スマートフォンに特化したマルチプラットフォーム開発手法の提案

毛受 崇洋† 福田浩章†
Takahiro Menju† Hirotaki Fukuda†
†芝浦工業大学　工学部情報工学科
† Information Engineering, Shibaura Institute of Technology.

要旨

スマートフォンには iOS や Android などのプラットフォームが存在している．スマートフォンで動作するアプリケーション(以下アプリ)はスマートフォンのプラットフォーム毎にプログラミング言語や API が異なる．このため同じアプリでも異なるプラットフォームで開発を行う場合，それぞれのプラットフォームに合わせて作りなおさなくてはならない．

本研究では，独自のプログラミング言語と API を提供する．アプリの開発にはイベントハンドラを書く部分やボタンなどを表すコンポーネントがあるなどの特徴が存在し，その特徴をまとめて API にすることによって共通化を行う．

1. はじめに

近年，スマートフォンが流行しており，スマートフォンで動作するアプリケーション(以下アプリ)の個数が増加している．またスマートフォンには iOS や Android などのプラットフォームが存在している．[1]によるとスマートフォンのプラットフォームごとの所有者の割合は Android OS が 56.7%，iOS が 39.6%である．一つのプラットフォームに合わせて開発を行うと，半分程度のスマートフォンのユーザに対してしかアプリを供給することが出来ない．このことから，多くののユーザに開発したアプリを利用してもらうためには，それぞれのプラットフォームに合わせてアプリを開発しなくてはならないことが分かる．しかし，アプリは Android の場合 Java，iOS の場合 Objecitve-C と具合にプラットフォーム毎にプログラミング言語や API(Application Programming Interface)が異なる．このため，同じアプリでも異なるプラットフォームで開発を行う場合，それぞれのプラットフォームに合わせて作りなおさなくてはならない．そのため，複数のプラットフォームに対応した開発を行うマルチプラットフォーム開発が注目を集めている[2]．スマートフォンのマルチプラットフォーム開発手法に関しては HTML５/JavaScript といった WEB の技術を用いる方法が存在する．しかし動作速度などの問題が存在する[4].

一方スマートフォンにはイベントハンドラを書く部分やボタンなどを表すコンポーネントがあるなど，Android,iOS 共に特徴が存在している．しかし，Android の NFC のように，一つプラットフォームにしか存在しない API やハードウェアなどが存在している．

本研究では，スマートフォンの開発の特徴を踏まえ，独自のプログラミング言語と API を提供することでマルチプラットフォーム開発手法の提案を行う．プラットフォームで共通な処理をする部分と，プラットフォーム毎に異なる処理をする部分に分割してプログラムを記述し，それぞれのプラットフォーム固有のソースコードを自動生成する．プラットフォームで共通な処理をする部分は独自のプログラミング言語で，プラットフォーム毎に異なる処理をする部分はそれぞれのプラットフォームの言語で記述する． この手法により，例えば独自言語でコンポーネントのボタンが押されたときの処理を書くことで，両方のプラットフォームでのボタンが押された時の処理を記述できる．

2. 背景と問題点

この節では，既存のマルチプラットフォーム開発手法の問題点とスマートフォンアプリ開発の特徴について説明する．また本研究ではスマートフォンのプラットフォームで共通な処理をする部分と，プラットフォーム毎に異なる処理をする部分に分割してプログラムを記述する．その 2 つを柔軟に織り込む方法としてアスペクト指向プログラミングを利用しているため，アスペクト指向プログラミングを説明する

2.1. マルチプラットフォーム開発手法

マルチプラットフォーム開発の既存手法として，主に 3 つの手法が存在する．1 つ目は HTML5/JavaScript(以下 WEB の技術)で開発する手法, 2 つ目はモデル駆動アーキテクチャ[3]を用いて開発する手法, 3 つ目は AndroidNDK[4]

を用い，C 言語を利用して開発する手法である．

表 1　既存の開発手法の比較

開発手法	プラットフォーム言語	実装の生成	アルゴリズム以外の記述
WEB の技術を用いる	×	○	○
モデル駆動を用いる	○	×	○
C 言語を用いる	○	○	×

WEB の技術を用いる手法は，動作速度が遅いこと，ハードウェアアクセスの制限があることといった問題がある[5]．実際に Facebook の iOS に対してのアプリは動作速度の改善のために WEB の技術を用いたアプリから Objective-C を用いたアプリに書き直され，多数の好意的なレビューが寄せられた．

モデル駆動を用いる手法は，実装を抽象化したモデルからソースコードを生成することによって行う．まずプラットフォーム非依存モデルを最初に作成し，それぞれのプラットフォームに合わせてプラットフォーム依存モデルを自動生成する．生成したプラットフォーム依存モデルからプログラムを自動生成する事によって行う．しかし，モデルにソースコードレベルの情報を記述できないため，実装を完成させるためにはモデルから生成されたソースコードに対して書き加える必要がある[6].

