Vol.2015-SE-187 No.20
2015/3/12

iPhone アプリと Android アプリの同時開発支援システム

村山優弥[†1]　　紫合治[†2]

本論文では iPhone と Android アプリケーションを同時に作成する手法とツールについて報告する．まず，Android と iPhone のユーザインタフェースが類似していることに着目し，見た目や動作が似通っている箇所をツール上では同一のものとした．詳細の違いはツール独自の規格に当てはめることで共通化を行った．見た目や動作の類似点から共通化を行っているため本ツールでの開発は共通化された GUI(Graphical User Interface)によって,視覚的に設計を行ったのち，システムが自動的に Android と iPhone のソースコードを生成する．生成したソースコードを eclipse や Xcode などの各開発環境に渡すことでアプリケーションを完成させる．

Simultaneous development support system for Android and iPhone applications

YUYA MURAYAMA[†1]　　OSAMU SHIGO[†2]

This paper presents a method and a tool to develop a smart phone application both on Android and iPhone simultaneously. The common graphical elements for Android and iPhone, like button, label and text, are defined as the GUI tool elements to design the application user interface. The tool automatically generates the corresponding source codes in Java for Android and in Objective-C for iPhone simultaneously, which will be transferred to the eclipse for Android and the Xcode for iPhone to build the applications.

1. はじめに

近年のスマートフォン市場は不安定なものとなってきている．2014 年 5 月から 7 月のスマートフォン販売シェアは日本では Android が 69%，iOS が 29%と Android が優勢である．しかしながら 2014 年 1 月から 3 月での調査では Android が 41.5%iOS が 57.6%と iOS 側が優勢となっている(図 1). [1]

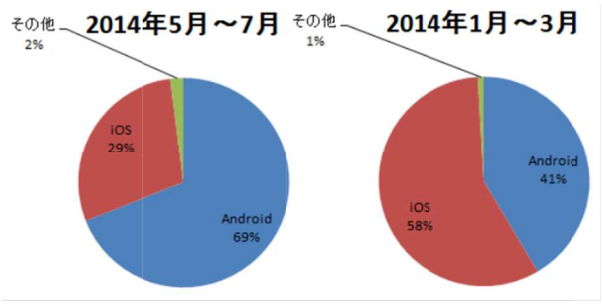

図 1 日本のスマートフォン市場

このように現在の日本でのスマートフォンシェアは流行や情勢に大きく左右されやすい．時期によって新規契約するユーザーの比率が多く異なるが，機種変更し OS を乗り換えるユーザーも少なくなく，日本では Android と iOS のどちらが優勢であり安定していると断言することはできない．そのため片方の OS にしか対応しないアプリケーションではユーザーの多くを逃すことになる．また，アプリケーションに興味があるのに OS の違いによって手に取ることが無いまま終わってしまう潜在的ユーザーも存在する．このような事態に陥らないためにもアプリケーションは Android と iOS の両方に対応していることが望ましい．

しかし，OS や開発環境,開発言語の違いから Android アプリケーション，iOS アプリケーション間では互換性を持たせることが難しく，両方に対応したアプリケーションを作成するには，通常はそれぞれの方法で一から作らなくてはならない．そこで本研究では 2 つの異なるアプリケーションを同時に開発するシステムを提案する．

2. 研究背景

ここでは Android アプリケーションと iOS アプリケーションを同時に作成するシステムとその手法について述べる．異なる二つのアプリケーションを同時に作成することによって開発の二度手間を無くし開発者の負担を減らすことを目的とする．

(1) Android と iPhone の相違点

Android と iPhone は同じスマートフォンであれどもその

†1 東京電機大学情報環境学研究科
Graduate School of Information Environment, Tokyo Denki University.
†2 東京電機大学情報環境学部
School of Information Environment, Tokyo Denki University.

仕組みは大きく異なっている．Android は Google が中心に開発を行っているスマートフォン用の OS，Android を搭載した携帯端末のことである．Android アプリケーションの開発は統合開発環境 eclipse を使用し Android SDK をプラグインとしてインストールすることによって作成を行うことが可能になる．開発言語は java 言語を用いる．[2]

一方 iPhone は，アップル製のスマートフォンで OS は Mac 系列の iOS が搭載されている．iPhone アプリケーションは Objective-C 言語で統合開発環境 Xcode を使用し作成する．[3]

この二つのアプリケーションは種類が全くの別物であり，開発環境も異なっているため Android アプリケーションと iPhone アプリケーションは安易に移植を行うことができない．双方のスマートフォンで使えるアプリケーションを作成するためには，同一のアプリケーションをそれぞれの環境で一から作成する必要がある．

