Write Once, Taste Twice: Webアプリと同一ソースでモバイルアプリを生成する手法

楊　建星[†]　白石和彦[†]
新井イスマイル[††]　西尾信彦[†††]

Webアプリケーションの高機能化は目覚ましいが，それらの多くはPCのWebブラウザを対象としており，スマートフォンに代表されるモバイル端末では，主にパフォーマンス，入出力インターフェース，プラグインなどの問題によってWebアプリケーションの利用が困難になってしまう．そのため，本研究ではWebアプリケーションをモバイル端末で利用可能とするため，Webアプリケーションのモバイルスタンドアロンアプリケーション化を行った．プロトタイプ実装として，GWTとAndroidを対象プラットフォームとするクロスプラットフォーム開発環境を構築した．

Write Once, Taste Twice: The Way to Generate Mobile App Using Same Source Code as Web App

Kensei YO,[†] Kazuhiko SHIRAISHI,[†] Ismail ARAI[††] and Nobuhiko NISHIO [†††]

While Web applications become more advanced, most of them are designed for PC Web browsers despite the proliferation of mobile clients such as smartphones. We have observed that one may face performance issues, ugly user interface, unsupported browser plugins and other problems when they try to use some Web applications on a mobile client. To solve the problems, we propose a way to translate a Web application to an equivalent mobile stand-alone application. As a prototype implementation of the translation mechanism, we implementd a cross-platform development environment for Google Web Toolkit and Android.

1. はじめに

近年，多くのWebブラウザがHTML5へ段階的に対応してきたことやマッシュアップなどのWeb2.0技術が盛んになってきたことにより，Webブラウザ上で動作するWebアプリケーション(Webアプリ)の多機能化・高性能化が目覚ましい．また，スマートフォンに搭載されているWebブラウザ(モバイルWebブラウザ)の多くがJavaScriptやHTML5の一部に対応してきたことにより，モバイルでのWebアプリの利用も現実的になってきた．しかし，未だに多くのWebアプリは，モバイルWebブラウザでの利用が困難である．例えばGoogle Street ViewのようなFlashアプリケーションは，プラグインが存在しないモバイルWebブラウザでは表示できず，他のWebアプリの多くもパフォーマンスや操作性に課題を残している．

そこで本研究では，モバイルWebアプリのモバイルスタンドアロンアプリケーション(モバイルアプリ)化に着目した．モバイルアプリにはクロスプラットフォームを実現するための開発コストがWebアプリよりも高いという問題が存在するものの，マルチタッチのようなプラットフォーム固有の操作や機能に対応可能であるという大きな利点がある．実際に地図アプリケーションの多くやGoogle Street View ☆がモバイルアプリとしても構築されている．これらの状況を踏まえて我々は，Webアプリとモバイルアプリを同一ソースで開発可能とするクロスプラットフォーム開発環境を提案する．提案する開発環境を利用することで，

† 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†† 立命館大学総合理工学研究機構
The Research Organization of Science and Engineering, Ritsumeikan University
††† 立命館大学情報理工学部
Department of Computer Science, Ritsumeikan University

☆ http://maps.google.com/intl/ja/help/maps/streetview/

開発者の負担を軽減しつつ，プラットフォームごとの機能も追加可能なアプリケーションの生成を目指す．

本稿の構成を以下に示す．まず，2 章で Web アプリをモバイル端末で利用する場合に起こりうる問題を提起する．次に，3 章で本研究が目指すクロスプラットフォーム開発環境について述べる．4 章では提案する開発環境についての設計を行い，5 章で設計に基づくプロトタイプ実装について述べる．6 章で提案する開発環境と関連する研究を比較し，最後に 7 章で結論と今後の課題についてまとめる．

2. Web アプリをモバイル端末で利用する際の問題

Web アプリはその多くが PC の Web ブラウザ (PC ブラウザ) からの利用を想定しており，モバイル端末で利用する際には以下のような問題が発生する．

- **端末スペック**

Web アプリの高機能化によって，動作スペックへのハードルは日増しに高まっている．文献 1)2) では，Web ページをサーバで画像化することによって，端末スペックに依存しない Web ブラウジングの高速化を行っているが，これらは Web アプリまでは考慮していない．近年ではモバイル端末の高性能化も進んでいるが，未だに PC と同様の JavaScript を動作させるには十分ではない．

