携帯電話間をマイグレーションするScheme言語処理系

松崎泰裕[†] 長慎也[††]
兼宗進[††] 並木美太郎[†††]

本論文では，携帯電話の Java VM 上で動作する Scheme 処理系「YaScheme」の開発に本処理系では Scheme の継続を携帯端末や計算機間でマイグレーションすることに対応し，モバイルエージェントの仕組みを用いた開発に対応する．さらに携帯電話上で Scheme プログラムを入力してその場で実行することにも対応する．この入力ためのエディタは，Emacs が Lisp で拡張できるように，Scheme で拡張できるものとする．処理方式にツリートラバース型インタプリタを採用して実装を行った結果，処理系の核部分のサイズは約 55 KB，起動後の実行時メモリは約 230 KB となった．今後の課題は，マイグレーションによる情報処理などの応用システム開発である．

Scheme Interpreter which migrates between Cellular Phones

Yasuhiro Matsuzaki,[†] Shinya Cho,[††] Susumu Kanemune[††]
and Mitaro Namiki[†††]

This paper describes development of a Scheme interpreter YaScheme running on Java VM for cellular phones. The interpreter enables the migration of a continuation to another phones. The program editor is extendable using by Scheme like Emacs. The interpretation and execution of scheme programs are traversing the program tree of Scheme codes. The size of this interpreter core is 55 KB and the amount of using memory is 230 KB. Application systems such as distributed processing or information system with the migration of continuations are prospected.

1. はじめに

近年，携帯電話はとても普及しており，多くの人が持つ小型の計算機として，その計算機の上で稼働する応用システムが必要とされている．携帯電話の特徴は，一人一台持っている端末であること，また各端末に CPU や記憶装置などの資源が存在することである．この特徴を考えると，応用システムの開発手法としてモバイルエージェントの仕組みを用いた手法が挙げられる．例えば，プログラムが各計算機間をマイグレーションし，各計算機に存在する記憶装置や入出力装置から検索や入出力を行ったりするエージェント，または自己を書き換えながらマイグレーションするエージェントなどが考えられる（図 1，2.1 参照）．

† 東京農工大学大学院情報工学専攻
Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology
†† 一橋大学 総合情報処理センター
Computer Center, Hitotsubashi University
††† 東京農工大学大学院共生科学技術研究院
Graduate school of Engineering, Tokyo University of Agriculture and Technology

現在の携帯電話は Java VM を搭載しており，アプリケーション（以下 AP）の開発は Java 言語を用いて可能となっている．しかし現在の携帯電話の Java では，前に挙げたモバイルエージェントの仕組みを考慮しておらず，実現には開発基盤が必要となっている．現在の携帯電話の Java VM では動的に生成されたバイトコードを実行することが制限されているため，携帯電話上でエージェントプログラムの作成や変更は不可能である．また日本の各キャリアは同じ Java 言語を採用しているものの，各キャリアによってプロファイルが異なるため，キャリアを越えたマイグレーションにはプロファイルの相違への対応も必要となっている．

本研究ではこの問題を解消するために，携帯電話の Java VM 上で動作する Scheme 言語処理系「YaScheme」を開発する．本処理系では携帯電話のネットワークを通して，Scheme の継続を移送することに対応し，Scheme 言語を用いて携帯電話間をマイグレーションするモバイルエージェントシステムの開発基盤を築く．

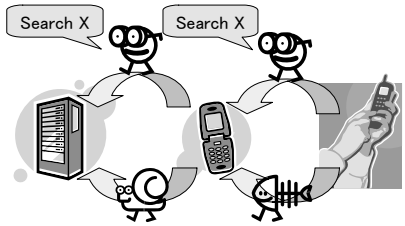

図 1 マイグレーションイメージ図
Fig. 1 the image of the migration

2. AP 例と既存処理系

本章では，まず具体的に応用システムの例を挙げる．次に携帯電話の Java VM 上で動作する既存の言語処理系を紹介し，その問題点や応用システム実現の可否を考える．また，PC の Java VM 上で動作する言語処理系についても紹介し，携帯電話への移植の可否を考える．

2.1 応用システム例

応用システムの例として，まず情報収集エージェントが挙げられる．各携帯端末にある情報から必要なデータの検索を行ったり，その端末を操作しているユーザへデータの入力を求めたりし，集計を行うシステムである．具体的には，会議を行う日程を決定するために各参加者のスケジュールを収集したいという例が挙げられる．このようなシステムは携帯電話の Java でも実装が可能であるが，プログラムの通信処理は複雑になる．また各端末へ個別のプログラムのインストールが必要であり，プログラムや収集内容が変わった場合にはすべての端末での再インストールが必要となる．従ってモバイルエージェントの仕組みを用いた実装が望ましい．

他の例として，ゲームが挙げられる．現在，携帯電話で通信対戦ができるゲームは多く存在するが，それらは対戦をする各端末にゲーム毎のプログラムのインストールが必要である．あるユーザがあるゲームで友人と対戦をしようとした時に，友人の端末に同じゲームが既にインストールされていることは少なく，対戦のためにインストールをしてもらう必要がある．またゲームとしてロボットの思考ルーチンをプログラムで入力して戦わせるといったゲームもあるが，そのようなゲームは現在の携帯電話の Java では難しく，Java VM 上のスクリプト言語が必要である．