(2) 関連研究

Android と iPhone 双方のアプリケーションを作成する既存のシステムとして Titanium Mobile[4]や PhoneGap[5]が例として挙げられる．

Titanium Developer は Appcelerator 社が提供しているオープンソースのソフトウェア開発環境である．HTML と JavaScript の二つの言語を用いて iPhone と Android 両方のプラットフォームに向けてモバイルアプリケーションを開発することができる．Titanium API は iOS と Android OS 固有の API へのアクセスを抽象化するミドルウェアになっており，ソースコードのほとんどを iPhone と Android 間で共用可能である．

PhoneGap はアドビシステムズによって提供されているオープンソースのクロスプラットフォームモバイルアプリ開発フレームワークである. PhoneGap はアプリケーション自体が Web サイトのような構造をしている．HTML と JavaScript,CSS などがアプリケーション内部に埋め込まれておりアプリケーションに内蔵したブラウザがそれらを読み取る．

Titanium と PhoneGap は共に HTML と JavaScript 言語で Android と iPhone 双方のアプリケーション開発を行っている．また，開発環境が完全に独立しており，本来の開発環境である Eclipse や Xcode を使用しない．そのためアプリケーション開発者が従来の開発手法を活かすことができない．

それに対し，本研究ではツールで作成したアプリケーションの基盤を基に java 言語と Objective-C 言語の二つソースコードを生成することで本来のアプリケーション開発の環境である Eclipse と Xcode をサポートする形で双方のアプリケーションを開発する．これにより開発者が開発環境や言語を変えることなくアプリケーションが作成できる．

3. システムの構成

(1) システムの概要

本システムでは GUI でのアプリケーション開発が主となる．これは全く性質の異なる Android と iPhone を共通化するのにあたって最も類似している見た目を軸としてアプリケーションを作成するためである．Android と iPhone で見た目や動作が類似している箇所をツール上で 1 つにまとめ共通化し，視覚的に表すことで一度の作業で双方のアプリケーションを同時に開発することができる．

共通化した GUI で作成したアプリケーションの基盤を基にシステムが自動で Android と iPhone で同一の動作を行うソースコードを java と Objective-C でそれぞれ生成する．出来上がったソースコードを各開発環境に渡すことでアプリケーションを完成させる．

(2) システムの流れ

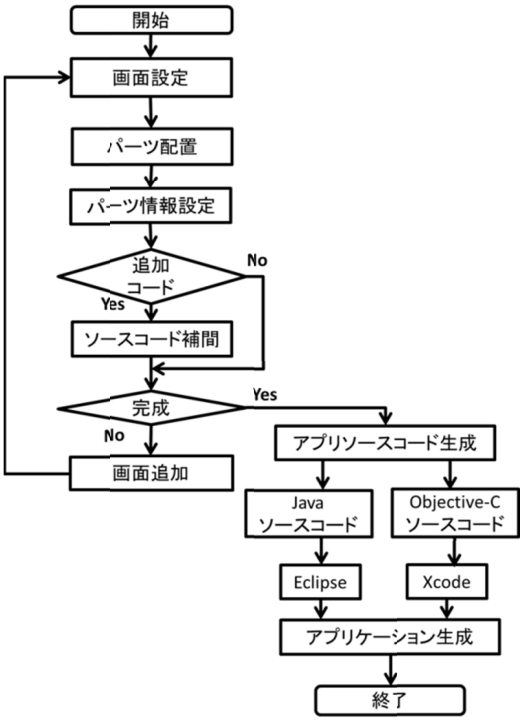

図 2 アプリケーション開発の流れ

本システムによるアプリケーション開発の流れを図 2 に示す. 本システムでは GUI が開発の中心になるためスマートフォンの画面にアプリケーションのパーツを貼り付けていくようにしてアプリケーションを作成する．まず土台となるスマートフォンの画面の設定を行う．その後，画面にボタンやラベルなどのアプリケーションのパーツを配置していき，パーツ毎に構成やイベント処理の設定を行う．その際，GUI だけでは表せない処理がある場合はソースコードを記述する．一画面以上作成したい場合は画面を追加し同様の工程を繰り返す.

