2006年度　卒　業　論　文

# プログラム変換による携帯電話用アプリケーションの移植作業の簡略化について

指導教員：渡辺 大地講師

メディア学部 ゲームサイエンスプロジェクト
学籍番号　M0103259
高橋 武司

**2006 年度　　卒　業　論　文　概　要**

**論文題目**

# プログラム変換による携帯電話用 アプリケーションの移植作業の簡略化について


| | | | | |
|---|---|---|---|---|
| **メディア学部 学籍番号：M0103259** | **氏名** | 高橋 武司 | **指導教員** | 渡辺 大地講師 |



| | |
|---|---|
| **キーワード** | 携帯電話、i アプリ、S!アプリ、Java、プログラム変換 |



現在、国内には NTT DoCoMo、au、Softbank の携帯電話や WILLCOM の PHS が利用されている。発売当初の携帯電話や PHS は電話をするためのものであったが、近年では多機能化が進み、企業や個人が開発したアプリケーションを実行できる商品が販売されている。

しかし、これらの機器向けのアプリケーションはそれぞれ開発環境が異なる。そこで、開発者が複数のキャリアに対応したアプリケーションを開発する場合、それぞれの環境に合わせて別々に開発を行う方法や、相違点に着目してプリプロセッサを用いて開発する方法、片方のキャリア向けに開発したアプリを移植する方法などを用いて実現している。しかし、これらの方法での開発は人手や時間などのコストがかかることや、アプリケーションのサイズが大きくなることにより非力な環境である携帯電話や PHS では制限が大きくなるなど問題が生じる。

そこで、本研究では、アプリケーションのサイズが大きくなりすぎないようにプラットフォームの変換を行うツールを利用することにより、携帯電話や PHS 用のアプリケーションの移植作業を簡略化する方法を提案する。

なお本研究では、現在最も多くの人が利用している NTT DoCoMo 用のアプリケーション「i アプリ」の書式から開発環境が比較的 i アプリに似ており、一般ユーザも容易にアプリケーションを公開することができる Softbank 用のアプリケーション「S!アプリ」の書式への変換を実現した。

# 目 次

# 図 目 次

# 表 目 次

# 第 1 章

# 序章

## 1.1 はじめに

現在、国内にはNTT DoCoMo、au、Softbankの3キャリアの携帯電話や、WILLCOMのPHSが利用されている [1, 2]。携帯電話やPHSは、当初通話のみを目的としていたが、E メールの送受信やインターネットへの接続、カメラ機能などさまざまな機能が搭載されるようになった。これに伴い、コミックを閲覧するシステムや歩行者ナビゲーション情報の表示方法など、エンターテインメントの分野だけではなく、教育や災害対策などのさまざまな分野で利用する方法が研究されている [3, 4, 5, 6]。また、近年発売される携帯電話やPHSの多くは企業や個人が開発したアプリケーションを実行できる環境が用意されている。

しかし、これらの機器向けのアプリケーションはそれぞれ開発環境が異なる [7, 8, 9, 10, 11, 12]。そこで、開発者が複数のキャリアに対応したアプリケーションを開発する場合、それぞれの環境に合わせて別々に開発を行う方法や、相違点に着目してプリプロセッサを用いて開発する方法 [13]、片方のキャリア向けに開発したアプリを移植する方法 [14] などを用いて実現している。しかし、これらの方法での開発は人手や時間などのコストがかかることや、アプリケーションのサイズが大きくなることにより非力な環境である携帯電話やPHSでは制限が大きくなるという問題が生じる。また、異なるキャリア間でのデータ共有を前提とした、マルチベンダ対応アプリケーションを実現するために、アプリケーションに必要な

データをクライアントの機種やキャリアに依存しないサーバに蓄え、必要時にクライアントとサーバの間でデータの送受信を行うデータ管理方式 [15] もあるが、アプリケーション自体の開発の簡略化には触れられていない。サーバ分野やパーソナルコンピュータ用のアプリケーションの移行作業の低減は行われている [16] が、携帯電話や PHS 用のアプリケーションの分野ではまだ確立されていない。

そこで、本研究では、アプリケーションのサイズが大きくなりすぎないようにプラットフォームの変換を行うツールを利用することにより、携帯電話や PHS 用のアプリケーションの移植作業を簡略化する方法を提案する。本研究の検証は、自作の変換元のプログラムと変換後のプログラム、変換ツールを用いて作ったプログラムを比較することによりアプリケーションのサイズを比較する。また、Web に公開されているプログラムを変換することにより、変換ツールの精度確認する。なお本研究では、現在最も多くの人が利用している [17]NTT DoCoMo 用のアプリケーション「i アプリ」の書式から開発環境が比較的 i アプリに似ており、一般ユーザも容易にアプリケーションを公開することができる Softbank 用のアプリケーション「S!アプリ」の書式への変換を実現した。

## 1.2 論文構成

本論文の構成は以下の通りである。2 章では、この研究で変換の対象とする携帯電話や PHS 用のアプリケーションの開発の現状を解説する。3 章では、変換プログラムの現在の利用のされ方と実装について解説する。4 章では、自作のプログラムと書籍や Web などで公開されているプログラムを実際に変換した結果を考察する。5 章では、本研究のまとめを示す。

# 第 2 章

# 携帯電話等のアプリケーション開発

## 2.1 携帯電話やPHS用アプリケーションの種類

現在日本では、NTT DoCoMo、au、Softbank の 3 キャリアの携帯電話や WILLCOM の PHS がサービスを提供している [1, 2]。また、各キャリアともアプリケーションの実行ができる端末を提供している。しかし、各キャリアで利用できるアプリケーションは異なる。表 2.1 に各キャリアで利用できるアプリケーションの違いを示す。

NTT DoCoMo では、i アプリと i アプリ DX というアプリケーションが実行できる [18]。i アプリは、mova503i 以降及び FOMA シリーズで利用可能でインターネットの接続はアプリケーションをダウンロードしたサーバとのみ許されている。プログラムの容量は機種などにより 20KB〜500KB のものがある。i アプリ DX は、i アプリの機能に加え、携帯電話本体に保存されているアドレス帳などのデータを利用することができる。また、ダウンロードもとのサーバ以外のサーバとの通信も可能になっている。

au では、EZ アプリ (Java$^{TM}$)、EZ アプリ (BREW) とオープンアプリ (Java$^{TM}$) というアプリケーションが実行できる [8, 9, 10]。EZ アプリ (Java$^{TM}$) は、端末内のアドレス帳や LED、バイブレータなどを利用することが可能になっている。EZ アプリ (BREW) は、UDP・TCP サーバ、TCP クライアントとしての機能を持つ。また、端末内の型番や発着信履歴の表示、電池残量などの取得ができ、ネットワー