さらにマイグレーションする端末は携帯電話だけに限らず，一般の計算機と携帯電話間のマイグレーションも考えられる．計算機上で作業していたものを携帯電話へマイグレーションし，外出先などで作業を継続することや，携帯電話からのエージェントが計算機へマイグレーションし，計算機上のデータベースなどで処理をするといった例が挙げられる．

2.2 soyBasic

soyBasic[1] は，NTT DoCoMo の i アプリとして動作する BASIC 言語処理系である．処理対象の言語は BASIC のサブセット言語を採用しており，対話実行や行番号による実行をサポートしている．携帯電話上での入力に配慮をしており，編集状況や文脈などから自動的に予約語などの候補を表示し，コード補完を行う．

この処理系は，携帯電話上でプログラムを入力して実行するという点では優れており，前節で挙げた思考ルーチンプログラムの入力ゲームなどには適している．しかし，携帯電話の通信機能などには対応しておらず，マイグレーションについても非対応である．また，BASIC のサブセットという言語の特性上，比較的大きなシステムの開発は困難である．現在提供されている処理系の動作するプロファイルは，Web アプレットと i アプリ（DoJa）のみである．

2.3 JAKLD

JAKLD[2] は，Java アプリケーション組み込み用の Lisp ドライバである．主に Java アプリケーションへの組み込み用途を想定して開発されているが，単体の処理系としても利用可能であり，NTT DoCoMo の i アプリ版も公開されている．処理対象の言語は IEEE Scheme を採用しているが，準拠していない点として次の 3 点がある．

- 継続は，脱出にのみ使用できる
- 末尾呼び出しの最適化は行われない
- 文字列は immutable であり，既存文字列の変更ができない

この処理系の問題点として，まず携帯電話向けの API が提供されていないことが挙げられる．i アプリ版が公開されているものの，携帯電話の液晶画面への描画や通信機能へのアクセスをするための手続きが提供されておらず，数値計算や文字出力のみとなっている．

また，IEEE Scheme に準拠していない点も問題となる．継続や末尾呼び出しの最適化は Scheme の特徴となっており，一般の AP 開発においては必要となると考えられる．

2.4 SISC

SISC[8] は，PC の Java VM 上で動作する Scheme 言語処理系である．言語仕様は R5RS[3] に完全に準拠している．Scheme プログラムを独自の中間コードに

コンパイルし，独自の VM で実行する．

SISC ではネットワークアクセスなどの手続きが提供されているが，それらは PC 用であり，携帯電話ではプロファイルの差異や制限により動作しない．携帯電話への移植をするには多くの変更が必要となる．また，処理系の JAR パッケージのサイズは約 210 KB となっている．現在の携帯電話の Java では，JAR パッケージのサイズが制限されており，現在広く普及しているものでは制限は約 100 KB[5]〜256 KB[6] となっている．最近ではこの制限も緩くなる傾向にあるが，処理系上で実行するプログラムも JAR パッケージへ格納することを考えると，余地を残して処理系を小さくする必要がある．サイズの削減やプロファイルの変更を考えると，携帯電話への移植は容易ではない．

2.5 Kawa

Kawa[9] は，PC の Java VM 上で動作する Scheme 言語処理系である．Scheme プログラムを Java のバイトコードにコンパイルし，Kawa 自身を実行する Java VM で実行する．Java との親和性の高さを特徴としており，Scheme から Java の各機能へのアクセスが可能となっているが，その反面，継続の使用や末尾呼び出しの最適化に制限が存在する．

携帯電話の Java VM では動的に生成したバイトコードの実行が許されていないため，Kawa 処理系の携帯電話への移植は不可能である．

3. 本研究の目標

本研究では，携帯電話の Java VM 上で動作する Scheme 言語処理系「YaScheme」を開発する．Scheme 言語は継続をファーストクラスオブジェクトとして扱える言語であり，この継続を携帯電話のネットワークを通して移送することに対応することで，Scheme プログラムが携帯電話間をマイグレーションすることを実現する．

本研究では S 式を入力するためのエディタを開発し，携帯電話上での Scheme プログラムの入力や，Scheme プログラムに対するデータの入力に対応する．Scheme 処理系と共に用いることを活かし，Emacs が Lisp で拡張できるように，このエディタは Scheme で拡張や操作ができるようにする．これによりユーザによる自由な拡張を実現し，携帯電話の制限されたキー配列でのより効率的な入力を実現する．

Scheme の言語仕様は R5RS[3] に準拠する．現在広く使われている Scheme の言語仕様としては，他に Common Lisp や Emacs Lisp などがある．しかしそれらの仕様は携帯電話を考慮しておらず，携帯電話には不要なものも含まれているため，Scheme 言語として最低限の仕様を定めた R5RS を採用する．R5RS では強力なマクロ機能を提供しており，記述を短くするための新たな構文の定義やミニ言語の作成なども可能である．この機能は携帯電話の限られた入力装置においてエージェントなどのプログラムを入力する際に，プログラムを短く記述する上で有効である．このマクロ機能は，JAKLD の採用する IEEE Scheme には含まれていない．