すべての画面にパーツの配置と設定が完了したらソースコードへと変換する．生成された java と Objective-C のソースコードをそれぞれの開発環境である eclipse と Xcode に渡すことでアプリケーション制作が完了する．

4. ツール概要

iPhone アプリと Android アプリ同時開発支援ツール「Dorufon」の全体図を図 3 に示す．

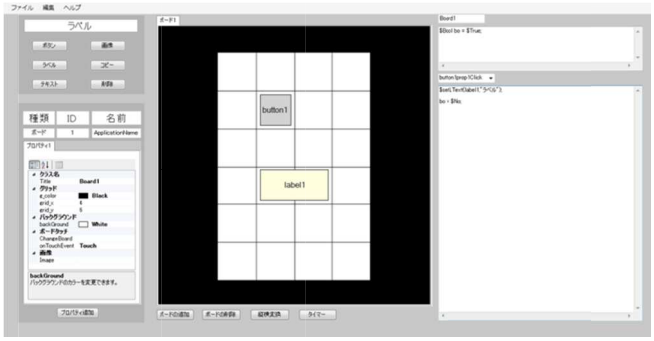

図 3 Dorufon 全体図

(1) ボード

図 3 の黒縁で囲まれたところがアプリケーション開発の中心となるスマートフォンの画面に当たる部分でボードと呼ぶ．このボードはスマートフォンの画面であると同時に一つのクラスとしても扱う．そのためボードに付けた名前がそのままクラスファイル名となる．

ボードは追加することができ，ボードごとに異なる設定を行うことができる．

(2) グリッド

ボード上に張り巡らされた網目状の線をグリッドと呼び本ツールの尺度になる．グリッドはスマートフォン画面の height と width をそれぞれ決められた数で分割したものとなっており，グリッドの数値はボードの設定から変更することができる．グリッドによって分割され出来たマス目が一目盛りとなり，ボード上に貼り付けるアプリケーションのパーツの位置と大きさは図 4 のようにすべてこのグリッドの目盛りを基準に生成される．グリッドの目盛りは図 4 のようにボードの左上が原点となり右下に進むほど値が大きくなる．アプリケーションのパーツの height と width はマス目の左上を原点に生成され，ボード同様右下に進むほど値が大きくなる．図 4 の例では X 座標が一目盛り，Y 座標が一目盛り，width が二目盛り，height が一目盛りのボタンをグリッド(4,4),(4,6),(8,8)のボードで生成した例である．図 4 のように同様の目盛りを持つアプリケーションのパーツであってもグリッドの値を変えることで実際のサイズが変動する．グリッドは色の変更や非表示にすることも可能だがアプリケーションに反映される際は比率だけでグリッド線そのものは表示されない．このグリッドを基準としたアプリケーションの開発によりレイアウトを統一し，画面の大きさが異なるスマートフォンであってもすべて同一の比率でアプリケーションを開発することができ，Android と iPhone の画面の共通化を可能にしている．

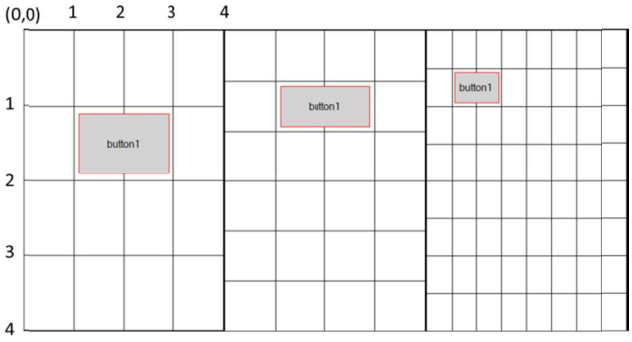

図 4 グリッド数によるサイズの違い

(3) ウィジェット

ボードに貼り付けるボタンやラベル等のアプリケーションのパーツを Dorufon ではウィジェット（図 5）と呼ぶ．ウィジェットは表 1 の様に Android のオブジェクトと iPhone の UI パーツを共通化させたツール独自の存在となっている．Dorufon ではアプリケーションの表示用部品の基本となるボタン，テキスト表示，編集テキスト，画像表示の 4 つを中心に Android と iPhone の共通化を行った．

表 1 Dorufon の規格による共通化

Dorufon	Android	iPhone
ボタン	Button	UIButton
ラベル	TextView	UITextView
テキスト	EditText	UIEditText
画像	ImageView	UIImageView

Dorufon で生成されるボタンウィジェットは Android では Button，iPhone では UIButton にあたり，ボタンを押した時の処理を設定することができる．テキスト表示を行うラベルウィジェットは java では TextView クラス, Objective-C では UITextView クラスに変換される. テキストウィジェットは編集テキストを指し，Android の EditText，iPhone の UIEditText に対応する．画像ウィジェットは生成時には無色透明で黒い枠線で囲まれた状態だが，設定でファイルから画像を選択することでボード上の画像ウィジェットに画像を表示することができる．ソースコードに変換した際は java では ImageView クラスに, Objective-C では UIImageView クラスとして生成される．