- **入力インターフェース**

多くの場合，PC の主な入力インターフェースであるマウスの代わりとして，モバイル端末にはタッチパネルが搭載されている．タッチパネルは直感的な操作と入力/出力の一体化による端末の小型化が可能であり，携帯端末における入力インターフェースの主流となりつつある．しかし，タッチパネルとマウスのインターフェースデザインは完全に一致しているわけではないため，タッチパネルの入力をマウスの入力と同様に処理しても，同様の処理結果が得られるとは限らない．例えば，タッチパネルで画面をなぞる操作には，Web アプリに対するドラッグ操作と画面ごとのスクロール操作が対応している．そのため，タッチパネルの入力とマウスの入力を同様に処理した場合，マウスによる入力処理を想定して開発された Web アプリでは，入力によってユーザの意図した通りの動作が反映されない可能性がある．

また，タッチパネルのタップ (指で軽く叩く) 操作をマウスのクリックと対応させる例では，マウスのクリックよりも大きい面積がタップされることになるため，Web アプリのボタンが小さい場合に誤操作がしばしば起こる．

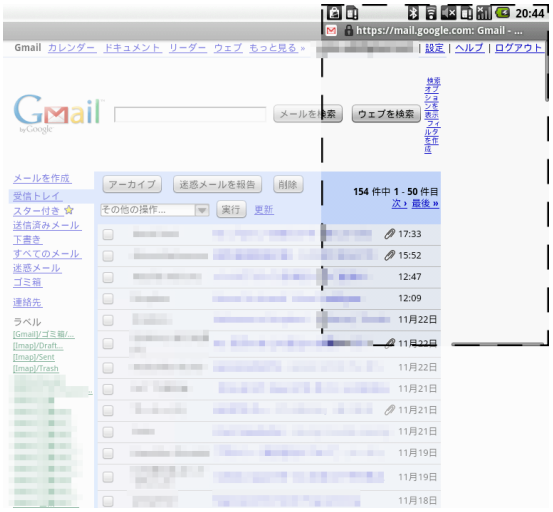

図 1 PC ブラウザ向け Web アプリをモバイル Web ブラウザで表示させた場合

- **画面デザイン**

Web アプリは CSS を利用することによって異なる画面サイズに対応することが可能である．しかし，モバイル端末の画面サイズは PC よりもかなり小さいため，CSS のみによるアプローチでは画面デザインの崩れを招いてしまう恐れがある．例えば Gmail ☆をモバイル Web ブラウザから利用する場合，図 1 のように全体の一部しか見ることができず，所々で画面デザインが崩れてしまう．

最近登場してきているモバイル Web ブラウザには Web ページの拡大縮小をサポートしているものがある．これにより PC ブラウザと同じ Web ページを画面デザインの崩れなく表示することが可能となってきている．このアプローチでは Web ページ全体を大まかに把握できるため，Web ページの閲覧には効果を発揮すると思われる．しかし，Web ページの頻繁な拡大縮小操作は煩わしく，モバイル端末の画面サイズに対応した Web ページの方が使い易い．

一部の Web アプリではこの画面デザインの問題について，独自にモバイル端末用に CSS を用意し，それを適用することにより改善している．先ほどの Gmail の例では，図 2 のようなモバイル端末の画面デザインに最適化した Web アプリを用意することによって，画面デザインの崩れを防いでいる．しかし，そのような改善策は現状では個々に対応しているにとどまって