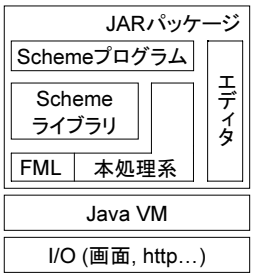

図 2 本処理系の全体構成
Fig. 2 The composition of the interpreter

本処理系では，現在多くの携帯電話で採用されている DoJa プロファイルおよび MIDP，そして一般の計算機向けの Java 2 Second Edition で動作するものとする．各プロファイルでは画面表示などの API が異なっているが，エージェントが異なるプロファイル間でもマイグレーションできるように，この差異は本処理系が吸収し，Scheme プログラムに提供する手続きは共通とする．

現在の携帯電話の Java VM では，プログラムの JAR パッケージサイズが制限されており，現在広く普及しているものでは約 100 KB[5]〜256 KB[6] と厳しい．本処理系の JAR パッケージに実行する Scheme プログラムを格納する場合を考慮し，本処理系の動作に不可欠なインタプリタコアのサイズは 60 KB 以下を目標とする．

4. 本処理系の概要

本処理系の全体構成は図 2 のとおりである．本処理系で実行する Scheme プログラムは，計算機上であらかじめ作成して本処理系の JAR パッケージへ格納して本処理系の起動時に実行するか，または携帯電話上のエディタで Scheme プログラムを入力し，その場で実行するものとする．

Scheme 言語処理系において Scheme プログラムを実行する方式には，表 1 の 3 通りがある．ネイティブコンパイル方式は Kawa が採用する方式であり，Java

表 1　Scheme プログラムの処理方式
Table 1　Methods to process a program

方式	実装	処理速度	サイズ
ネイティブコンパイル	複雑	高速	大きい
Virtual Machine	複雑	中速	大きい
ツリートラバース	単純	低速	小さい

と同等の速度を実現できるが，現在の携帯電話の Java VM では動的に生成したバイトコードの実行が許されていないため，この方式は使用できない．VM 方式は SISC が採用する方式であり処理速度も良いが，コンパイラと VM の両方が必要となるため，処理系のサイズが大きくなる傾向にある．前に述べたとおり現在の携帯電話では JAR パッケージのサイズ制限が厳しいため，なるべく処理系を小さくすることを考え，ツリートラバース型インタプリタ方式を採用する．

携帯電話の Java における二次記憶には，DoJa にはスクラッチパッド，MIDP にはレコードストアがあるが，これらのプロファイルの間には大きな仕様の違いがある．また，R5RS の二次記憶へのアクセスはファイルシステムを想定しているが，スクラッチパッドやレコードストアにはファイルシステムの概念が無い．そこで両プロファイルに対応したファイルシステムライブラリである FML[4] を利用して，Scheme プログラムに入出力機能を提供する．ただし FML の追加による JAR パッケージの増加が問題となるため，FML の不要な機能を削減し，シーケンシャルアクセスのみに対応したサブセット版を使用する．また必要に応じて FML ライブラリの削除ができるようにする．

5. 評価器の設計

本章では，インタプリタの核である評価器の設計について述べる．携帯電話の Java VM ではサイズ制限が厳しいため，省サイズを目指した設計とする．また評価器はマイグレーションが可能な構造とする．

5.1 データ表現

Scheme においてすべてのアトムは S 式であり，この構造は Java の継承関係を利用して表現できる．本処理系ではサイズを削減するために，S 式を表わすためのクラスを新たに定義せずに，Java で暗黙のうちにすべてのクラスの基底クラスとなる Object クラスが S 式を表わすものとする．各アトムの区別には Java の instanceof 演算子を用いてクラスの識別を行って区別する．

本処理系における各アトムの表現クラスは図 3 のとおりである．点線で囲まれたクラスは，Java の標準クラスである．本処理系では，処理系のサイズを削減するためにできる限り Java の標準クラスを用いる．真偽値，文字，数値は，それぞれ Java のプリミティブ型の配列で保持する．Java にはプリミティブ型のためのクラスが存在するが，インスタンスの生成や値へのアクセスがより高速な配列を採用する．文字列については String クラスの他に StringBuffer も用いることで，書き換え可能文字列に対応する．

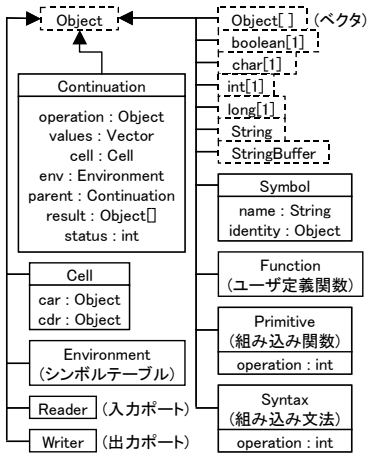

図 3　データ表現クラス
Fig. 3　Classes for Scheme Atoms

5.2 式の評価処理

式の評価は，評価対象のリストの各要素を部分式として再帰的に評価し，先頭要素の評価結果に応じて処理を行う．ただし先頭要素を評価して構文である場合には，各構文に応じて必要な要素のみを評価する．