ウィジェット生成の手順は作成したいウィジェットを

選択し，ボード上で貼り付けたい場所をクリックすることで縦横一目盛りのウィジェットを生成することができる．生成時にドラッグすることでウィジェットの大きさを変えることができる．位置と大きさ以外の設定はデフォルトの設定が施される．設定を変えることでボード上に表示されているウィジェットも同様に変化する．位置と大きさはボード上からウィジェットをドラッグすることでも変更することができる．ウィジェットには ID が種類ごとに割り振られている．異なるボードで生成されたウィジェットであっても ID は共通である．ウィジェットの ID は、Android では id，iPhone では tag に対応する．

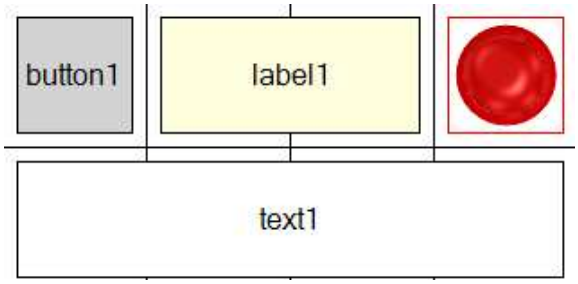

図 5 ウィジェット

(4) プロパティ

ボードやウィジェットに設定を行う箇所を本ツールではプロパティ（図 6）と呼ぶ．プロパティには Android と iPhone の要素を共通化したツール独自の設定が一覧で表示されている．プロパティでは構成の設定だけでなくイベント処理の設定も行うこともできる．

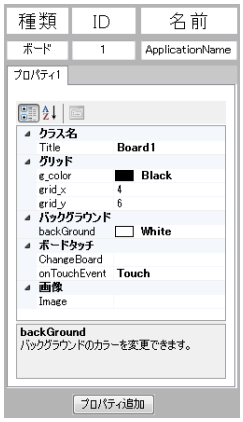

図 6 ボードのプロパティ

プロパティは共通プロパティと個別プロパティの二種類がセットで存在する．共通プロパティは ID と名前を設定でき一つのボード, ウィジェットに対して一つ存在する．一方複数プロパティにはそれ以外の設定を行うことができ，表示される要素は種類によって異なる．個別プロパティは共通プロパティと違い，一つのウィジェットに対して複数個持たせることができる．この二つのプロパティにより図 7 のようにボード上に存在する一つのウィジェットに異なる複数の設定を行うことができるようになる．個別プロパティはタブで切り替えることができ，ボード上のウィジェットは現在選択されているプロパティの設定が反映される．複数の設定を切り替えることでウィジェットの状態変化を表すことができる．

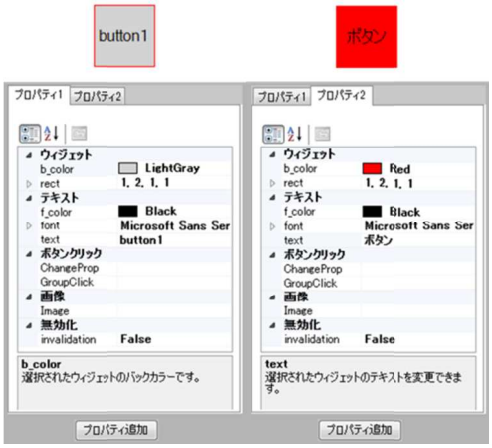

図 7 ウィジェットの複数の個別プロパティ

ボードは共通プロパティでボード全体をまとめ，個別プロパティの部分でそれぞれのボードの設定を行う．プロパティには現在選択されているボードの設定が表示される．ボードの個別プロパティを切り替えることによって画面遷移を表すことができる．

これらの個別プロパティの切り替えはボードやウィジェットのプロパティに存在するイベント処理(アクション)から呼び出したいプロパティを選択することで行うことができる．

(5) 変数登録欄

図 3 の右上に存在するテキストボックスを変数登録欄と呼び，変数宣言のソースコードを記述することができる．変数登録欄は変数を宣言したい場合に使用する．変数登録欄はボードごとに存在し，ボードを切り替えると連動して変数登録欄も切り替わる．ここで記述したコードはソースコード生成の際に所定の位置に組み込まれる．

(6) 処理登録欄

図 3 の右下に存在するテキストボックスを処理記述欄と呼び GUI では表せないウィジェットやボードのイベント処理（ボタンを押した時の処理等）をソースコードを記述する．処理記述欄はボードやウィジェットのプロパティごとに存在しそれらを生成すると自動的に作成される．イベント処理が複数存在する場合，ウィジェット生成時に同時に必要な数だけ自動で生成される．また，ウィジェットのプロパティに存在するイベント処理の Group から新たに名前を付けて処理登録欄を作成することができる．Group で

作成された処理登録欄は別のウィジェットの Group でも選択することができ，一つの処理登録欄で複数のウィジェットに共通した処理を記述することができる．処理登録欄はコンボボックスから名前を選択することで切り替える．処理記述欄はソースコード変換時には一つのメソッドとしてソースコード中に組み込まれる．メソッドの宣言はツール上で自動的に行うため，処理記述欄にはその中身である処理のみを記述する．