クを通して必要なプログラムをダウンロードする機能がある。また、データフォルダやカメラ機能との連動、外部メモリの利用、パソコンへのアクセスなども可能になっている。オープンアプリ (Java$^{TM}$) はオープンアプリプレイヤーという Java アプリケーション再生用 BREW アプリケーションを用いて実行できる。ユーザ保護のため一日の通信料に規制が行われているなどの制約がある。最近発売された端末では EZ アプリ (Java$^{TM}$) は実行できなくなっている。

Softbank では S!アプリというアプリケーションが実行できる [11]。主にゲームなどに利用されており、早くからゲーム関連の機能が充実していた。現在では、赤外線通信やモーションセンサーコントロールなどを利用したアプリケーションもある。プログラムの容量も 50KB〜1MB と比較的大きいサイズのアプリケーションがある。

WILLCOM では Java アプリというアプリケーションが実行できる [12]。独自の機能などはない。

## 2.2 携帯電話用アプリケーションの開発方法

各アプリケーションの開発環境は異なっている。表 2.2 に、各アプリケーションの開発環境の違いを示す。

i アプリ及び i アプリ DX の開発環境は携帯電話などの容量の小さい環境向けの Java 環境である Java 2 Platform, Micro Edition(J2ME) と携帯電話などのリソースが限られた機器を対象としたコンフィギュレーションである Connected Limited Device Configuration(CLDC)、NTT DoCoMo 用のプロファイルである DoJa プロファイルを用いて開発する。ローカルデータはスクラッチパッドと呼ばれる領域に保存することができる。一般のユーザも端末上で実行可能な i アプリの開発を行うことはできるが、セキュリティ上の問題から i アプリ DX の開発を行うことはできない。

EZ アプリ (Java$^{TM}$) も J2ME 及び CLDC を使うが Mobile Information Device Profile(MIDP) という標準的なプロファイルとこれを拡張する独自のプロファイル

を用いて開発する。一般ユーザも端末上で実行可能な EZ アプリ (Java$^{TM}$) の開発を行うことはできるが、セキュリティの問題で一般ユーザに公開されていない機能もある。

EZ アプリ (BREW) は、Java 言語を用いて開発を行うことが多い携帯電話用アプリケーションの中では珍しく C/C++言語で開発する。Java 言語と違いネイティブコードであるため、CPU ごとにコンパイルを行うことが必要である。現在端末上で実行可能なアプリケーションの開発は行えない。ただし、PC 上のエミュレータでの実行までは行うことができる。

オープンアプリ (Java$^{TM}$) は EZ アプリ (Java$^{TM}$) と同様に J2ME、CLDC 及び MIDP を用いて開発するが、EZ アプリ (Java$^{TM}$) の独自プロファイルなどは利用できない。オープンアプリプレイヤーを使って実行するため、利用できる機能に制限がある場合もあるが、一般ユーザも端末上で実行可能なオープンアプリの開発を行うことができる。

S!アプリは、J2ME、CLDC、MIDP と独自の Java$^{TM}$ 拡張 API 群である J-Phone Specific Class Libraries(JSCL) を用いて開発を行う。記憶領域には、クラスファイルとリソースファイルを圧縮した JAR ファイルと内容変更可能なレコードストアに割り当てられる。端末上で実行可能なアプリケーションの開発を行うことができる。

Java アプリはオープンアプリと同様に J2ME、CLDC 及び MIDP を用いて開発を行う。一般ユーザも端末上で実行可能なオープンアプリの開発を行うことができる。

このように、開発方法が異なるため、複数のキャリア用のアプリケーションを開発するためには、同じ挙動をするアプリケーションを開発する際にも設計の段階から別々に行う必要があり、非常にコストがかかる。

表 2.1: 各キャリアで利用できるアプリケーションの種類と特徴

| キャリア | アプリケーション | 特徴 |
|---|---|---|
| NTT DoCoMo | i アプリ | ・インターネットの接続はアプリケーションをダウンロードしたサーバとのみ可 |
| | i アプリ DX | ・携帯電話本体に保存されているアドレス帳などのデータを利用できる<br>・ダウンロードもとのサーバ以外のサーバとも通信可能 |
| au | EZ アプリ (Java$^{TM}$) | ・端末内のアドレス帳や LED、バイブレータなどを利用可能 |
| | EZ アプリ (BREW) | ・UDP・TCP サーバ、TCP クライアントとしての機能がある<br>・端末内の型番や発着信履歴の表示、電池残量などの取得できる<br>・ネットワークを通して必要なプログラムをダウンロードできる<br>・データフォルダやカメラ機能との連動、外部メモリの利用、パソコンへのアクセスなども可能 |
| | オープンアプリ (Java$^{TM}$) | ・一日に通信できるデータ量に制限がある<br>・独自の拡張などはない |
| Softbank | S!アプリ | ・ゲーム関連の機能が充実している<br>・赤外線通信やモーションセンサーコントロールなどを利用できる |
| WILLCOM | Java アプリ | ・独自の拡張などはない |

表 2.2: 各アプリケーションの開発環境

| アプリ | 開発環境 |
|---|---|
| i アプリ | ・Java 2 Platform, Micro Edition<br>・Connected Limited Device Configuration<br>・DoJa プロファイル |
| i アプリ DX | ・Java 2 Platform, Micro Edition<br>・Connected Limited Device Configuration<br>・DoJa プロファイル |
| EZ アプリ (Java$^{TM}$) | ・Java 2 Platform, Micro Edition<br>・Connected Limited Device Configuration<br>・Mobile Information Device Profile<br>・独自プロファイル |
| EZ アプリ (BREW) | ・C/C++言語 |
| オープンアプリ (Java$^{TM}$) | ・Java 2 Platform, Micro Edition<br>・Connected Limited Device Configuration<br>・Mobile Information Device Profile |
| S!アプリ | ・Java 2 Platform, Micro Edition<br>・Connected Limited Device Configuration<br>・Mobile Information Device Profile<br>・J-Phone Specific Class Libraries |
| Java アプリ | ・Java 2 Platform, Micro Edition<br>・Connected Limited Device Configuration<br>・Mobile Information Device Profile |