部分式を処理するには，単純な方式としては再帰呼び出しを用いて処理する方式がある．これは JAKLD でも用いられている方式である．この方式では，評価途中の部分式のコンテキスト情報が評価器自身の Java のスタックへ格納される．継続や末尾呼び出しの最適化ではこのコンテキスト情報へのアクセスが必要となるが，Java ではスタックへのアクセスが許されていないため，実現が困難である．本処理系では継続に対応し，さらにその継続のマイグレーションへ対応することを目標としているため，部分式の処理方式としてこの方式と採用することはできない．

本処理系では部分式の評価に再帰呼び出しを用いずにループのみで処理をし，評価途中の部分式のコンテキスト情報をヒープ領域に格納して，評価器が自由にアクセスできるようにする．本処理系では Continuation クラスを定義し，そのインスタンスが各部分式のコンテキスト情報を保持する．Continuation クラ

スが保持する情報は次のとおりである．

- 命令 行うべき処理を表わす命令である．S 式の先頭を評価して得た，構文や手続きが格納される．
- 評価済み要素の結果
  命令より後の各要素のうち，既に評価を終えた結果を保持する．
- 未評価残り要素
  これから評価すべき残りの要素を保持する．
- 環境
  この部分式を評価する環境（シンボルテーブル）を保持する．
- 次の継続
  現在の評価が終わった後に処理すべき継続，親となる式への参照を保持する．

この方式により，末尾呼び出しの最適化にも対応する．ある評価しようとしている式が末尾である場合には，評価が終わった後に処理すべき次の継続として現在の継続ではなく 1 つ上の継続を格納することで最適化を実現する．

5.3 継　　続

前節で述べたとおり，本処理系では Continuation クラスのインスタンスが各部分式におけるコンテキスト情報を保持しており，評価のある時点での継続はこのインスタンスから構成されるリストで表現されている．Scheme プログラムから継続の取得が要求された場合には，このリストのコピーを評価結果とする．ここでコピーが必要となるのは，要求の後に実行される継続と，評価結果として返された後に起動されるであろう継続の干渉を防ぐためである．継続の起動がされた場合には，評価対象を起動すべきリストの末端の Continuation クラスのインスタンスに切り替えて処理を行う．

5.4 シンボルテーブル

ラムダ式で作られる環境は，Environment クラスで各環境におけるシンボルテーブルと親となる環境への参照を保持する．R5RS Scheme ではマクロを使用した時に，同じ階層に存在する同じ名前のシンボルが別の束縛となることがある．この問題に対処するために，シンボルテーブルにはハッシュテーブルを用いるが，その走査にはシンボル名を用いない．マクロの展開時に各シンボルに束縛ごとにユニークなオブジェクトを割りあて，そのオブジェクトでシンボルテーブルの走査を行う．

5.5 組み込み構文と手続き

R5RS で定義されている組み込み構文と手続きは，各手続きごとにユニークな数値を割り当て，評価器で switch 文を用いて分岐して処理を行う．

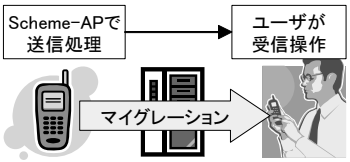

図 4　マイグレーションの流れ
Fig. 4　an operation of the migration

6. マイグレーション

本研究ではモバイルエージェントシステムの開発基盤として，携帯電話間の Scheme プログラムのマイグレーションを実現することを目標としている．本章ではそのマイグレーションの実現方法と，移送が必要となる各オブジェクトの対応方法について述べる．

6.1 マイグレーション概要

マイグレーションとは，動作していたプログラムがある時点で一時停止し，別の端末へプログラムが移動して，一時停止したところから移動先の端末の資源で実行を再開することである．この機能により，各端末への個別のプログラムのインストールは不要となり，2 章で述べた AP 例などが実現可能となる．マイグレーションするデータは，構文解析済みのプログラムと，シンボルテーブルや実行場所などを保持するコンテキスト情報である．

携帯電話のネットワーク通信には大きな制限があり，その通信プロトコルは HTTP のみ，また通信は自発的なクライアント通信のみとなっている．携帯電話間の直接通信は不可能である．この制限のため，マイグレーションを行う場合には図 4 のように，まず移送元の端末で送信操作をして一旦データを HTTP サーバーへ送信し，次に移送先の端末で受信操作をすることでマイグレーションを実現する．

Scheme プログラムでマイグレーションを行う際には，まず移送元の端末で図 5 のように継続を取得し，その継続を HTTP で送信する．次に移送先の端末で図 6 のように HTTP で受信した継続を起動する．これらのコードをプログラムの任意の場所に挿入することで，マイグレーションを実行できる．

継続を HTTP 通信を通して転送するには，Continuation オブジェクトを HTTP 通信で扱えるバイト列へ変換する必要がある．本処理系ではオブジェクトが持つメンバの値を再帰的に参照し，辿り得るすべての参照を型情報と数値や文字列としてバイト列へ変換する．移送先ではこの型情報と値からインスタンスと参

```
(call/cc
  (lambda (cont)
    (define http (Connector-open "http://foo"))
    (http-setRequestMethod http "POST")
    (let ((out (http-getOutputPort http))
      (write-object cont out)
      (close-output-port out)
      (http-connect http)
      (http-close http)))))
```