(7) ソースコード変換

これらすべての設定と配置が完了したら，メニューバーのファイルにある Android 変換，iPhone 変換を選択することにより本ツールで作成したアプリケーションの基盤を java と Objective-C のソースコードファイルに変換することができる．変換には Android，iPhone それぞれのテンプレート(後述)が使用される．

Android 変換を選択することで生成される java ファイルはボードの数だけ作成され，ボードの複数プロパティで設定した名前がクラス名となる．ボードの共通プロパティの名前はパッケージ名となる．

iPhone 変換を選択すると生成される Objective-C のソースコードファイルはボードの数だけ.m ファイルと.h ファイルの二つのファイルがボードの複数プロパティで設定した名前で生成される．それに加えて AppDelegate.h ファイル ViewController.h ファイルと ViewController.m ファイルが自動で追加生成される．

これらのクラスファイルを各開発環境に渡すことで本システムによりアプリケーション開発が完了する．

5. 共通化手法

本ツールでは Android と iPhone を共通のものとして作成するにあたって様々な手法を用いている．

(1) GUI による共通化

図 8 の様にボードやウィジェットで似通っている要素をツール上の GUI で共通化し独自の規格でアプリケーションを作成する手法が本ツールの軸となる手法であるが，この手法だけではアプリケーションの動作を表現することができない．

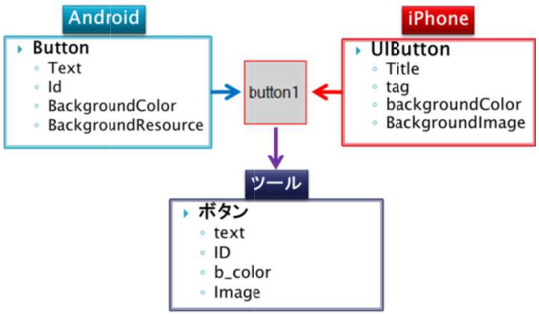

図 8 Dorufon の規格による各要素の共通化

(2) 複数の個別プロパティによる処理表現

複数の個別プロパティはツール上で表現できるアプリケーションの幅を増やすために存在する．複数の個別プロパティは一つのウィジェットに複数の設定を持たせることができるというものだが，この複数のプロパティの切り替えをアプリケーション上でも行うことでウィジェットの状態の変化を表現することができる（図 9）．

個別プロパティの切り替えは個別プロパティ内にあるイベント処理の設定から行うことができる．イベント処理内のボタンを押すことで図 10 の切り替えウインドウが新たに開き，ウィジェットの切り替えを設定できる．切り替えウインドウに存在する項目は Dorufon 上に存在するプロパティすべてが一覧で表示される．アクションに応じて呼び出したいプロパティの状態を指定することで，アプリケーション上でアクションを行った際に設定の切り替えが起き，ウィジェットの状態を変化させることができる．プロパティの切り替えは他のウィジェットや同一のボードに存在しないウィジェットのプロパティもまとめて選択することができる．プロパティ同士の変化を連結することによってウィジェットの変化を Android と iPhone を共通の形としてツール上で表現できる．