C 言語を用いる手法では共通言語を利用できる事によって共通部分を一つのソースコードで記述できる．しかしプラットフォーム毎に API が異なるためアルゴリズムやビジネスロジックなどといった共通する部分以外を記述することが困難であり，一部の共通部分に対してしか共通化が出来ない．

2.2. アプリの開発手法の特徴

iOS や Android の開発には次の 3 つにあげるような特徴がある．1 つ目はボタンや編集可能なテキストなどの UI(User Interface)が用意されている，2 つ目はボタンをクリックした時に実行されるイベントハンドラを記述する必要がある，3 つ目は MVC アーキテクチャを利用しているなどという特徴がある．表 2 では実際に特徴の 1 つ目と 2 つ目が使われている．

表 2　アプリのソースコードの例

iOS	Android

- (IBAction)onClick:(id)sender {
int et1=textEdit1.text.intValue;
int et2=textEdit2.text.intValue;
[textView3 setText:[NSString stringWithFormat:@"%d",et1+et2]];
}　①　②　③
	public void onClick(View view) {
	int et1 = Integer.parseInt(editText1.getText().toString());
	int et2 = Integer.parseInt(editText2.getText().toString());
	textView3.setText(Integer.toString(et1+et2));
	}　①　②　③

表 2 で示すように，言語によって文法は異なるが API に似ている部分が存在する．例えば表 2 の①のように textView3 というコンポーネントに対して，表 2 の②の setText メソッドを呼ぶことで文字列を設定可能になっているなどである．

しかし，Android の NFC のように，あるプラットフォームにしか存在しない API やハードウェアなどが存在しているため，異なる部分も存在する．

2.3. アスペクト指向プログラミング

ロギングやセッション管理などのような非機能的な処理をオブジェクト指向言語では上手くモジュール化することができない．アスペクト指向プログラミングではモジュール間にまたがる横断的関心事をクラスとは別にアスペクトとしてモジュール化することができる．またアスペクトは織り込み(weave)を行うことによって横断的関心事に対応する．織り込むことが可能である場所をジョインポイントと呼び，ジョインポイントを選ぶ記述をポイントカットと呼ぶ．

3. 提案と実装

3.1. アプローチ

本研究では Android, iOS を対象とする．

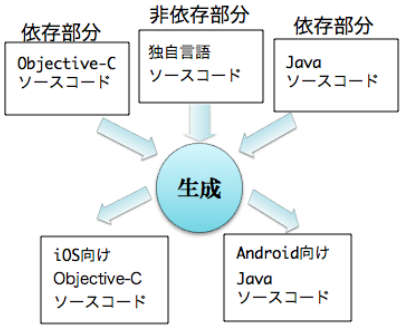

図 2　アプローチの概略

図 2 に示すようにプラットフォーム毎に異なる処理をする部分（以下依存部分）とプラットフォーム毎に同じ処理をする部分（以下非依存部分）に分割しプログラムを記述し，それぞれのプラットフォームに合わせてソースコードを生成する．非依存部分は独自言語で，依存部分はそれぞれのプラットフォームの言語で記述する．API としてはボタンやテキストエディトなど Android と iOS で同じように提供されているコンポーネントやそれらを扱う API を提供する

3.2. 既存の開発手法との比較

表 3　本研究の開発手法

開発手法	プラットフォーム言語	実装の生成	アルゴリズム以外の記述
本研究	○	○	○

本研究の手法によって表 3 に示すように表 1 で挙げたそれぞれの問題点を解決する．本研究は最終的にプラットフォームに依存したアプリケーションを開発するための言語で生成する．また独自言語ではモデルと異なりソースコードレベルで記述できるため実装も生成できる．アルゴリズム以外も API を提供することによって一つのソースコードで記述することが可能となる．