図 5 マイグレーション送信プログラム
Fig. 5 The receiving migration

```
(let ((http (Connector-open "http://foo")))
  (http-setRequestMethod http "GET")
  (http-connect http)
  (let ((in (http-getInputPort http)))
    (define cont (read-object in))
    (close-input-port in)
    (http-close http)
    (cont)))
```

図 6 マイグレーション受信プログラム
Fig. 6 The sending migration

照を作り直し，復元する．シリアライズの際には入出力順に番号をつけ，既出のオブジェクトについては番号で参照することで循環参照にも対応する．

携帯電話特有のオブジェクトには，上記のように単純にシリアライズできないものがある．次節以降ではそれらのオブジェクトについて述べる．

6.2 グラフィックス

携帯電話の液晶画面への表示には，大きく分けて二つの方法がある．一つはキャンバス描画であり，液晶画面やオフスクリーンのビットマップに対して線などの低レベルな描画を行う方法である．もう一つは高レベルな GUI コンポーネントを用いた方法であり，あらかじめ用意されているボタンや入力エディタなどのコンポーネントを必要に応じて画面へ追加して表示するものである．

キャンバスのマイグレーションを行う際には，各点に描画されている内容を記録して転送し，移送先でビットマップを復元する．ただしそのデータ量は多いため，DoJa では JPEG 形式で圧縮して転送する．JPEG 形式は不可逆圧縮であるが，キャンバスはユーザへのグラフィックス表示に用いるものであり，目で見てわからない程度の劣化は問題ない．MIDP に関しては JPEG 圧縮機能が無いため，圧縮は保留とし，非圧縮で送る．

GUI コンポーネントのマイグレーションを行う際には，追加されているコンポーネントの種類とそのキャプションなどの内容を記録して転送し，移送先で同じコンポーネントを作成して再構築することで対応する．DoJa では追加されているコンポーネントの一覧が取得できないため，追加を行う際に別に記録しておくことで対応する．

6.3 HTTP 通信

マイグレーションする Scheme プログラムが，何らかの別の HTTP 通信を行うプログラムである場合がある．本処理系ではそのようなプログラムにも対応し，マイグレーション後に通信することや，受信データの処理を行うことに対応する．

携帯電話の HTTP 通信には前に述べた以外にも制限があり，接続中に送受信するデータを操作することができない．データを送信する場合にはあらかじめデータを設定しておき，接続を要求をしてはじめてデータが送信される．受信データはメモリ上にバッファリングされ，切断されてはじめてアクセスが可能となる．また通信は同時に複数行うことはできない．これらの制限により，マイグレーションを行う際には Scheme プログラムが行う通信の状態は，接続を開始する前か，切断後のどちらかとなる．

接続前であった場合には，接続すべき URL と，既に設定されている送信すべきデータをシリアライズして転送する．移送先ではメモリ上にストリームを作成し，Scheme プログラムが送信したいデータの続きを追加できるようにする．

切断後であった場合には，受信データのうち，Scheme プログラムがまだ処理していないものをストリームからすべて読み出して転送する．移送先では転送したデータを元にストリームを作成し，Scheme プログラムが受信データの続きを処理できるようにする．

6.4 二次記憶

本処理系では携帯電話の二次記憶であるスクラッチパッドやレコードストアへのアクセスには，ファイルシステムライブラリである FML を利用する．ファイルを開いている途中の Scheme プログラムをマイグレーションする際には，ファイルのまだ処理していない残りの内容を転送し，移送先でストリームを作成して続きを処理できるようにする．

マイグレーション先で新たに移送元にあったファイルを開く場合には，あらかじめファイルの内容を転送しておく方法と，要求があった時に転送する方法がある．しかし携帯電話の通信は自発的にしか行えないため，後者の方法は困難である．本処理系では前者の方法を採用し，マイグレーションをする際に二次記憶の内容を同時に転送しておき，移送先ではメモリ上の二次記憶のコピーに対して FML で処理を行い，移送元にあったファイルの処理に対応する．

表 2　API 一覧
Table 2　API for cellular phone

手続き名	説明
show	画面に Panel,Canvas を表示
setSoftLabel	ソフトキーを設定
make-Panel	Panel を作成
Panel-add	Panel に Item 追加
Panel-setKey...	キー押下時の手続きを指定
make-TextField	TextField 作成
TextField-getText	文字列を取得
TextField-setText	文字列を設定
make-Canvas	Canvas を作成
Canvas-getGraphics	Graphics を取得
Canvas-setPaint	描画時に起動する手続きを指定
Canvas-setKey...	キー押下時の手続きを指定
Canvas-repaint	再描画を要求
:	
Image-createImage	Image を作成
Image-getGraphics	Graphics を取得
:	
Graphics-drawImage	Image を描画
Graphics-drawLine	線を描画
Graphics-setColor	描画色を指定
:	
Connector-open	URL から http 接続を作成
http-setRequestMethod	GET/POST メソッドを指定
http-getOutputPort	送信データを設定
http-connect	通信終了時に起動する手続きを指定して，実際に接続する
http-getInputPort	受信データを取得
read-object	入力からオブジェクト復元
write-object	出力ポートへシリアライズ