☆ http://mail.google.com/

図 2 モバイル Web ブラウザ向け Web アプリを表示させた場合

おり，根本的な解決に至っていない．

- **プラグイン**

PC ブラウザには多くのプラグインが存在している．2010 年 5 月にモバイル OS の Android ☆がバージョン 2.2 から Flash のサポートが発表されたことにより，PC ブラウザのプラグインをモバイル Web ブラウザで利用することが現実的になってきた．実際に Android2.2 を搭載したモバイル端末で Flash の挙動を試したところ，いくつかの Flash 広告が PC ブラウザ大差のないパフォーマンスで見ることができた．しかし，一部の Flash は表示することができず，全ての Flash に対応できていないことがわかった．また，表示できた Flash の中には，マウスオーバーやマウスホイールのようにモバイル端末では物理的にほぼ不可能な操作をもつものもあり，そのため，処理内容によって Flash が利用できるかどうかは大きく変わってきてしまう．

また，現状ではその他にも多くのプラグインがモバイル Web ブラウザに対応できておらず，それらのプラグインを用いた Web アプリケーションは未だにモバイル端末で利用することができない．

3. クロスプラットフォーム開発環境

本研究では前章の問題を解決するために，Web アプリとモバイルアプリの同一ソースによるクロスプラットフォーム開発環境の構築を行う．

モバイルアプリに着目した理由としては，モバイル端末の入出力インターフェースに最適なアプリケーションを作成可能である点や PC ブラウザのプラグイン問題もモバイルアプリ開発時に機能を実装することによって解決できる点が挙げられる．また，本開発環境を利用することにより，開発者はモバイル Web アプリとモバイルアプリを同時生成することが可能となり，これにより，端末スペックや入出力インターフェースに合わせたアプリケーションの構築が容易となり，Web アプリのモバイル端末への対応にかかる開発コストの増加も最小限の抑えることが期待できる．更にユーザに対しては，端末環境やユーザのニーズに合わせた最適なアプリケーションを選択する自由を提供できるようになる．

本章では本開発環境を実現する上で求められる機能要件とそれを解決するためのアプローチについて考察する．

3.1 機 能 要 件

3.1.1 開発コストの軽減

クロスプラットフォーム開発が困難となる最も大きな原因の一つが開発コストである．ほとんどの場合，複数のアプリケーションを開発するためには，一つのアプリケーションを開発する場合よりも多くの開発コストがかかってしまう．クロスプラットフォーム開発では，更に複数のプログラミング言語の習得や各プラットフォームの特徴理解などが必要となり，その開発コストは更に増大してしまう．そのため，クロスプラットフォーム開発の低コスト化が求められる．

3.1.2 モバイル OS の SDK アップデートやサードパーティ製ライブラリへの対応

モバイル端末に搭載されているモバイル OS は，バージョンが定期的にアップデートされている．それに伴い，今までにない新たな機能が次々にサポートされて来ている．SDK アップデートもモバイル OS のバージョンアップに合わせて行われるため，このようなアップデートは今後も継続的に行われることが予想できる．クロスプラットフォーム開発環境の陳腐化を防止するためには，このようなアップデートへの対応が求められる．

また，モバイル OS の開発元以外の個人や団体が提供するサードパーティ製のライブラリは，効率的なアプリケーション開発や拡張機能の利用を可能とする．そのため，サードパーティ製ライブラリを利用できるようにすることは，本開発環境の有用性を高める上で必要であると言える．

3.1.3 各プラットフォームに特有の処理への対応

各プラットフォームには，他のプラットフォームにはない特有の処理が存在する場合がある．例えば，Android はアプリケーションにライフサイクルが存在し，アプリケーションの状態に合わせた処理の実装を容易

☆ http://www.android.com/

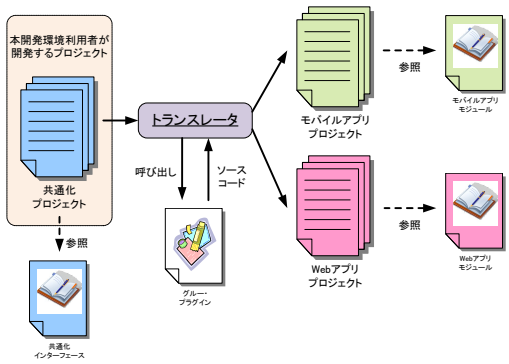

図 3 本開発環境の概念図

に行うことができる．具体的には，「3D レンダリングを行うような高負荷なアプリケーションがバックグラウンドになった際にはレンダリング処理を停止させ，アプリケーションが最前面になったときにレンダリング処理を再開させる」といった処理を実現することができる．このような特有の処理は，アプリケーションの正常な動作に影響を及ぼしたり，ユーザビリティを大きく左右する要素となり得るため対応する必要があるが，アプリケーション間の整合性が取れなくなるため，特有の処理がソースコードのどの部分に存在するかを明示的に把握できる仕組みも必要となる．