# 第 3 章

# 変換プログラムの開発

## 3.1 対象

本研究では、事業者別契約数 [17] の最も多い NTT DoCoMo 用アプリケーションを変換対象のアプリケーションとすることにした。この中で i アプリ DX は i アプリの機能を拡張したものであるので、基本である i アプリを選択した。次に、au 用アプリケーションの EZ アプリ (BREW) では実機での動作試験を行えないことから候補からはずした。また、契約数では au が Softbank を上回っているが、au 用アプリケーションの EZ アプリ (Java$^{TM}$) は、最新の機種で利用できなくなっていることから、Softbank 用アプリケーションの S!アプリを選択した。

変換もとのアプリケーションのプログラムはすでに完成して動作確認されているものを想定し、使用者は企業の開発者及び一般の開発者を問わないこととした。

変換できる機能は、i アプリの DoJa-1.0 プロファイルの中からアプリケーション制作の際に利用頻度の高い文字の描画、基本図形の描画、画像の描画、キー入力の認識を対象とし、これらの機能を S!アプリのプログラムで実行できる形に変換することを目標とする。

## 3.2　本プログラムの目的

図 3.1 に携帯電話用のアプリケーションの開発をウォーターフォール開発モデルを示す。通常の携帯電話用アプリケーションの開発は、発案、外部仕様決定、内部仕様決定、開発、動作試験の順に開発が進められる。

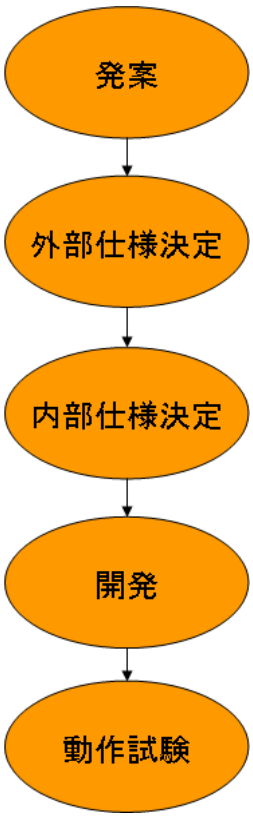


図 3.1: ウォーターフォール開発モデル

また、本研究で対象とする i アプリと S!アプリで同じ動作をするアプリケーションを開発する場合、どのようなアプリケーションを開発するかを考える発案の工程と、発案の工程で考えた動作をどのように実装するかを決定する外部仕様を決定する工程は同じなので、発案と外部仕様決定の工程を統一することができる。図 3.2 に i アプリと S!アプリで同じ動作をするアプリケーションを開発するウォーターフォール開発モデルを示す。

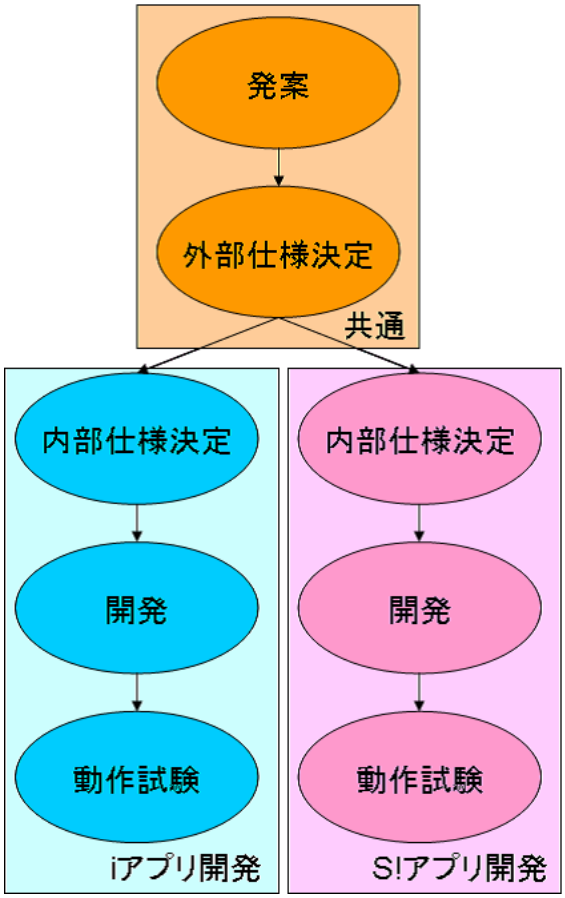


図 3.2: 複数の環境向けに開発を行う場合

本研究で開発する変換プログラムは、内部仕様決定、開発の工程を統一化することで作業工程を削減することを目的とする。図 3.3 に本研究の目的とする開発工程をウォーターフォール開発モデルで示す。

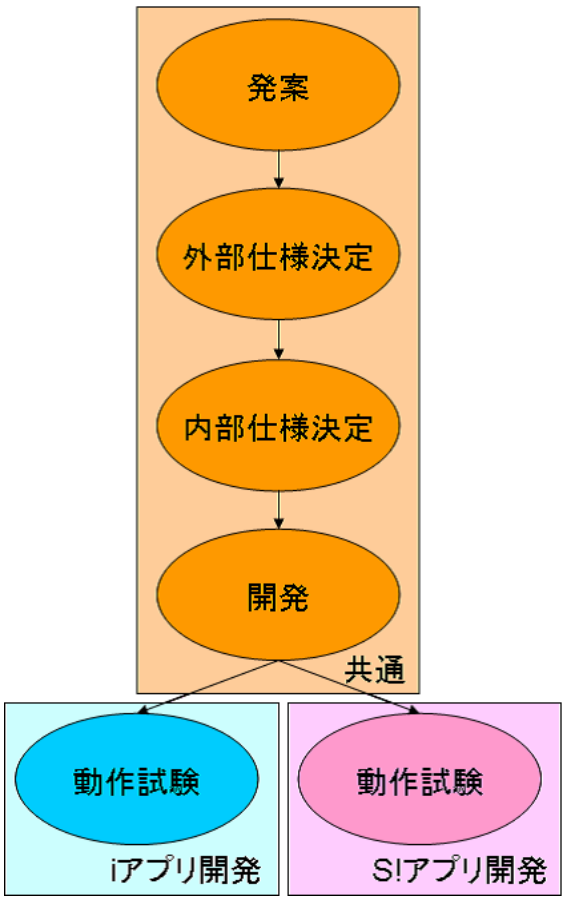


図 3.3: 目的とする作業工程

変換用プログラムを用いることで一方の開発の工程を省略することができると考えられる。また、一方の開発の工程がなくなることに伴い、内部仕様決定の工程も省略することができると考えられる。