7. API の設計

携帯電話の機能を操作するための API は，Java で提供されるものと同様のものを Scheme プログラムに対しても提供する．ただし Java と異なり，Scheme はオブジェクト指向ではないため，操作対象のオブジェクトを引数に渡す形で記述する．本処理系では各プロファイルの差異を吸収して Scheme プログラムに対して共通の API を提供する．API の一部を表 2 に示す．

8. S 式エディタ

8.1 エディタ概要

本処理系では S 式を入力するためのエディタを搭載し，携帯電話上での Scheme プログラムの入力や，実行中の Scheme プログラムに対するデータの入力に対応する．このエディタでは，Emacs が Lisp で拡張できるように，Scheme での拡張や操作に対応する．携帯電話のリソースには限りがあり，様々な機能を搭載したエディタの実現は難しい．しかしこの拡張機能により，ユーザが必要となる機能，例えば高度な置換機能などを Java のコードに変更を加えることなく追加

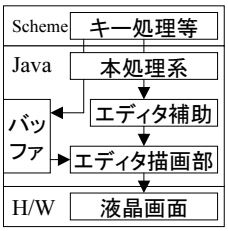

図 7　エディタ全体構成
Fig. 7　The composition of the editor

可能となる．また機械的に生成できる長いデータなどは Scheme プログラムで生成することで，キーの制限された携帯電話での入力を短縮できる．

8.2 全 体 構 成

携帯電話の Java で用いることができる GUI コンポーネントには，テキストを入力するためのエディタも用意されている．しかしこのエディタはメールなどの文章を入力するためのものであり，S 式など，特に記号の入力には向いていない．またこのコンポーネントを拡張することも許されていない．そこで本処理系では低レベルのキャンバス描画を用いて独自のエディタを作成する．

エディタの全体構成は図 7 のとおりである．エディタの実装は，基本的な部分は Java で行う．携帯電話の液晶画面は大きいものではなく，表示できる情報はある程度固定されてしまうと考えられる．拡張が必要とされるのは，主に携帯電話でネックとなっているキー操作になると考えられる．速度の高速化も考え，バッファから画面への描画は Java で実装し，Scheme にはこのバッファへの操作をする API を提供する．また，後に述べる識別子予測や補完などの処理についても，速度が必要となるため Java で処理する．

エディタの概観は図 8 のとおりである．基本的には現在の携帯電話のメールなどの入力画面と同様の構成とする．文字の入力は，半角の英数字や記号についてはエディタ画面でボタンを押すことで入力可能とする．漢字などの全角文字については，入力に膨大な辞書などが必要となるため，Java の GUI コンポーネントでネイティブのエディタを呼び出して入力を受け付ける．

8.3 設　　計

バッファの構造にはいろいろな方法があり，Emacs では連続した位置への編集が高速なギャップバッファを使用している．しかしギャップバッファは高速な反面，メモリに無駄があり，処理も複雑である．携帯電話で編集するプログラムは，あまり大規模ではないと考えられる．そこで本エディタでは Scheme における

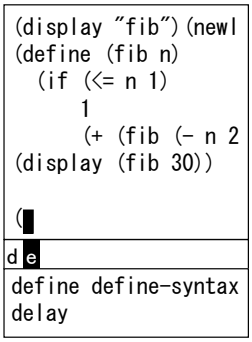

図 8 エディタ概観
Fig. 8 The image of the editor

可変文字列を行ごとのバッファとし，そのバッファが集まったベクタが全体のバッファを表すものとする．

本エディタではSchemeプログラム入力の効率化を図るために，少ないタッチ数での入力を実現する．まずSchemeで頻繁に使用される括弧記号は，開き括弧と閉じ括弧をそれぞれ携帯電話のアスタリスクとシャープボタンで入力可能とする．またスペースは0ボタンで入力される．これらのボタンは単語の入力途中には大文字への変換や訂正に使われるが，単語が確定されて何も入力途中でない場合には使われないため，これらの記号の入力に用いる．さらに識別子の入力においてタッチ数を減らすために，soyBasicで用いられている方式を拡張して採用する．通常の入力方式では英字を1文字入力するために同じボタンを最大4回押さなければならないが，1文字に対して1回ボタンを押し，残りは識別子辞書により予測する．例えばdefineと入力する場合には，333463と押せば良い．予測結果は下部に候補が複数表示され，選択すると入力される．識別子辞書には，R5RSで定義されている識別子，およびSchemeプログラムを読み込んだ時に使われている識別子，さらに今までに入力した単語を登録しておく．辞書検索のキーには押すボタンの数字列を使用し，二分探索により検索する．

識別子辞書による予測入力では，識別子辞書に存在しない新しい単語の入力ができないため，現在日本の携帯電話で一般的な方法でも入力可能とする．この入力方式においても，識別子辞書から前方一致の単語を検索し，補完候補を表示する．前に述べた予測入力で候補が見つからない場合には，自動的にこの従来の入力方式による補完候補を表示するため，この二つの入力方式は切り替えをせずに利用可能である．