3.2 アプローチ

本開発環境の概念図を図 3 に示す．本開発環境には，各要件を解決する 3 つのアプローチが存在するため，ここではそれぞれのアプローチを図 3 に対応させながら述べる．

3.2.1 シングルソースから複数のアプリケーションを生成するトランスレータ

開発コストを削減するため，本研究ではクロスプラットフォーム開発をシングルソースのみで行えるようにすることでの解決を図る．具体的には，シングルソースを複数のアプリケーションに変換し生成するトランスレータを構築する．なお，本研究ではこのようなシングルソースを**共通化プロジェクト**と呼ぶ．ここでいうプロジェクトには，ソースコードだけでなく画面レイアウトを表す XML や画像ファイルといったアプリケーションを構成するファイルも含まれる．

トランスレータの構築を行うことで，開発者は図 3 にあるように，共通化プロジェクト開発するだけでコンパイル可能な Web アプリプロジェクトとモバイルアプリプロジェクトを生成することができるようになる．

トランスレータは，各プラットフォームのアプリケーションを生成するモジュールを持つ．本開発環境を新規プラットフォームに対応させる場合は，そのプラットフォームのアプリケーションを生成するモジュールを追加することで解決していく．

3.2.2 共通化ライブラリ

本研究では，共通化プロジェクトの開発に利用するライブラリやトランスレータが生成したアプリケーションがそれぞれ利用するライブラリとして，共通化ライブラリを用意する．共通化ライブラリとは，図 3 の共通化インターフェース，Web アプリモジュール，モバイルアプリモジュール，グルー・プラグインの 4 つの要素によって構成されるライブラリである．共通化ライブラリを用意することによって，各プラットフォームの処理をトランスレータから分離することができるため，SDK のバージョンアップに伴う影響に各アプリモジュールの更新のみで対処することが可能である．また，共通化ライブラリを利用することによりソースコードの可読性の向上が期待できる．

4 つの要素の役割はそれぞれ以下の通りである．

- **共通化インターフェース**
  クラスやメソッドなどのシグネチャの定義のみを集めた実装の存在しないモジュール．本開発環境を利用する開発者は共通化インターフェースを利用して共通化プロジェクトの開発を行う．
- **Web アプリモジュール，モバイルアプリモジュール**
  共通化インターフェースのクラスやメソッドに対してそれぞれのプラットフォームに対応する処理を実装したモジュール．アプリモジュールを複数用意することによって，複数の Web 言語やモバイル OS に対応することが可能．
- **グルー・プラグイン**
  トランスレータから呼び出されるプログラム．特定のプラットフォームのみに必要な処理に関するソースコードの追加や設定ファイルの編集を自動的に行う．これにより，各アプリモジュールの切り替えのみでは実現できない処理にも対応できるようにする．

  例として，モバイルアプリのみに位置情報を利用するための初期化処理が必要である場合を挙げる．図 4 と図 5 は，トランスレータによって生成された各アプリプロジェクトのソースコードのサンプルである．このような差異に対して，グルー・プラグインは図 5 で示しているようなモバイルアプリモジュールの初期化のコードを追加し，アプリケーションの処理の整合性を保つ．

```
package jp.ac.ritsumei.cs.ubi.yo;

import unified.location.Geolocation; ;
import unified.location.Location;
import unified.ui.Text;

public class Sample {

    public static void main(String[] args) {

        Location location = Geolocation.getlocation("gps");

        Text.show(location.getLatitude());
        Text.show(location.getLongtitude());

                    ⋮
```