アプリケーションの画面遷移を表現したい場合はウィジェットのイベント処理から移動先のボードプロパティを選択することで表すことができる．

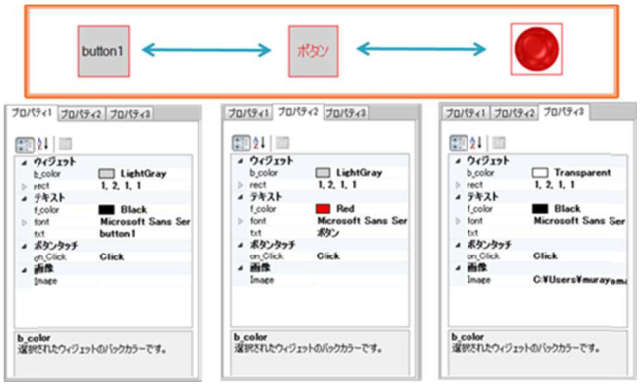

図 9 個別プロパティ切り替えによる変化

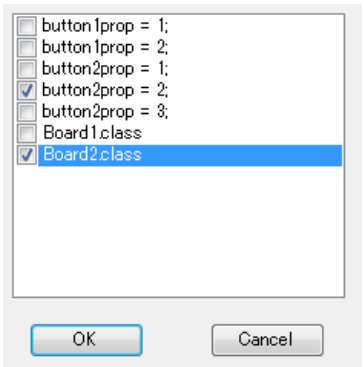

図 10 プロパティ切り替え

(3) 共通コードによる記述サポート

本ツールの機能を使ってソースコードを書かずにアプリケーションを作成することが本システムの軸であるが，それだけでは表現しきれない処理が出てくる．その場合に処理記述欄で補足を行うが，この際使用する言語に問題が生じる．java と Objective-C は元が C 言語であるためある程度同様の書き方を行うことができる．しかしメソッド呼び出しや Android と iPhone のそれぞれ独自の機能はコードが全く異なっている場合もある．処理記述欄でも Android と iPhone のコードを同一視し共通化を行っているため，共通のコードを使用できる場合はそのまま記述できるが java と Objective-C で異なるコードを記述する場合は二つの言語で記述する必要が出てくる．この負担を減らすために java と Objective-C の一部のコードを共通化した共通コードを作成した．共通コードは変数登録欄と処理記述欄で使用することができる．

共通コードは各ソースコードへの変換時に自動で java と Objective-C に対応したコードに変換される．共通コードは変数に関わるものと，本ツールのウィジェットに関するコードを中心に行った．

```
$String(name,"太郎");
$Int(i,0);
```

図 11 変数宣言の共通コードの例

図 11 は変数宣言時の共通コードの例である．共通コードのルールとして使用する際に必ず先頭に$を付ける．そしてその次の単語の頭文字は必ず大文字で記述する．変数宣言の独自コードはこのルールに加え最後を括弧でくくり第一引数で変数名を宣言，第二引数で初期化を行う．

共通コードには同一の意味を持つコードが存在する．図 12 のブーリアン型の独自コードがその例である．

```
$Bool(flag,$Yes);
$Boolean(flag,$Ture);
```

図 12 同一の意味を持つ共通コード

ブーリアン型は$Bool と$Boolean,$Yes と$Ture,$No と$False それぞれ java と Objective-C のどちらの書き方でも記述することができる．これは本ツールでのソースコードの記述方法が java と Objective-C の二つの言語を意識しながら記述することになるので，似通っていながらも異なる単語を使用するブーリアン型は記述する際に間違いが生じしやすかったためである．$Bool も$Bookean もソースコード生成時にはそれぞれの言語に合わせたコードに変換されるので，$Boolean で宣言した変数の初期値で$No を使用することもできる．

図 13 はウィジェットに関する共通コードとその変換結果の例である．使用する方法はルールの基本である先頭の$の後に，操作の種類を選択，値を代入したいなら set，取得したいなら get を記述する．次にウィジェットの種類を頭文字は大文字にして記述する．この時使用する単語はツール基準のものとする．次に操作対象プロパティを同様に記述する．その後括弧でくくり第一引数でウィジェット名を選択する．値を代入する場合は第二引数で記述する．
これらをまとめたものがウィジェットに関する独自コードとなる．

共通コードにはこのほかにも片方のソースコードにしか出力したくないコードを記述する場合，先頭に$a（Android 専用）又は$i（iPhone 専用）を宣言することでソースコード変換時にその行を対応する言語にのみ出力するといったこともできる．本ツールの GUI でも独自コードでも対応しない処理を記述する場合や片方の言語にしかない処理を記述する場合はこのコードを使い java と Objective-C を分けて記述する．

図 13 ウィジェットに関する共通コードの例

6. 変換方式

ここでは作成したアプリケーションの基盤を各ソースコードに変換する方式を説明する．本ツールではプロパティの値を Android テンプレート（図 14），iPhone テンプレート（図 15）にそれぞれ代入し変数登録欄，処理記述欄と組み合わせることで各ソースコードへと変換する．

```
@SuppressLint("DrawAllocation")
class $0$$View extends View{//ボード名
        private Bitmap image1; //背景画像があるときのみ
        private int dW, dH; //dW=ディスプレイの幅, dH=ディスプレイの高さ

        @SuppressWarnings("deprecation")
        public $0$$View(Context context) {//ボード名
                super(context);
                setBackgroundColor(0x$2$);
                Resources r=context.getResources();
                image1=BitmapFactory.decodeResource(r, R.drawable.$1$);//背景画像
                //画面のサイズの取得
                WindowManager   wmWindowManager = (WindowManager) context.getSystemService(Context.WINDOW_SERVICE);
                Display display = wmWindowManager.getDefaultDisplay();
                dW = display.getWidth();
                dH = display.getHeight();
        }
```