## 3.3 実装方針

実装する方法として、以下の三種類の方法が考えられる。

1. i アプリ用のプログラムを S!アプリ用のプログラムへ変換
2. S!アプリ用のプログラムを i アプリ用のプログラムへ変換

3. 新たなスクリプト言語を作り、この言語を用いて開発したアプリケーションからiアプリとS!アプリ用のプログラムを生成

この中で、三つ目の方法は、開発者に新しい言語を覚える手間が増えるため、候補からはずす。また、iアプリの方がS!アプリに比べ書籍やWebなどで情報を集めやすく、現在開発を行っている人も多いことから本研究では、一つ目の手法を選択する。

次に変換方法として以下の三種類の方法が考えられる。

1. 構文解析を行い変換
2. メソッド呼び出しなどの変換の必要な場所を見つけて変換
3. DoJaプロファイルで用意されているクラスをS!アプリで使える形にしたものを利用

三つ目の方法は容量が多くなることが想定される。そのため、資源の限られた携帯電話用のアプリケーションには不向きである。また、一つ目の方法と比較して実装が容易と思われる二つ目の方法を選択する。

## 3.4 変換仕様

本研究で開発したプログラムでは以下のことを踏まえて実装を行った。

### 3.4.1 開始

iアプリとS!アプリでは、プログラム開始時に呼び出されるクラスのスーパークラスが異なる。また、必要なメソッドにも違いがある。表3.1に開始クラスの違いを示す。

開始するクラスは、iアプリではcom.nttdocomo.ui.IApplicationクラスを継承してつくり、startメソッドが最初に呼び出される。S!アプリではjavax.microedition.midlet.MIDPクラスを継承してつくり、startAppメソッドが最初に呼び出される。

表 3.1: 開始クラスの違い

| i アプリ | S!アプリ |
|---|---|
| com.nttdocomo.ui.IApplication<br>クラスを継承<br>start メソッドに記述 | javax.microedition.midlet.MIDP<br>クラスを継承<br>startApp メソッドに記述<br>pauseApp メソッドが必須<br>destroyApp メソッドが必須 |

また、S!アプリでは一時停止する際に呼ばれる pauseApp メソッドと終了時に呼び出される destroyApp メソッドの記述が必須である。

このことから、変換プログラムでは com.nttdocomo.ui.IApplication クラスを継承しているクラスを発見した際、javax.microedition.midlet.MIDP クラスを継承するように変更する。また、start メソッドを startApp メソッドに書き換え、resume メソッドや terminate メソッドがある場合は、それぞれに対応するメソッドに書き換え、ない場合は追加する。

### 3.4.2 低レベルインターフェイス

i アプリや S!アプリでは文字や画像などの描画を自分で処理する低レベルインターフェイスと、あらかじめ用意されたコマンドボタンやテキストボックスなどのコンポーネントを利用する高レベルインターフェイスがあり、本変換プログラムでは低レベルインターフェイスのみを対象とする。この低レベルインターフェイスを利用する際に宣言するクラスのスーパークラスが i アプリと S!アプリでは異なる。表 3.2 に低レベルインターフェイスの違いを示す。

i アプリでは com.nttdocomo.ui.Canvas クラスを継承して宣言する。S!アプリでは javax.microedition.lcdui.Canvas クラスを継承して宣言する。どちらも paint メソッドの記述が必須である。

このことから、変換プログラムでは com.nttdocomo.ui.Canvas クラスを継承しているクラスを発見した際、javax.microedition.lcdui.Canvas クラスを継承するよ

表 3.2: 低レベルインターフェイスの違い

| i アプリ | S!アプリ |
|---|---|
| com.nttdocomo.ui.Canvas<br>クラスを継承<br>paint メソッドが必須 | javax.microedition.lcdui.Canvas<br>クラスを継承<br>paint メソッドが必須 |

うに変更する。

### 3.4.3 色の指定

i アプリと S!アプリでは文字や図形を描画する際の色の指定に用いるメソッドが異なる。表 3.3 に色の指定方法の違いを示す。

表 3.3: 色の指定方法の違い

| i アプリ | S!アプリ |
|---|---|
| setColor(getColorOfName(int))<br>setColor(getColorOfRGB(int, int, int)) | setColor(r, g, b)<br>setColor(0x00RRGGBB) |

i アプリでは色の RED や GREEN などの定数値で指定する方法とレッド、グリーン、ブルーの値を (r, g, b) の形式で指定する方法がある。S!アプリではレッド、グリーン、ブルーの値を (r, g, b) の形で指定する方法と 16 進数で (0x00RRGGBB) の形式で指定する方法がある。

このことから、変換プログラムでは i アプリでどちらのメソッドを使って色を指定しても S!アプリでは 16 進数で指定する方に変換するようにする。

### 3.4.4 文字の描画

i アプリと S!アプリでは文字の描画に使うメソッドが異なる。表 3.4 に文字の描画方法の違いを示す。

表 3.4: 文字の描画方法の違い

| i アプリ | S!アプリ |
|---|---|
| drawString(文字列, x 座標, y 座標) | drawString(文字列, x 座標, y 座標, アンカーポイント) |

i アプリでは com.nttdocomo.ui.Graphics クラス、S!アプリでは javax.microedition.lcdui.Graphics クラスの drawString メソッドを利用する。i アプリでは、描画する文字列と表示する文字列の左下の座標を引数に取る。S!アプリでは、描画する文字列と表示する文字列の座標、座標の基準となるアンカーポイントの指定が必要である。