図 4 Web アプリプロジェクトのソースコードのサンプル

```
package jp.ac.ritsumei.cs.ubi.yo;

import unified.location.Geolocation;
import unified.location.Location;
import unified.ui.Text;

public class Sample {

    public static void main(String[] args) {

        Geolocation.init();

        Location location = Geolocation.getlocation("gps");

        Text.show(location.getLatitude());
        Text.show(location.getLongtitude());

                    ⋮
```

図 5 モバイルアプリプロジェクトのソースコードのサンプル

あらかじめ用意された共通化ライブラリ以外にも，拡張共通化ライブラリを自由に追加することができる．拡張共通化ライブラリは共通化ライブラリと並列に存在し，共通化プロジェクト開発時に拡張共通化ライブラリの共通化インターフェースをインポートすることで，各アプリプロジェクトにも自動的にインポートされる．これによりサードパーティが構築したライブラリも，そのライブラリをラップした拡張共通化ライブラリの構築を行うことで本開発環境で利用することが可能である．

3.2.3 特有の処理の半自動生成

特有の処理の具体的な内容については，開発者が自ら実装する必要があるが，特有の処理を記述するためのテンプレートについては自動生成が可能である．そのため本開発環境では，特有の処理記述のためのテンプレートまでの自動生成をサポートする．

生成済テンプレートがソースコードのどの部分に生成されたかについては，生成済テンプレート情報をまとめたファイルを同時生成することで把握できるようにする．

4. 設　　計

本章では，前章にて提案したクロスプラットフォーム開発環境の設計を行う．

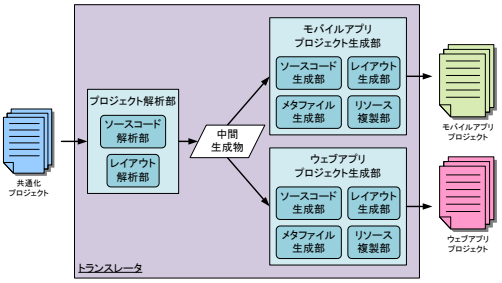

図 6 トランスレータの内部設計

まず，本開発環境は図 3 の矢印に沿って，以下の手順で利用される．

(1) 共通化プロジェクトを共通化インターフェースを利用して開発する
(2) 開発した共通化プロジェクトをトランスレータに読み込ませる
(3) トランスレータがグルー・プラグインを利用してモバイルアプリプロジェクトと Web アプリプロジェクトを生成
(4) 生成された各プロジェクトは各プラットフォーム向けのアプリモジュールを読み込んで実行

これにより，開発者は本開発環境を利用することで共通化プロジェクトの開発を行うだけで Web アプリとモバイルアプリを同時に開発することが可能となる．

次に，トランスレータの内部設計を図 6 に示し，各モジュールについての詳細を述べる．

プロジェクト解析部

共通化プロジェクトを読み込み，ソースコードとレイアウトの解析を行うモジュールである．ここで，ソースコード内で利用されているクラスメソッドやレイアウトの構成を解釈し，結果を中間生成物として生成する．

プロジェクト生成部

中間生成物から各プラットフォームに対応したプロジェクトを生成するモジュールである．グルー・プラグインも中間生成物の内容を参照してこのモジュールから実行される．

ソースコード生成部では共通化プロジェクトのソースコードを各アプリプロジェクトに再配置する．同時に，特有の処理を記述するテンプレートを生成し，生成済テンプレート情報ファイルを作成/更新する．

レイアウト生成部は共通化プロジェクトのレイアウトの解析結果を元に，各プラットフォームにあわせたレイアウトの生成を行うモジュールである．

メタファイル生成部は各アプリプロジェクトのコンパイルおよび各アプリケーションの実行に必要な設定ファイルを生成するモジュールである．

リソース複製部は各アプリケーションで利用される画像などのファイルを複製するモジュールである．

5. プロトタイプ実装

前章の設計を元にプロトタイプ実装を行った．実装環境を表 1 に示す．

表 1 プロトタイプ実装の実装環境

開発環境	Eclipse3.6 (Helios)
共通化プロジェクトの開発言語	Java
Web アプリ開発フレームワーク	Google Web Toolkit 2.0.4
モバイル端末の OS	Android 2.2