図 14 Android テンプレートの例

テンプレートの中身はアプリケーションのソースコードを分割したものになっている．代入を行う箇所は$で囲まれた数字で穴埋めされており，それぞれ対応したプロパテ

ィの値が入る．テンプレートは必要に応じて繰り返し使われるものもある．プロパティに何も値が入力されていない場合はテンプレートに対応する行が出力されない．現在 Android テンプレートは全 19 個，iPhone のテンプレートは全 26 個存在している．

```
#import "AppDelegate.h"

-(void)$0$
[

    //背景色
    self.view.backgroundColor = [UIColor colorWithRed:$1$ green:$2$ blue: $3$ alpha:$4$];

    //背景画像
    UIGraphicsBeginImageContext(self.view.frame.size);//$6$
    [[UIImage imageNamed:@"$6$"]drawInRect:self.view.bounds];//$6$
    UIImage *backgroundImage = UIGraphicsGetImageFromCurrentImageContext();//$6$
    UIGraphicsEndImageContext();//$6$
    self.view.backgroundColor = [UIColor colorWithPatternImage:backgroundImage];//$6$
```

図 15 iPhone テンプレートの例

これらのテンプレートはメソッドごとではなく役割に応じて分割されている．Android テンプレートでソースコードを作成する場合の流れが図 16 である．初めに package 文， import 文に加え，生成したソースコードの使い方の説明とクラスの宣言，ボードの背景画像，背景色,スマートフォンの画面サイズの取得とグリッドの設定を行うコードが一つのテンプレートとなる．その次にウィジェットを宣言するテンプレートを組み合わせる．このテンプレートはボード上に作成したウィジェットの数だけ繰り返される．この後に変数登録欄で記述したソースコードが組み合わさる．その次のテンプレートは宣言したウィジェットをレイアウトに貼り付けるコードでこれもウィジェットの数だけ繰り返されたのち，次のテンプレートに移行する．その次のテンプレートは宣言したウィジェットの要素をプロパティの数だけ繰り返し設定する．そのあとにそれぞれのウィジェットのイベント処理のテンプレートが組み合わされる．このテンプレートはコンボボックスに存在するメソッドの数だけ繰り返される．そしてこのテンプレートの中に処理記述欄のソースコードが組み込まれる．最後にこのボードクラスを閉じるテンプレートが組み合わさり一つのボードのソースコードファイルが完成する．この例が大まかなソースコード作成の流れである．実際はこれらのテンプレートに加え，各クラスを閉じるテンプレートやプロパティで設定した機能に応じたテンプレートが必要に応じて組み込まれる．この流れを Dorufon で作成したボードの数だけ繰り返すことでアプリケーション全体のソースコードが完成する．

iPhone テンプレートでソースコードを作成する場合はボード一つにつき.m ファイルと.h ファイルの二つを作成する．それに加え AppDelegate.h ファイル ViewController.h ファイルと ViewController.m ファイルを作成する必要があるので Android のテンプレートより数が多くなっている．5 種類のファイルの内，中心となるのがボードの.m ファイルである．.m ファイル作成の流れを図 17 で示す．

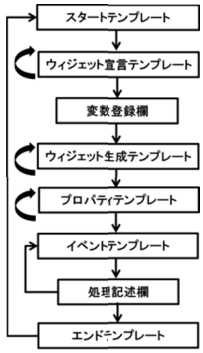

図 16Android テンプレートによるソースコード生成の流れ

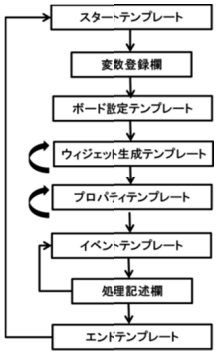

図 17 iPhone テンプレートによるソースコード生成の流れ

.m ファイルのスタートテンプレートでは import の宣言し変数登録欄を組み込んでいる．次のテンプレートで UIView である initWithFrame クラスを作成しその中にボードの設定を代入する．続けてその次のテンプレートをボード上のウィジェットの数だけ繰り返し，ウィジェットを View に配置する. その次のテンプレートはそのウィジェットのプロパティの数だけ繰り返し，個別プロパティごとの設定を行う．次のテンプレートで Android の時と同様にイベント処理のメソッドをその数だけ繰り返し生成し，その中に処理記述欄のソースコードを組み込む．最後にクラスを閉める end テンプレートを組み合わせることで.m ファイルが完成する．.h ファイルは最初のテンプレートで import の宣言と@interface コンパイラディレクティブを使用し，ボードプロパティの Title を代入することでクラスを宣言する．その次のテンプレートは現在のボード上に貼り付けたウィジェット数だけ繰り返し，ウィジェットの宣言を行う．その後 end テンプレートを組み合わせて.h ファイルが完成する．このようにテンプレートの中には他のファイルと共通し使用することのできるものも存在する．AppDelegate.h ファイルの作成流れは最初に UIApplicationDelegate の宣言と UIWindow の宣言をしたのち次のテンプレートで個別プロパティを切り替えるための変数を Dorufon 上すべてのボードで作成したウィジェット及びボードの数だけ繰り返す．最後に end テンプレートで閉め AppDelegate.h が完成する．iPhone アプリケーションのファイルには AppDelegate.m ファイルが存在するが本シ