このことから、変換プログラムでは文字列描画の引数に左下を示すアンカーポイントを付け加える必要がある。

### 3.4.5 基本図形の描画

i アプリと S!アプリでは直線や矩形などの基本図形の描画方法はほぼ同じである。表 3.5 に基本図形の描画の違いを示す。

表 3.5: 基本図形描画の違い

| i アプリ | S!アプリ |
|---|---|
| com.nttdocomo.ui.Graphics をインポート | javax.microedition.lcdui.Graphics をインポート |
| drawLine(始点 x 座標, 始点 y 座標, 終点 x 座標, 終点 y 座標) | drawLine(始点 x 座標, 始点 y 座標, 終点 x 座標, 終点 y 座標) |
| drawRect (左上 x 座標, 左上 y 座標, 幅, 高さ) | drawRect (左上 x 座標, 左上 y 座標, 幅, 高さ) |
| fillRect (左上 x 座標, 左上 y 座標, 幅, 高さ) | fillRect (左上 x 座標, 左上 y 座標, 幅, 高さ) |

直線、四角、塗りつぶし四角の描画では、メソッドの名前や引数が同じである

ため、変換プログラムはインポートするクラスを com.nttdocomo.ui.Graphics から javax.microedition.lcdui.Graphics に変更するだけで利用できた。

### 3.4.6 画像の描画

i アプリと S!アプリでは利用できる画像の形式や画像を利用する方法が異なる。表 3.6 に画像描画の違いを示す。

表 3.6: 画像描画の違い

| i アプリ | S!アプリ |
|---|---|
| GIF 形式の画像が利用可 | PNG 形式の画像が利用可 |
| getImage メソッドで画像を取得<br>use メソッドで利用できる形式に変換<br>getImage メソッドで<br>イメージオブジェクトを取得 | createImage メソッドを利用 |
| drawImage(イメージオブジェクト,<br>x 座標, y 座標) | drawImage<br>(イメージ, x 座標, y 座標, アンカー) |

i アプリでは GIF 形式の画像、S!アプリでは PNG 形式の画像が利用できる。また、i アプリで画像を利用するには、com.nttdocomo.ui.MediaManager.getImage メソッドで画像を取得し、com.nttdocomo.ui.MediaResource.use メソッドで利用できる形式に変換し、com.nttdocomo.ui.MediaImage.getImage メソッドでイメージオブジェクトを取得する必要がある。S!アプリでは javax.microedition.lcdui.Image.createImage メソッドを利用するだけで画像が利用できる。実際に画像を利用する際には、i アプリでは com.nttdocomo.ui.Graphics.drawImage メソッドを利用し、S!アプリでは javax.microedition.lcdui.Graphics.drawImage メソッドを利用する。i アプリでは画像の左上の座標を指定のに対し S!アプリでは座標と座標の基準になるアンカーポイントを指定する必要である。

このことから、事前に画像形式を変更しておく必要がある。また、変換する際に i アプリでイメージオブジェクトを準備するメソッドを省く必要がある。さらに、

drawImage メソッドの引数にアンカーを付け足す必要がある。

### 3.4.7 キー入力

i アプリと S!アプリにはそれぞれ 2 種類のキー入力を認識する方法がある。一つ目は、ボタンが押されたときや離されたときに呼び出されるメソッドを利用する方法である。二つ目は、キーの状態を取得するメソッドを必要なときに呼び出して利用する方法である。前者の方法では、表 3.7 のフィールド値でボタンの判定をする。また、i アプリでは後者の方法でも表 3.7 のフィールド値を利用して判定することができるが、S!アプリではできない。しかし、後者の方法では、複数のボタンを同時に押したときの処理やゲームなどのアプリケーションを開発する際にループの中で常にチェックするなどの利用方法がある。

表 3.7: 各ボタンのフィールド値（一例）

| | i アプリ | S!アプリ |
|---|---|---|
| 0 | KEY_0 (=0) | KEY_NUM0 (=48) |
| 1 | KEY_1 (=1) | KEY_NUM1 (=49) |
| 2 | KEY_2 (=2) | KEY_NUM2 (=50) |
| 3 | KEY_3 (=3) | KEY_NUM3 (=51) |
| 4 | KEY_4 (=4) | KEY_NUM4 (=52) |
| 5 | KEY_5 (=5) | KEY_NUM5 (=53) |
| 6 | KEY_6 (=6) | KEY_NUM6 (=54) |
| 7 | KEY_7 (=7) | KEY_NUM7 (=55) |
| 8 | KEY_8 (=8) | KEY_NUM8 (=56) |
| 9 | KEY_9 (=9) | KEY_NUM9 (=57) |
| ← | KEY_LEFT (=10) | LEFT (=2) |
| ↑ | KEY_UP (=11) | UP (=1) |
| → | KEY_RIGHT (=12) | RIGHT (=5) |
| ↓ | KEY_DOWN (=13) | DOWN (=6) |
| 決定 | KEY_SELECT (=14) | FIRE (=8) |

変換プログラムでは、インポートするクラスや利用するメソッドの変換、フィールド値の変換を行うことでキー入力を利用できるようにした。

### 3.4.8 ダブルバッファリング

ゲームなどのアプリケーションでは、画面のちらつきを抑えるためにダブルバッファリングという処理を用いる。表 3.8 にダブルバッファリングの違いを示す。

表 3.8: ダブルバッファリングの違い

| i アプリ | S!アプリ |
|---|---|
| 明示的に行う | 自動的に行われる |

i アプリでは、lock メソッドと unlock メソッドを用いて明示的に行う必要がある。しかし、S!アプリでは、自動的に行われるので、この処理は必要ない。

# 第 4 章

# 検証

## 4.1 変換プログラムの実装

本研究では、3 章で示した方針を満たす変換プログラムを Windows 環境で C++ 言語を用いて開発を行った。C++クックブック [19] のアルゴリズムを参考に文字列の検索及び置き換えを行うことによりプログラムを変換した。

本変換プログラムでは、下記のように実行することで i アプリ用のプログラムから S!アプリ用のプログラムに変換する。

```
>parser 入力ファイル名 出力ファイル名
```

