Keystroke-Level Model）[4] によって，予測の効果を見積もる．KLM では，表 4 に示す各オペレータの実行回数の総和からタスク遂行時間を推定する．表 4 に示したオペレータの実行時間は標準的な値を採用しているが，直線の描画は URL 選択のタスクに含まれないため省略する．また，簡単のため，システムの応答速度は 0（無視）とした．

表 4: KLM オペレータ

オペレータ	意味	時間 [sec]
K	キー打鍵，クリック	0.20
P	マウスによるポイント	1.10
H	手の移動	0.40
D	直線の描画	—
M	精神的準備	1.35
R	システムの応答速度	0

図 5 の GUI を操作する場合，PWB の予測が正しければ，ユーザは閲覧ボタンを押下するだけで良い．すなわち，閲覧ボタンをポイント（**P**）してからクリック（**K**）する．精神的準備時間（**M**）は，操作の選択に要する時間を代表するオペレータであり，原則として一連の操作系列の前に挿入される．

$$\mathbf{MPK} = 1.35 + 1.10 + 0.20 = 2.65[\mathrm{sec}]$$

一方，予測と異なるサイトを閲覧する場合は，URL を選択してから閲覧ボタンを押下する必要がある．

$$\begin{aligned}\mathbf{MPKPK} &= 1.35 + 2 \times 1.10 + 2 \times 0.20 \\ &= 3.95[\mathrm{sec}]\end{aligned}$$

従って，予測によって操作手数を約 33%削減できると考えられる．

テキスト形式も同様である．予測が正しいときは Enter キーの押下のみで良い．

$$\mathbf{MK} = 1.35 + 0.20 = 1.55[\mathrm{sec}]$$

異なるサイトを閲覧するには番号（ここでは 2 桁と仮定）を入力してから Enter キーを押下する．

$$\begin{aligned}\mathbf{M2KK} &= 1.35 + 2 \times 0.20 + 0.20 \\ &= 1.95[\mathrm{sec}]\end{aligned}$$

この場合，約 21%の手数削減となる．

6.5.2 予測の精度

PWB の学習手法は事例に基づく学習である．時刻を属性としており，時間帯毎にユーザの閲覧傾向を学習することができる．PWB の予測の精度を把握するために，被験者 1 名に対して簡単な実験を行なった．図 9 はその学習曲線である．

URL のリストは，被験者が使用している Web ブラウザの閲覧履歴から抽出した 73 個のサイトから構成した．事例を 80 個獲得した時点で，予測の精度は 36.2%であった．

被験者は，GUIとテキスト形式のインタフェースでPWBを使用した．学習曲線が示すように，予測の精度は緩やかに向上しており，継続的な学習が行なわれたことが分かる．

7 考察

本研究の遠隔実行機構は，5節で述べたように，関数閉包の転送によって実現している．部分プログラムは関数閉包によってカプセル化されるため実現が容易であり，プレースの実行環境の軽量化が期待できる．関数閉包をエージェントのテンプレートに，部分プログラムの本体をエントリポイントに対応づければ，この方式は移動エージェント[1, 9]に類似しているとみなすこともできる．遠隔実行では，移動エージェントに見られる巡回の機能はない[3]が，局所変数（関数閉包の環境）を用いたデータの透明な移動が可能である．遠隔実行中の値の変更が呼び出し側に返されるため，関数閉包の環境を書き戻すコストを要するものの，より直観的なプログラミングを行なえるという利点がある．

1節で述べた応用領域では，固定的なネットワーク構成を仮定することができない．このため本研究では，無制限の寿命を持つSchemeの一級継続（first-class continuation）を移動することは適当でないとの立場から，継続の移動を許していない．開放型分散環境を想定した継続の拡張は今後の課題である．

6節で述べたPWBでは，アプリケーションの本体をサーバ側に用意し，端末上で実装されるGUIを遠隔実行によって制御している．これにより，ユーザや状況に応じて振る舞いを変える動的なプログラムに移動性を持たせることが可能となった．PWBではイベントを用いて実行時に機能とインタフェースを関連づけており，視覚的表現の実現の自由度が高い．

8 関連研究

Kali Scheme[5]は，アドレススペースおよびプロクシという2つの抽象化機構を導入することで，任意のSchemeオブジェクトを含むメッセージをノード間で交換可能としている．ただし，高信頼性のネットワークを仮定しており，接続管理等は行なっていない．

渡部[18]は，部分継続（partial continuation）を用いた移動コードのための言語機構を提案している．部分継続は一般的であり記述力も高いが，本研究で想定している応用領域においては無制限の寿命を持つ継続の扱いが難しい．

Migratory Applications[3]は，インタフェースを持つアプリケーションの移動性を実現している．インタフェースの状態を含むプログラムの実行イメージを完全に転送できるが，規模に比例した転送コストがかかる．これに対して，PWBでは部分プログラムの遠隔実行によってアプリケーションの移動性を実現しており，サーバの存在を前提として効率の良い転送を可能としている．