本研究では，Web アプリ開発フレームワークとして Google Web Toolkit ☆(GWT) を利用した．GWT の特徴として，Java-to-JavaScript コンパイラによる Java で開発したソースコードの純粋な JavaScript への変換がある．Android のアプリケーションも開発言語が Java であるため，共通化プロジェクトを Java で開発することでプログラミング言語の変換を考慮する必要がなくなり，共通化ライブラリを純粋にシグネチャのラップだけでできる．そのため，プロトタイピングの敷居が低く，更に Android 端末の普及状況を考慮して開発意義がある．

また，GWT はバージョン 2.0 から UiBinder が導入された．この UiBinder を用いることで，Android と同様に処理の記述とレイアウトの記述を分離することが可能となった．レイアウトの記述はどちらも XML を採用しているので，レイアウトの変換もソースコード同様容易に行うことができる．なお，レイアウトの各 XML タグが参照するウィジェットの共通化ライブラリをまだ実装していないため，現在はレイアウト生成部を暫定的にレイアウトの各 XML タグの 1 対 1 マッピングで対処している．

5.1 共通化ライブラリの仕様と実装

今回のプロトタイプ実装では，共通化インターフェースを GWT の API のシグネチャを用いて実現した．これにより，Web アプリモジュールは共通化インターフェースと同じシグネチャになるため，ラップは行わずに GWT の API をそのまま流用した．モバイルアプリモジュールは Android の API を GWT の API のシグネチャでラップして用意する．この手法を採用した理由として実装の簡素化が挙げられるが，他にも「Web アプリと同一ソースからモバイルアプリを生成」という本研究の目的が達成できるという意味合いもある．

また，GWT の API の内容によっては，モジュールあるいはクラスの追加生成を行う必要があり，それらの追加生成はグルー・プラグインによって行われる．このような例としては RPC 共通化ライブラリがある．これは，GWT の RPC インタフェースを Android 上でも利用できるようにした共通化ライブラリである．RPC インタフェースのみを共通化し，通信プロトコルは GWT と Android で別のものを使用している．GWT ではプロジェクトのビルド時に RPC のスタブクラスを生成してスタブ経由で通信を行うが，このスタブ生成処理はプラットフォームごとのアプリモジュールの切り替えでは対応できない処理である．そのためグルー・プラグインでスタブクラスを生成することで，処理の共通化を実現している．

他にも，現在実装できている共通化ライブラリとしては，アラートダイアログやイベントハンドラがある．

5.2 未実装の共通化ライブラリへの対処

本研究では，共通化プロジェクトを共通化インターフェースを用いて開発する．しかし，共通化インターフェースのシグネチャとして利用した GWT の API は，提供する API が多岐に渡っている．そのため，API のラップ作業を全て行うには膨大な時間が必要となり，現状では API の一部しかラップができていない．ここで，ラップができていない共通化ライブラリを **Null 共通化ライブラリ**と定義する．Null 共通化ライブラリは，シグネチャは存在するが中身の実装が存在しないライブラリである．つまり，共通化インターフェースとアプリモジュールの一部のクラスやメソッドが実装されているライブラリであり，片方のアプリモジュールのみに実装が存在するメソッドなども存在する．プロトタイプ実装では，GWT の API のシグネチャを用いて共通化インターフェースを実現しているため，実装できていない共通化ライブラリは，全て Null 共通化ライブラリとして存在することになる．これにより，共通化インターフェース，Web アプリモジュールのシグネチャ，モバイルアプリモジュールのシグネチャが全て存在することになるため，共通化プロジェクトから生成した各アプリプロジェクトをコンパイルする際のビルドエラーを回避することができる．