ステムではこのファイルは生成しないためテンプレートは存在しない. ViewController.h のテンプレートでは import と UIViewController の宣言の後に Dorufon 上で作成したボードをその数だけ繰り返し宣言する．最後に end テンプレートで閉め ViewController.h が完成する．ViewController.m では import と ViewController の宣言の後にインスタンス化された直後に呼び出される viewDidLoad メソッドの中にボード 1 を最初に表示される画面として View に追加する．end テンプレートで閉じ, ViewController.m が完成したら iPhone アプリケーションのソースコードが完成する．

7. 検証と評価

本システムで実際にアプリケーションを作成し，本システムの特徴的な要素である Android と iPhone を共通化する 3 つの手法をそれぞれ検証し評価した．今回作成したアプリケーションは二つ，一つは脱出ゲームで複数の個別プロパティの切り替えを使いソースコードを書かずにツールにある機能のみを使い作成した．もう一方のアプリケーションは簡易電卓でこちらは複数プロパティを使わず単一のプロパティと共通コードを使用し作成した．また，これら二つのアプリケーションを, 本システムを使わず作成し, GUI による共通化の有効性を検証した．

結果, 第一の共通化手法である GUI による共通化は非常に有効であると判明した. java と Objective-C のソースコードを同時に作成することで開発時間の短縮を行うだけではなく，アプリケーションを開発する際の開発者の意識にも変化が生じた．従来の方法で別々にアプリケーションを作成する場合は必ずどちらか一方のアプリケーションを作成してからもう片方のアプリケーション開発に取り掛かるため，Android アプリケーションと iPhone アプリケーションを共に開発するというより片方のアプリケーションの動作に合わせて作成するといった意識でアプリケーション開発を行っていた．一方 Dorufon を使用した場合，GUI によって外見から統一しているのでどちらか一方に先入観を持つことなく双方のアプリケーションを開発者の意識から共通化してアプリケーション開発を行うことができた．

第二の共通化手法である複数の個別プロパティによる処理表現のみで作成した脱出ゲームはツール上で視覚的にウィジェットの状態変化の確認が行えるため，比較的容易にアプリケーションを作成することができた．しかし，すべてのプロパティに動作前と動作後の設定を行わなければならず，アプリケーション作成に時間がかかってしまった．また，ウィジェットやプロパティの数が増えるほど前後関係が不明慮となり管理が困難になることが判明した．

第三の共通化手法である共通コードによる記述サポートを使い開発した電卓はウィジェットのアクション処理を一つの処理記述欄にまとめ使いまわすことができるので，複数の個別プロパティを使用する際に一つ一つに設定を行わなければならない手間を省くことができた．しかし，共通コードを使用するにはアプリケーション開発者が java と Objective-C でコードが異なっている部分を理解しなくてはならず，第一，第二の手法のように Android と iPhone の違いを意識することなくアプリケーションを開発することができない（例えば数値のフォーマット変換方式等）もあったが，概ね共通化できた．電卓の処理記述 22 行の内，Android 専用と iPhone 専用の行は 7 行であり，ほぼ共通化できたといえる．

8. おわりに

本論文では Android アプリケーションと iPhone アプリケーションを同時に作成するために Android と iPhone をシステム上で共通化する三つの手法を述べた．GUI による視覚的共通化により開発者の負担を減らしアプリケーションを開発することに成功した．複数の個別プロパティによる共通化ではツール上でウィジェットの状態変化を指定できるようにし，開発できるアプリケーションの幅を増やした．共通コードによる共通化は第一、第二の手法では手間がかかったり，表すことができなかった処理を記述することができた．

しかしながら各手法には欠点が存在する．今後の課題として GUI と複数の個別プロパティと共通コードそれぞれの利点を組み合わせ欠点をカバーする仕組みを，検討していきたい．

参考文献

1)株式会社カンター・ジャパン　カンター・ワールドパネル・コムテック
2014.9.04【ニュースリリース】
http://kantar.jp/whatsnew/2014/09/kantarjapan_pr_0904.html
2014.5.08【ニュースリリース】
http://kantar.jp/whatsnew/2014/05/kantarjapan_pr_0508.html
2)eclipse
https://eclipse.org/downloads/
3) Xcode
https://developer.apple.com/jp/xcode/downloads/
4) Titanium Mobile Application Development
http://www.appcelerator.com/titanium/
5) PhoneGap Fan
http://phonegap-fan.com/