PWBの実装は，システムに環境変化への適応性を持たせる研究[14, 16]にも関連する．これらは手続き的リフレクションをベースに動的適応性を実現している．PWBで用いた手法はwrapperとイベントによる単純なものであるが，プレースを環境と考えれば，遠隔実行される学習機構を環境に適応させているとみなすこともできる．Plangent[19]ではプラニングを用いている．

インタフェースの外部制御に関する興味深い研究に，ScriptAgent[8]で用いられた「マリオネットストリング」アプローチがある．これは，内部表現と商用アプリケーションのマクロ言語との相互変換を行なうことで，複数のアプリケーションを統一的に扱うものである．相互変換にはCommon Lispで記述されたparser/unparserを利用している．これに対して，PWBではwrapperに制御を委譲（delegate）することで，学習機構の独立性を高めている．

9 まとめ

プログラムの移動性を実現するための遠隔実行機構を提案した．これはプログラムの部分移送に基づいており，実行状態を保存したまま指定した部分のみを転送することができる．従って，移動部分を分離したり，実行状態の保存や復元を意識する必要がなく，より自然なプログラミングが可能となっている．また，具体的な応用例の実装を示すことで，ユーザや状況に応じて振る舞いを変える動的なプログラムの移動性を，本研究の遠隔実行機構を用いて実現できることを示した．

[3]ネストした`remote`式によって同様の動作が可能であるが，必ず呼び出し側に制御が戻る点において本質的に異なっている．

参考文献

1) Aglets Software Development Kit. `http://www.trl.ibm.co.jp/aglets/`.

2) David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. In *Machine Learning*, Vol. 6, pp. 37–66, 1991.

3) Krishna Bharat and Luca Cardelli. Migratory applications. In *Proc. ACM Symposium on User Interfaces Software and Technology*, Pittsburgh, PA, November 1995.

4) Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, 1983.

5) Henry Cejtin, Suresh Jagannathan, and Richard Kelsey. Higher-order distributed objects. In *ACM Transactions on Programming Languages and Systems*, September 1995.

6) Allen Cypher, editor. *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.

7) IEEE. *IEEE Standard for the Scheme Programming Language (IEEE P1178)*, 1991.

8) Henry Lieberman. Integrating user interface agents with conventional applications. In *International Conference on Intelligent User Interfaces*, San Francisco, January 1998.

9) ObjectSpace Voyager. `http://www.objectspace.com/Voyager/`.

10) Sun Microsystems Inc. *Java Object Serialization Specification*.

11) Zhonghua Yang and Keith Duddy. Corba: A platform for distributed object computing.

12) Taiichi Yuasa. An object-oriented scheme system bubu with seamless interface to java. In *Proceedings International Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications*. World Scientific, 2000.

13) 椎尾一郎. 実世界指向インタフェース. 人工知能学会誌, Vol. 14, No. 6, 1999.

14) 天野憲樹, 渡部卓雄. 動的適応可能なソフトウエア・モデルのための言語的アプローチ. 第 1 回プログラミングおよび応用のシステムに関するワークショップ（SPA'98）. 日本ソフトウエア科学会, March 1998.

15) 増井俊之. 予測/例示インタフェースの研究動向. コンピュータソフトウェア, pp. 1–16, May 1997.

16) 鵜林尚靖, 玉井哲雄. オブジェクト間協調に基づく環境適応型プログラミング言語 eden の設計. 情報処理学会論文誌, Vol. 39, No. SIG 1 (PRO 1), pp. 50–60, 1998.

17) 平野聡. HORB Flyer's ガイド.

18) 渡部卓雄. 部分継続にもとづく移動コード機構について. ソフトウエア工学の基礎 V （日本ソフトウエア科学会 FOSE'98）, pp. 10–19. 日本ソフトウエア科学会, 近代科学社, November 1998.

19) 永井保夫, 田原康之, 入江豊, 大須賀昭彦, 本位田真一. Plangent i: インテリジェント・ネットワークエージェント. In *JSAI Technical Report SIG-HOT/PPAI-9603-6*, pp. 29–36, 1996.

A 性能評価用プログラム

5.4 節で使用した性能評価用プログラムを示す.

- `empty` 関数

```
(define (empty)
  (remote ("robin" 8140)))
```

- `fib` 関数

```
(define (fib)
  (remote ("robin" 8140)
    (define (fib n)
      (cond ((= n 0) 0)
            ((= n 1) 1)
            (else
             (+ (fib (- n 1))
                (fib (- n 2))))))
    (fib 20)))
```

- `trlis` 関数

```
(define (trlis)
  (let ((l *1000-integers*))
    (remote ("robin" 8140)
      (collect-positive l))))
```

`*1000-integers*`は 1,000 個の整数を要素とするリストである（500 個が正数）. `collect-positive` 関数はプレース上で定義されている.

```
(define (collect-positive l)
  (cond ((null? l) '())
        ((positive? (car l))
         (cons (car l)
               (collect-positive
                (cdr l))))
        (else (collect-positive
               (cdr l)))))
```