また，Null 共通化ライブラリを利用した各アプリプロジェクトには，Null 共通化ライブラリの未実装クラスや未実装メソッドを実装する必要がある．そのよう

☆ http://code.google.com/intl/ja/webtoolkit/

な課題に対して，現在は Null 共通化ライブラリを各アプリプロジェクトにコピーし，参照を切り替える手法を検討している．

5.3 拡張共通化ライブラリ

今回はベースとなる共通化ライブラリのシグネチャに GWT の API を用いた．これにより，Web アプリとしての機能を数多く実現することができるようになるが，全ての機能を網羅することはできない．そのため，GWT に網羅されていない API を利用したいという要求も，サードパーティが構築したライブラリと同様に，拡張共通化ライブラリによって対処する．

本研究では拡張共通化ライブラリの例として，3D レンダラーライブラリの実装を行う．3D グラフィックスは最近の Web アプリで見られるようになってきたが，GWT には対応する API が存在しておらず，かつ GWT の API のシグネチャを利用した共通化ライブラリに存在しないことにもなるため，拡張共通化ライブラリの例として適していると言える．

共通化インターフェースのシグネチャとして，今回は Android の 3D グラフィックスに採用されている OpenGL ES [☆]のシグネチャを利用した．Web アプリモジュールの実装には O3D[☆☆]を用いた．O3D とは Google 社が提供する 3D グラフィックスを描画するためのオープンソース WebAPI である．O3D を利用することで，高度な 3D モデルを描画可能な Web アプリの開発が可能となる．O3D の API は JavaScript で書かれており，そのままでは開発言語の違いにより拡張共通化ライブラリに組み込むことができないが，GWT には JavaScript Native Interface [☆☆☆](JSNI) と呼ばれる Java のソースコード内に JavaScript 文を記述する手法が存在しており，O3D の JavaScript の処理を Android の OpenGL ES のシグネチャでラップすることによって Web アプリモジュールの構築が可能である．

他にも，我々は Geolocation API [☆4]や Web Storage [☆5]といった GWT の拡張クラスについても拡張共通化ライブラリの実装を行っており，今後は GWT が対応していない HTML5 の各機能を中心に順次追加していく予定である．

☆ http://www.khronos.org/opengles/
☆☆ http://code.google.com/intl/ja/apis/o3d/
☆☆☆ http://code.google.com/intl/ja/webtoolkit/doc/latest/DevGuideCodingBasicsJSNI.html
☆4 http://dev.w3.org/geo/api/spec-source.html
☆5 http://dev.w3.org/html5/webstorage/

6. 関 連 研 究

本研究では一つの共通化プロジェクトから複数のアプリケーションを生成する手法を扱う．同様に，シングルソースから複数の生成物を得る研究やサービスは多く存在する．ここではその一部を紹介する．

Literate Programming[3] はプログラムとドキュメントを同時に記述する手法であり，Literate Programming を実装したシステムとしては WEB システムがある．WEB システムは一つのソース (WEB 文書) から，TANGLE プログラムを利用して Pascal のソースコードを，WEAVE プログラムを利用して Tex のドキュメントをそれぞれ生成する．一つの WEB 文書にソースコードとドキュメントを併記する点，生成したいものによって別のプログラムを利用する点が本研究と異なっている．

SWIG[4](Simplified Wrapper and Interface Generator) は C/C++で書かれたプログラムやライブラリを，Tcl/Tk, Perl, Python, Ruby, PhP などのスクリプト言語に接続するためのオープンソースのソフトウェア開発ツールである．SWIG を利用して C/C++ のプログラムやライブラリにアクセスするためのグルーコードを生成しておくことで，開発者は複数のスクリプト言語から同一のプログラムやライブラリを利用することができるようになる．本研究では，元となる一つの共通化プロジェクトから複数のプロジェクトを生成するため，各プラットフォームの実装に対応した各ライブラリ (アプリモジュール) を用意する必要があり，複数のスクリプト言語から一つのライブラリを参照する SWIG とは正反対のアプローチであるといえる．