3.3. 実装

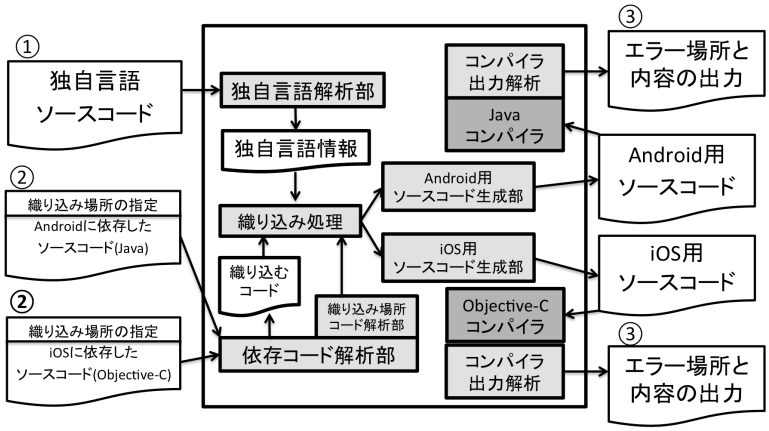

図 3　実装の概略

細かな実装としては図 3 のようになっている

3.3.1. 提供する独自言語

図 3 の①では独自のソースコードを用いてソースコードを記述する．この独自言語は基本的には Java の文法に則っている．Java にした理由は主に 2 つある．まず 1 つ目に Android で採用されていること，2 つ目にオブジェクト指向言語であり，その部分において iOS で利用されている Objecitve-C と似ている部分が存在するためである．

3.3.2. 提供する API

スマートフォンの特徴となる部分を API として提供する．例えばボタンや編集可能なテキストなどの UI(User Interface)のコンポーネントや，ボタンをクリックした時に実行されるイベントハンドラを用いることができるよう API として提供する. 提供する API によって, 行なっていることが同じでも, プラットフォーム毎に異なる処理を, 一つのソースコードで記述することが可能となる.

3.3.3. プラットフォーム依存部分

図 3 の②では独自言語に織り込むためにアスペクト指向プログラミングを利用する．独自言語内のどこに織り込むのか, アスペクト指向プログラミングで記述するための言語である AspectJ のポイントカットのような形により, 織り込み場所の指定を行う.

これによりメソッドの呼び出しの前や後にソースコードを織り込むことが可能である．しかしメソッドの途中などに柔軟に織り込むことには不十分であるので，独自言語内にアノテーションをつけられるようにし，ポイントカットでアノテーションを付けた場所を織り込む先を指定することができるようにする．これによりメソッド内の一部だけプラットフォーム毎に異なる処理を書きたい場合にも柔軟にプラットフォーム依存部分に書くことができ，織り込むことが可能となる.

3.3.4. エラー出力

図 3 の③では織り込まれたソースコードをそれぞれの言語のコンパイラでコンパイルし，コンパイラの出力を解析し，エラーの場所から織り込み元のコードの場所を出力する．この表示によって織り込まれた後のソースコードを見ずに開発を行うことができる.

3.3.5. 処理の流れ

図 3 の①の独自言語ソースコードを字句解析や構文解析(parse)をすることによって抽象構文木を構築する(以下独自言語構文木). その次に図 3 の②の織り込み場所の指定とプラットフォームに依存したソースコードを解析し，織り込むコードと織り込み場所の指定に分離する．独自言語構文木に対して織り込み場所の指定通りに織り込むコードを織り込む．織り込まれた独自言語構文木に対してそれぞれのプラットフォームに合わせてソースコードの生成を行う.

4. 今後の予定と評価

現在独自言語と API の実装中である.

複数のケーススタディによって評価を行う．各ケーススタディでは, 提案する開発手法を用いて開発を行う．評価方法としては, 提案する開発手法を用いた場合と, 用いなかった場合それぞれのコード行数を比較し, またそれぞれのコードについて考察を行う.

参考文献

[1] 株式会社ディーツーコミュニケーションズ,”スマートフォン普及動向調査”,2012
[2] Andre Charland, Brian LeRoux: Mobile Application Development: Web vs. Native,2011
[3] Richard Soley and the OMG Staff Strategy Group:Model Driven Architecture,2000.
[4] AndroidNDK,http://developer.android.com/tools/sdk/ndk/index.html
[5] Reto Meier, Michael Mahemoff:HTML5 versus Android: Apps or Web for Mobile Development?,Google I/O 2011
[6] Bertrand Portier, Lee Ackerman:Model Driven Development Misperceptions and Challenges,2009.