表 3 処理系のサイズ
Table 3 The size of the interpreter

対象環境	本処理系	JAKLD
DoJa	55 KB	40 KB
MIDP	55 KB	-
PC	45 KB	36 KB

9. 実装と評価

9.1 処理系のサイズ

本研究では実際に実装を行い，処理系の最終的なJARパッケージのサイズを確認し，他の処理系と比較した．結果は**表 3** のとおりとなった．

本処理系のサイズはSchemeで書かれたライブラリなどを含めたJARパッケージで約55 KBとなった．またファイルシステムライブラリであるFMLを追加した合計のサイズは，約73 KBとなった．本処理系では新たにクラスを定義せずになるべくJavaの標準クラスを利用することで省サイズを実現した．この55 KBというサイズは，一般的な携帯電話でのJARパッケージのサイズ制限である100 KB[5]の約半分であり，処理するSchemeプログラムをJARパッケージに埋め込む十分な余地が残っている．FMLを追加した場合はやや大きなサイズとなっているが，ファイルシステムの必要に応じてライブラリの追加や削除をすれば良い．

JAKLDと比較すると，約15 KBの増加となった．これは携帯電話向けAPIを追加した上でのサイズであり，携帯電話向けAPIを省いた純粋なSchemeとしての本処理系は約45 KBである．この結果より，継続への対応による処理系サイズへの影響は小さいと言える．本処理系では継続への対応をした上でサイズを既存処理系と同等に抑えることができ，また携帯電話向けAPIを追加した上で約15 KBの増加に抑えることができた．

9.2 マイグレーションの評価

本処理系のマイグレーション機能を評価するために，マイグレーションをする簡単なAPを開発し，その動作を確認した．実際に，A社の携帯電話においてで動作させている様子が図9である．本APはカレンダーを表示し，ユーザにスケジュールの入力を要求し，そして別の端末へマイグレーションして別のユーザも入力を行うデモである．A社の携帯電話において本APを実行して入力し，その後B社の携帯電話にマイグレーションしてAPが再開され，継続入力ができることを確認できた．

マイグレーション時にHTTPで転送されるデータ

図 9 マイグレーションデモ
Fig. 9 The image of the editor

表 4 メモリ使用量
Table 4 The amount of the used heap memory

環境	起動時	リスト作成時
A 社携帯	233 KB	21 KB
B 社携帯	221 KB	21 KB
C 社携帯	232 KB	21 KB
PC	280 KB	16 KB

のサイズを確認したところ，約 35 KB であった．このサイズには起動時から変更されていない R5RS 定義の束縛も含まれており，そのサイズは約 30 KB である．変更されていないこれらの束縛は，転送しなくても移送先で復元可能であるため，転送は不要である．したがって改良によりマイグレーションの更なる効率化が可能である．

9.3 実行時メモリの評価

携帯電話 2 台，および PC 上で本処理系を動作させ，メモリ使用量を測定した．測定直前に Scheme 上から Java の GC を起動し，その後に全メモリ容量と使用量を取得することでメモリ使用量を算出した．JAKLD などの他の処理系については，メモリ使用量を取得する API がないため測定を行っていない．

結果は**表 4** のとおりとなった．リスト作成時とは，car 部に空リストを格納した長さ 1000 のリストを作成した時の消費メモリの増加分である．本処理系の消費メモリは起動時で約 230 KB，リスト作成時で 20 KB 程度となっており，これは数 MB〜10MB[7] のヒープメモリを搭載している最近の携帯電話において問題ない使用量である．

9.4 処理速度の評価

携帯電話 2 台，および 3GHz の CPU を搭載した PC 上で実行速度を測定した．比較のために，B 社の携帯電話においては JAKLD[2] についても測定を行った．また PC 上では SISC[8] と Kawa[9] についても測定を行った．比較に用いたコードは，Gabriel ベンチマーク[10] およびループ速度を計測するための**図 10** に示すコードである．ただし，Gabriel ベンチマークは PC 向けであり，携帯電話上では現実的な時間で処理できないため，携帯電話上では tak のみを測定した．また JAKLD と Kawa は継続と末尾呼び出しの最適化に対応していないため，一部の動作しないコードは測定できていない．

```
(define (loop-only n)
  (if (> n 0) (loop-only (- n 1))))
(define (func0) #t)
(define (loop-func n)
  (func0)
  (if (> n 0) (loop-func (- n 1))))
(define (loop-cont n)
  (define i n)
  (define con #f)
  (call/cc (lambda (c) (set! con c)))
  (set! i (- i 1))
  (if (> i 0) (con)))
(define (loop-do n)
  (do ((i n (- i 1)))
      ((= i 0))))
```