Web 技術を用いてモバイルアプリを開発するフレームワークとして，PhoneGap[5] や Titanium[6] などがある．HTML(+CSS) と JavaScript などのスクリプト言語を用いてモバイルアプリを開発することができ，対応するモバイルプラットフォームも SymbianOS, iOS, AndroidOS と様々である．Webkit ベースで動作するため，モバイルプラットフォーム間でのクロスプラットフォーム開発は，プラットフォームごとに共通化インターフェースの開発言語でラップしたアプリモジュールを用意する必要がある本研究よりも容易に行える．しかし，逆に Webkit がボトルネックとなってアプリケーションの性能が左右される恐れがある．また，モバイルアプリの開発を目的としているので，そのままでは Web アプリとして動作させることできない．

Puder は Android アプリを iPhone アプリに変換する手法について研究している[7)8)]．Android と iPhone のプログラミングモデルの違いを解決するために，Puder は XMLVM というバイトコードレベルのクロスコンパイラを用意している．XMLVM は，まず Java のソースコードからコンパイルされた Java クラスファイルを，XMLVM ツールによって XML ファイルとして変換する．この XML ファイルは，Java クラスファイルの構成を表現しており，XML タグを用いて Java コンパイラが生成したバイトコード命令単位で記述されている．この XML ファイルをバイトコード命令ごとに Objective-C に再変換することで，アプリケーション間の変換を行うことができる．この手法を利用すれば，既存の Android アプリも変換できるようになることが期待できるが，この手法では各プラットフォームの特有の処理までは考慮されておらず，更に，サードパーティ製のライブラリへの対応についても考慮されていないため，本研究の手法と比較して拡張性に乏しい．

7. まとめと今後

本研究では，Web アプリと同一ソースでモバイルアプリを生成するために，Web アプリとモバイルアプリの同時生成を可能とするクロスプラットフォーム開発環境の構築を提案し，プロトタイプ実装を行った．

プロトタイプ実装では，Web アプリ開発フレームワークとして GWT を，モバイルアプリのプラットフォームとして Android をそれぞれ対象とすることで，共通化プロジェクトの開発言語と生成する各アプリプロジェクトの開発言語を Java で統一することができたため，プログラミング言語間の変換を考慮することなく開発環境を構築することができた．また，実装した共通化ライブラリや拡張共通化ライブラリについても紹介し，未実装の共通化ライブラリへの対処も行った．

今後は，Null 共通化ライブラリに順次実装を追加していく予定である．

また，本研究で提案した共通化ライブラリがアプリケーションのパフォーマンスに及ぼす影響についても評価していく必要がある．

更に，本開発環境の有用性を証明するために拡張共通化ライブラリを利用したサンプルアプリケーションの開発を予定している．特に，現在実装中の 3D レンダラーライブラリを用いることで，Google Street View や Pano UMECHIKA[9)] のような 3D レンダリングによって構成される Web アプリをクロスプラットフォームで提供できるようになる．

参 考 文 献

1) 荒瀬由紀, 前川卓也, 原隆浩, 上向俊晃, 西尾章治郎: 携帯電話を用いた Web 閲覧のためのコンテンツ適応的提示システム, 情報処理学会論文誌, 2006.
2) 中村正人, 辻野友孝, 大囿忠親, 新谷虎松: マルチブラウザのための Web コンテンツ管理支援システムとその応用, 第 23 回人工知能学会全国大会, 2009.
3) Donald E. Knuth: Literate Programming, The Computer Journal, Vol.27, No.2, pp.97-111,1984.
4) David M. Beazle: "SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++", 4th Annual Tcl/Tk Workshop, Monterey, 1996.
5) PhoneGap, `http://www.phonegap.com/`
6) Titanium, `http://www.appcelerator.com/`
7) Arno Puder: Cross-compiling Android applications to the iPhone, Principles and Practice of Programming in Java, pp.69-77, 2010. ACM.
8) A. Puder and I. Yoon: Smartphone Cross-Compilation Framework for Multiplayer Online Games, "Mobile, Hybrid, and On-Line Learning, 2010. ELML '10.", pp.87-92, 2010. IEEE.
9) Ismail Arai, et al.: "Pano UMECHIKA: A crowded underground city panoramic view system," The proceedings of the International Symposium on Distributed Computing and Artificial Intelligence 2010 (DCAI '10), pp.174–181, Valencia, Spain, 8th September, 2010.