## 4.2 自作プログラム

付録 A に試験用に開発した図形を表示するアプリケーションのプログラムを掲載した。付録 A.1 は、変換元となる i アプリのプログラムである。付録 A.2 は、変換先の S!アプリのプログラムである。付録 A.3 は、変換プログラムを利用して付録 A.1 を変換したプログラムである。図 4.1 にこれらのプログラムの実行画面を図示する。


図 4.1: 実行画面

これらのプログラムは実行すると直線、長方形、塗りつぶされた長方形、画像、文字列を画面に描画する。また、起動後に上下左右ボタンを押すと星の画像が一定量ずつ移動し、決定キーを押すと初期位置に戻る。これらを用いて変換プログラムを検証する。

まず、作業工程は、内部仕様決定と開発の工程を省略することができたため、別々にやらなくてはならない工程は動作試験のみとなった。この結果、今回の試験用のプログラムの場合、別々にプログラミングする場合、i アプリと S!アプリのプログラムの合計 192 行から、i アプリのプログラム 101 行を書くだけになった。

次に、各々の場所がどのように変換されているかを比較する。

まず、testPaint.java を比較する。i アプリのプログラムで com.nttdocomo.ui.*をインポートしている文が変換後のプログラムでは javax.microedition.midlet.MIDlet と javax.microedition.lcdui.*をインポートしている。これは、利用しているメソッドにあわせて変換を行ったためである。また、クラスを宣言する際に i アプリのプログラムでは IApplication クラスを継承しているが、変換後のプログラムでは MIDlet を継承している。さらに、このクラス内のアプリケーションを実行した際に最初に実行する start メソッドを startApp メソッドに変換し、ディスプレイを設定するところも変換している。また、変換後のプログラムには i アプリのプログラムでは記述のない pauseApp メソッドと destroyApp メソッドの追加が確認できた。

次に、testCanvas.java を比較する。まず、testPaint.java と同様にインポートす

るクラスを利用しているメソッドにあわせて変換している。次に、色の指定について比較する。i アプリのプログラムでは getColorOfName メソッドを利用して、Graphics クラスで提供する定数によって設定しているのに対し、変換後のプログラムでは、16 進数の表記によって設定している。次に、文字列の描画を比較する。文字列の描画のメソッドでは、S!アプリに変換する際にアンカーの情報を追加している。次に、画像表示関係の行を比較する。i アプリのプログラムで宣言されている MediaImage 型の変数が変換後のプログラムでは宣言していない。また、画像を準備する部分も変更している。表示する部分の変換後のプログラムでは i アプリのプログラムの引数に加え、アンカーの情報を追加している。次に、キー入力について比較する。i アプリのプログラムでは processEvent メソッドでキーの状態を取得しているが、変換後のプログラムでは、keyPressed メソッドと keyReleased メソッドで状態の変化を受け取り processEvent メソッドに渡している。受け取ったキーの定数値はそれぞれ対応する定数値に変換している。

今回の試験用のプログラムでは、正常に動作するプログラムに変換することができた。また、手動で用意した S!アプリのプログラムの容量は 1.84KB、変換プログラムを利用して用意した S!アプリのプログラムの容量は 2.10KB と大幅な容量の増加はなかった。

## 4.3 公開されているプログラム

変換ツールの精度を確かめるため、Web ページに公開されているプログラムを 2 つ変換した。

一つ目は、株式会社エヌ・ティ・ティ・ドコモの Web ページ [7] に公開されている DoJa-1.5OE プロファイル向け i アプリ開発ツールにサンプルとして入っているテトリスのプログラムである。図 4.2 に携帯電話のボタンの例を示す。


図 4.2: 携帯用アプリケーションのボタン

本研究で開発した変換プログラムでは、数字キー、方向キー、決定キーの入力に関する処理には対応しているが、ソフトキーの入力には対応していない。i アプリと S!アプリのソフトキーの利用方法に以下の 4 つ違いがある。

1. 実装するインターフェイスの違い
2. キーアクションの受け取り方の違い
3. ラベルの設定方法の違い
4. キーの判定に用いる値の違い

実装するインターフェイスの違いは継承するクラスを変更する方法と同様の方法が考えられる。また、i アプリではキーが押されたときと離されたときに別々のメソッドが呼び出されるのに対して、S!アプリでは同じメソッドで処理する違いがあるが、数字キーなどの判定と同様の方法で対応できると考える。しかし、画面下部にソフトキーの機能などを表示するためのラベルを用意する際に、i アプリで

はどのソフトキーにどのラベルを設定するかを指定するのに対し、S!アプリでは表示する文字や設定するソフトキーの情報などを持ったクラスオブジェクトを使って設定する。また、i アプリのソフトキーの判定にはどのソフトキーが押されたかで判定するのに対し、S!アプリではラベルを用意する際に用いたクラスオブジェクトの値を用いて判定している。これらの違いを吸収するプログラムが用意できなかったためソフトキーの処理には未対応である。今回対象にしたテトリスのプログラムでは、このソフトキーを利用する仕様になっていた。そのほかにも、i アプリではスクラッチパッド、S!アプリではレコードストアというアプリケーションで利用できる不揮発性のメモリにデータを保存する処理など未対応の機能を含んでいた。また、外部クラスのメソッドの利用に未対応だったため、うまく変換できず、S!アプリの環境では実行できなかった。

二つ目は、i アプリの制作方法を紹介している Web ページ [20] に掲載されていたブロック崩しのプログラムである。こちらのプログラムでも、一つ目のテトリスのプログラムと同様にソフトキーの利用や外部クラスのメソッドの利用があったため、S!アプリの環境で実行できるプログラムに変換できなかった。

# 第 5 章

# まとめ

通常の携帯電話や PHS 用のアプリケーションの開発では、同じアプリケーションを開発する場合でも複数のキャリアで利用できるようにするためには別々に開発を行わなければならないため、人手や時間などの手間がかかる。本研究では、プログラム変換を行うことで開発工程を簡略化する手法を提案し、実際に変換プログラムを用意して検証した。

検証の結果、対応している機能のみを使っているプログラムの変換をする際には、製作の期間と容量があまり増えなかったことから、変換ツールの利用は有用だった。しかし、未対応の機能を含むプログラムではうまく変換できず、修正箇所を探したりする必要があるため、余計な作業が増えてしまった。