図 10 ループプログラム
Fig. 10 The loop program

結果は**表 5**，**表 6** のとおりとなった．ループは 1 ループあたりの時間である．JAKLD と比較すると，B 社携帯電話上では do 構文によるループ loop-do で本処理系が遅い結果が見られるが，PC 上では同等，または他のコードでは逆に速い結果が見られる．JAKLD ではコンテキスト情報が Java VM のスタックに格納されるのに対して，本処理系ではヒープに確保する．携帯電話での結果が PC 上と異なるのは，携帯電話の Java VM の性能によりスタックへの格納とヒープの確保で処理速度が異なるためと考えられる．本処理系では継続や末尾呼び出しの最適化に対応するために処理速度が犠牲になったが，携帯電話の Java VM の性能向上により改善が期待できる．

SISC や Kawa と比較すると速度の差が見られるが，これは Scheme プログラムの処理方式の違いによるものと考えられる．しかし継続を用いたループ loop-cont においては SISC よりも多少速い結果が得られており，一部ではツリートラバース方式によって Virtual Machine 方式と同等の処理速度を実現できている．コンテキストのコピーや復元が頻繁に発生する継続ループでは Virtual Machine 方式での利点である最適化が難しく，コンテキストのコピーや復元という処理のみにおいては二つの処理方式で大きな差は無いためである．

表 5 携帯電話での実行速度
Table 5 Benchmarks on cellular phones

	A 社携帯	B 社携帯		C 社携帯
コード	本処理系	本処理系	JAKLD	本処理系
tak	316.2s	189.2s	26.89s	221s
loop-only	2.170ms	746μs	-	1.230ms
loop-func	3.287ms	1.468ms	-	1.961ms
loop-cont	2.250ms	764μs	-	1.366ms
loop-do	2.431ms	849μs	76μs	1.324ms

表 6 PC 上での実行速度
Table 6 Benchmarks on PC

コード	本処理系	JAKLD	SISC	Kawa
cpstak	786ms	-	101ms	-
ctak	1.601s	13.56s	898ms	3.380s
deriv	1.905s	1.189s	333ms	69ms
destruct	25.64s	24.76s	3.766s	2.031s
div(ite)	12.92s	8.360s	1.188s	328ms
div(rec)	11.82s	10.69s	1.625s	344ms
earley	3.568s	-	1.141s	105ms
fib	12.15s	17.50s	3.516s	1.313s
hanoi	5.724s	7.563s	1.214s	311ms
nqueens	491ms	534ms	63ms	34ms
puzzle	8.812s	10.05s	1.625s	563ms
tak	406ms	491ms	80ms	16ms
takl	3.687s	4.89s	714ms	22ms
takr	436ms	502ms	91ms	70ms
travinit	10.11s	8.204s	2.640s	469ms
travrun	42.95s	38.59s	10.19s	890ms
loop-only	3.739μs	-	0.890μs	-
loop-func	4.890μs	-	1.390μs	-
loop-cont	4.818μs	-	6.500μs	-
loop-do	4.510μs	3.812μs	0.875μs	0.359μs

10. おわりに

本論文では，携帯電話の Java VM 上で動作する Scheme 言語処理系「YaScheme」の設計と実装について述べた．本処理系により，Scheme 言語による携帯電話向け AP の開発が可能となった．本処理系では，従来の処理系で非対応であった継続に対応し，さらにその継続を携帯電話のネットワークを通してマイグレーションすることに対応することで，モバイルエージェントシステムの開発基盤を築いた．またプロファイルの差異を本処理系で吸収することで，キャリアを越えた携帯電話間や，計算機へのマイグレーションを実現した．

現在のマイグレーションでは，継続から辿り得るすべての参照を転送しているため，転送しなくても復元可能なデータや，移送先で使われないデータも転送されている．今後の課題は，必要なデータのみを転送するといったマイグレーションの改良である．また，本処理系によって可能となったマイグレーションを用いたモバイルエージェントシステムの開発も今後の課題である．

参考文献

1) 長慎也: soyBasic, `http://www2.eplang.jp/soyBasic/` (2005).
2) 湯浅太一: Java アプリケーション組み込み用の Lisp ドライバ, 情報処理学会論文誌, Vol.44, No.SIG4, pp.1-16 (2003).
3) R. Kelsey, W. Clinger and J. Rees, editors: The revised5 report on the algorithmic language Scheme. In Higher-Order and Symbolic Computation 11(1), pp. 7–105 (1998).
4) 佐々木悠: 携帯電話の Java で稼動するファイルシステムライブラリの開発, 東京農工大学 工学 部情報コミュニケーション工学科 卒業論文 (2006).
5) NTT DoCoMo: i アプリコンテンツ開発ガイド for DoJa-4.x, `http://www.nttdocomo.co.jp/` (2006)
6) ソフトバンクモバイル: S!アプリ開発ガイド MIDP 1.0 対応端末編, `http://developers.softbankmobile.co.jp/` (2006).
7) NTT DoCoMo: i アプリ対応端末の情報, `http://www.nttdocomo.co.jp/` (2006).
8) Scott G. Miller and Matthias Radestock: SISC for Seasoned Schemers, `http://sisc.sourceforge.net/` (2006).
9) Per Bothner: The Kawa language framework, `http://www.gnu.org/software/kawa/` (2006).
10) Richard P. Gabriel: Performance and Evaluation of Lisp System, MIT Press in Computer Science, MIT Press, Cambridge, MA (1985).