検証を行うために、Web ページや書籍に掲載されたプログラムを調べた結果、アプリケーション実行中にメニュー画面を開いたり、アプリケーションを一時停止したりするためにソフトキーを利用していることが多々あった。また、ゲームなどのアプリケーションでは、ハイスコアやセーブデータの保存をスクラッチパッドなどの領域に保存していたり、アプリケーション自体の容量を削減するために画像など容量の大きいデータをインターネット上に用意し必要に応じて通信を行って取得していたりした。今後は、これらのことを踏まえ、より多くの機能に対応し、プログラムの記述の仕方が異なっていても変換できるように実装していきたい。

# 謝辞

本研究を進めるにあたり、本校メディア学部の渡辺大地講師をはじめ多くの方のご指導を頂きました。心より深く感謝いたします。

さらに、本研究を進めるにあたり、協力していただいた研究室のメンバー及び友人に深く感謝いたします。

最後に、私を支えてくれた家族と、全ての友人に感謝いたします。

# 参考文献


[1] ウィキメディア財団, Wikipedia「携帯電話」,
<http://ja.wikipedia.org/wiki/%E6%90%BA%E5%B8%AF%E9%9B%BB%E8%A9%B1>.

[2] ウィキメディア財団, Wikipedia「PHS」,
<http://ja.wikipedia.org/wiki/PHS>.

[3] 山田 雅之, 鈴木 茂樹, ラフマット・ブディアルト, 遠藤 守, 宮崎 慎也,
“携帯電話を利用したコミック閲覧システムとその評価”,
芸術科学会論文誌 Vol.3 No.2, pp.149-158, 2004.

[4] 福井 良太郎, 白川 洋, 歌川 由香, 重野 寛, 岡田 謙一, 松下 温,
“携帯電話における歩行者ナビゲーション情報の表示方法に関する提案と評価”, 情報処理学会論文誌 Vol.44 No.12, pp.2968-2977, 2003.

[5] 斉田 祐二,
“携帯電話における教育用 Java アプリケーションの実態調査と試作”,
岩手県立大学ソフトウェア情報学部卒業論文, 2002.

[6] 阿部 昭博, 佐々木 辰徳, 小田島 直樹,
“位置情報を用いて地域コミュニティ活動を支援するグループウェアの開発と運用評価”, 情報処理学会論文誌 Vol.45 No.1, pp155-163, 2004.

[7] 株式会社エヌ・ティ・ティ・ドコモ,
作ろうiモードコンテンツ：iアプリコンテンツの概要,
<http://www.nttdocomo.co.jp/service/imode/make/content/iappli/about/index.html>.

[8] KDDI 株式会社, KDDI au: 技術情報 EZ アプリ (Java),
<http://www.au.kddi.com/ezfactory/tec/spec/ezplus.html>.

[9] KDDI 株式会社,
KDDI au: 公式コンテンツで提供するサービス EZ アプリ (BREW),
<http://www.au.kddi.com/ezfactory/service/brew.html>.

[10] KDDI 株式会社,
KDDI au: 技術情報 ¿ オープンアプリ (JavaTM),
<http://www.au.kddi.com/ezfactory/tec/spec/openappli.html>.

[11] ソフトバンクモバイル株式会社, ソフトバンク,
<http://developers.softbankmobile.co.jp/dp/tech_svc/java/>.

[12] 株式会社 ソフィア・クレイドル, 携帯 Java アプリ開発手順－携帯 Java 技術情報－,
<http://www.s-cradle.com/developer/java/development_java_appli.html>.

[13] 株式会社 ソフィア・クレイドル, DoJa と MIDP の相違点&アプリサイズ,
<http://www.s-cradle.com/developer/java/difference_carrier.html>.

[14] 株式会社 ソフィア・クレイドル, 携帯 Java アプリを BREW へ移植する方法,
<http://www.s-cradle.com/developer/brew/porting_javatobrew.html>.

[15] 石山 慎, 佐々木 準人, 渡辺 雅人, 新美 礼彦, 高木 剛, 小西 修, 宮本 衛市,
高橋 修,

“GPS 携帯電話を用いたマルチベンダ対応チャットとスケジューラの実装”,
人工知能学会第 20 回全国大会 (JSAI2006), 2006.

[16] 山内 健一, 金沢 忠, 小山内 勉, 前田 光司, 太齋 真吾,
“プラットフォーム移行診断サービスの実績と今後の展開”,
日立 TO 技報 第 10 号, p39-45, 2004.

[17] 社団法人電気通信事業者協会, 事業者別契約数,
<http://www.tca.or.jp/japan/database/daisu/yymm/0611matu.html>.

[18] 株式会社エヌ・ティ・ティ・ドコモ, i アプリ,
<http://www.nttdocomo.co.jp/service/imode/content/iappli/index.html>.

[19] D. Ryan Stephens, Christopher Diggins, Jonathan Turkanis, Jeff Cogswell,
C++クックブック, 株式会社クイープ, pp.121-182, 2006.

[20] バズワード, i アプリの作り方,
<http://www.javadrive.jp/iappli/game/6/index.html>.

# 付録 A

# 変換の例

## A.1 変換前

testPaint.java

```
import com.nttdocomo.ui.*;

public class testPaint extends IApplication {
  public void start() {
    Display.setCurrent(new testCanvas());
  }
}
```

testCanvas.java

```
import com.nttdocomo.ui.*;

public class testCanvas extends Canvas {
  Image image;
  MediaImage mi;
  int l, t;
  int x, y;

  public testCanvas() {
    try {
      mi = MediaManager.getImage("resource:///star.gif");
      mi.use();
      image = mi.getImage();
    } catch(Exception e) {}

    l = getWidth() / 2 - 60;
    t = getHeight() / 2 - 60;
```

```
    x = l + 60;
    y = t + 5;
  }

  public void paint(Graphics g) {
    g.lock();

    // 描画領域の初期化
    g.setColor(g.getColorOfName(g.BLACK));
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(g.getColorOfName(g.WHITE));
    g.fillRect(l, t, 120, 120);

    // 直線の描画
    g.setColor(g.getColorOfName(g.RED));
    g.drawLine(l + 5, t + 5, l + 5, t + 65);
    // 四角の描画
    g.setColor(g.getColorOfName(g.GREEN));
    g.drawRect(l + 10, t + 5, 20, 60);
    // 塗りつぶし四角の描画
    g.setColor(g.getColorOfName(g.BLUE));
    g.fillRect(l + 35, t + 5, 20, 60);
    // 画像の表示
    g.drawImage(image, x, y);
    // 文字列の描画
    g.setColor(g.getColorOfName(g.BLACK));
    g.drawString("Sample Program.", l + 5, t + 120);

    g.unlock(true);
  }

  public void processEvent(int type, int param) {
    // キープレスイベント
    if (type == Display.KEY_PRESSED_EVENT) {
      getKey(param);
    }
  }

  public void getKey(int param) {
    switch (param) {
      case Display.KEY_LEFT:
        // 左キーが押されたとき
        x -= 5;
        break;
      case Display.KEY_UP:
        // 上キーが押されたとき
```

```
        y -= 5;
        break;
      case Display.KEY_RIGHT:
       // 右キーが押されたとき
        x += 5;
        break;
      case Display.KEY_DOWN:
       // 下キーが押されたとき
        y += 5;
        break;
      case Display.KEY_SELECT:
       // 決定キーが押されたとき
        x = l + 60;
        y = t + 5;
        break;
    }

    // 描画領域をはみ出したとき
    if (x < l)
      x = l;
    if (x > l + 70)
      x = l + 70;
    if (y < t)
      y = t;
    if (y > t + 70)
      y = t + 70;

    repaint();
  }
}
```

## A.2 手動変換後

testPaint.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class testPaint extends MIDlet {
   public void startApp() {
     Display.getDisplay(this).setCurrent(new testCanvas());
  }
   public void pauseApp(){}
   public void destroyApp(boolean unconditional){}
}
```

testCanvas.java

```
import javax.microedition.lcdui.*;

public class testCanvas extends Canvas {
  Image image;

  int l, t;
  int x, y;

  public testCanvas() {
    try {
      image = Image.createImage("star.png");
   } catch(Exception e) {}

    l = getWidth() / 2 - 60;
    t = getHeight() / 2 - 60;

    x = l + 60;
    y = t + 5;
  }

  public void paint(Graphics g) {
    // 描画領域の初期化
    g.setColor(0x00000000);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x00FFFFFF);
    g.fillRect(l, t, 120, 120);

    // 直線の描画
    g.setColor(0x00FF0000);
    g.drawLine(l + 5, t + 5, l + 5, t + 65);
    // 四角の描画
    g.setColor(0x00008000);
    g.drawRect(l + 10, t + 5, 20, 60);
    // 塗りつぶし四角の描画
    g.setColor(0x000000FF);
    g.fillRect(l + 35, t + 5, 20, 60);
    // 画像の表示
    g.drawImage(image, x, y, Graphics.TOP|Graphics.LEFT);
    // 文字列の描画
    g.setColor(0x00000000);
    g.drawString("Sample Program.", l + 5, t + 120,
                 Graphics.BOTTOM|Graphics.LEFT);
  }
```

```
  protected void keyPressed(int keyCode) {
    switch (keyCode) {
      case LEFT:
       // 左キーが押されたとき
        x -= 5;
        break;
      case UP:
       // 上キーが押されたとき
        y -= 5;
        break;
      case RIGHT:
       // 右キーが押されたとき
        x += 5;
        break;
      case DOWN:
       // 下キーが押されたとき
        y += 5;
        break;
      case FIRE:
       // 決定キーが押されたとき
        x = l + 60;
        y = t + 5;
        break;
    }

    // 描画領域をはみ出したとき
    if (x < l)
      x = l;
    if (x > l + 70)
      x = l + 70;
    if (y < t)
      y = t;
    if (y > t + 70)
      y = t + 70;

    repaint();
  }
}
```

## A.3 自動変換後

testPaint.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;
```

```
public class testPaint extends MIDlet {
  public void startApp() {
    Display.getDisplay(this).setCurrent(new testCanvas());
  }
public void pauseApp(){}
public void destroyApp(boolean unconditional){}
}
```

testCanvas.java

```
import javax.microedition.lcdui.*;


public class testCanvas extends Canvas {
  Image image;

  int l, t;
  int x, y;

  public testCanvas() {
    try {
      image = Image.createImage("star.png");
    } catch(Exception e) {}

    l = getWidth() / 2 - 60;
    t = getHeight() / 2 - 60;

    x = l + 60;
    y = t + 5;
  }

  public void paint(Graphics g) {
    // 描画領域の初期化
    g.setColor(0x00000000);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x00FFFFFF);
    g.fillRect(l, t, 120, 120);

    // 直線の描画
    g.setColor(0x00FF0000);
    g.drawLine(l + 5, t + 5, l + 5, t + 65);
    // 四角の描画
    g.setColor(0x00008000);
    g.drawRect(l + 10, t + 5, 20, 60);
    // 塗りつぶし四角の描画
```

```
    g.setColor(0x000000FF);
    g.fillRect(l + 35, t + 5, 20, 60);
    // 画像の表示
    g.drawImage(image, x, y,Graphics.TOP|Graphics.LEFT);
    // 文字列の描画
    g.setColor(0x00000000);
    g.drawString("Sample Program.", l + 5, t + 120,
                  Graphics.BOTTOM|Graphics.LEFT);
  }

  public void processEvent(int type, int param) {
    // キープレスイベント
    if (type == 0) {
      getKey(param);
    }
  }

protected void keyPressed(int keyCode) {
  processEvent(0, keyCode);
}
protected void keyReleased(int keyCode) {
  processEvent(1, keyCode);
}

  public void getKey(int param) {

    switch (param) {
      case LEFT:
       // 左キーが押されたとき
       x -= 5;
       break;
      case UP:
       // 上キーが押されたとき
       y -= 5;
       break;
      case RIGHT:
       // 右キーが押されたとき
       x += 5;
       break;
      case DOWN:
       // 下キーが押されたとき
       y += 5;
       break;
      case FIRE:
       // 決定キーが押されたとき
       x = l + 60;
       y = t + 5;
```

```
        break;
    }

    // 描画領域をはみ出したとき
    if (x < l)
      x = l;
    if (x > l + 70)
      x = l + 70;
    if (y < t)
      y = t;
    if (y > t + 70)
      y = t + 70;

    repaint();
  }
}
```