生産性と機能性を向上させるスマートフォン向け ネイティブアプリとウェブアプリのハイブリッド構造

Hybrid Structure of Native Application and Web Application for the Smartphones which improves Productivity and Functionality

渡　邉　充　隆

要　約　スマートフォン向けのアプリケーションを構築する場合に，各種あるプラットフォームのネイティブアプリケーションとして開発するには，学習のコストやスキルをもつ要員の確保の面で多くの課題がある．

本稿では，スマートフォン向けのアプリケーションの構造としてネイティブアプリケーション，伝統的なウェブアプリケーション，ブラウザ内で動作するウェブアプリケーションを解説する．それらを比較，評価した上で，ブラウザ内で動作するウェブアプリケーションをベースにして，ネイティブとウェブアプリケーションのハイブリッド構造を検討した．この構造により，端末に搭載された GPS やカメラなどを制御可能な JavaScript アプリケーションを記述できる．実際にプロトタイプを開発し，屋外の実証フィールドにて検証した．このハイブリッド構造はプラットフォーム固有の部分を局所化して分離している．これにより，サーバサイドのウェブアプリケーション技術者が保有している技術を使ってモバイル向けのアプリケーションを構築できることを検証した．

Abstract　When building the smartphone-oriented application as a native application on a wide variety of platforms, there are numerous problems over studying expenses and securing of skilled development personnel.

In this paper, we discuss the structure of the smartphone-oriented application referring to the native application, the traditional web application and the web application running in the browser. After comparing and evaluating them, we examine the hybrid structure of the native and web applications in terms of the web application running in the browser. Use of this structure allows us to write the JavaScript application program able to control GPS, cameras and such loaded onto the terminal. Actually we developed the prototype, verified one with outdoor demonstration field. This hybrid structure separates the pure application code from the platform-specific part. In this way, we verified that server-side web application developers could build the mobile application, using the technology which they already had.

1. は　じ　め　に

iPhone をはじめとするスマートフォンが注目を浴びている．iPhone の全世界での出荷台数は 2009 年 9 月時点で 3000 万台を突破しており，スマートフォン向けのアプリケーションの市場規模も 2009 年 8 月時点で，App Store[*1] だけで月間約 2 億ドルと推計されている[1]．このようにスマートフォンは端末台数でもアプリケーションの市場規模でも無視できない存在となっている．

市場に出回っているスマートフォンに共通しているのは，オペレーティングシステム（OS）

とメールやウェブブラウザなどの基本的なアプリケーション群がセットになった“プラットフォーム”を搭載していることである．プラットフォームには Windows Mobile，iPhone OS，Symbian OS，Android などといった種類が存在するが，これらのプラットフォームの最大の特長は，利用者がアプリケーションを追加導入できること，これまでの携帯電話のような高い参入障壁がなく，比較的容易にアプリケーションを開発，販売できるサービスが提供されている（または提供が計画されている）ことである．

このような大きな市場になりつつあるモバイル向けアプリケーションは，より私たちに身近になり，携帯端末からのクラウド利用を前提としたアプリケーションが今後一層増えることが予想される．これまでサーバ側のシステム構築を主なビジネスとしてきた SIer においても，将来的にモバイル向けアプリケーションへの対応を無視できなくなるだろう．

しかしながら，各種ある携帯端末プラットフォームごとに開発者をそろえるには，ハードウェアやプラットフォームの癖も含めた深い知識が要求されること，発展途上であるために技術進展のスピードが速くスキルや知識の有効期間が短いことから，短中期的には育成・維持のためのコストが多くかかると考えられる．

この課題に対し，サーバサイドの開発を行ってきた技術者が保有する技術を使ってモバイル向けのアプリケーションを開発できるようにし，技術的に着手しやすくすることと，プラットフォームごとの違いをできるだけ吸収し，統一した方法で開発できるようにすることがアプローチとして考えられる．

本稿では，今後ますます増加が予想されるモバイル向けアプリケーションに関して，構造のパターンを解説し，上記のアプローチで考案したネイティブアプリケーションとウェブアプリケーションの長所を兼ね備えたハイブリッド構造のアプリケーションを紹介する．また，この構造に基づいて実際にプロトタイプを開発し，屋外での実証実験に参加して期待した動作が実現できたかを検証した．

2.　モバイル向けアプリケーションの構造と開発の特徴

アプリケーション開発のプラットフォームとしても充実しているスマートフォン用のプラットフォームだが，一口にアプリケーションといっても様々な構造のものが存在している．この章ではそれらの特徴を整理する．

2.1　ネイティブアプリケーション

ネイティブアプリケーションは，各プラットフォーム固有の開発環境や開発言語を用いて作成され，ソースコードをコンパイルし，プラットフォーム上で直接実行できる形式に変換したものである．多くの場合，インターネット上のマーケットサイトから導入したり，PC と接続して導入のための支援ソフトウェアを経由して導入することとなる．アプリケーションもデータも端末に一緒に導入するのが基本だが，近年のスマートフォンのアプリケーションの流れとしてネイティブアプリケーションもサーバと連携して動作するものが増えている．

ネイティブアプリケーションの特徴は，カメラ，GPS，加速度計など端末固有のデバイスを利用できるため，位置情報や利用者の動作状況など，モバイル端末ならではのデータを利用する機能が実現できたり，音声や動画などをふんだんに扱うものなど，プラットフォームの持つ機能を活用できることである．動作速度も高速である．また，プラットフォームの操作性に則

れば，利用者は初期学習コストを低減でき，快適な操作にもつながる．

2.2　伝統的なウェブアプリケーション

クライアントサイドでのロジックは排除し，HTML のフォームを使って，利用者の操作ごとにサーバからページ全体を取得するアプリケーションを本稿では伝統的なウェブアプリケーションと呼ぶ．各スマートフォンは PC と比較してもそん色ないブラウザを搭載しており，これまでの PC 向けのウェブアプリケーションであっても一応の動作はするが，画面サイズや，指での操作を考慮したレイアウト変更，画像サイズの調整などの対応をすることで使い勝手が向上する．モバイル対応に力を入れているウェブサイトのアプリケーションは，後述するようなブラウザ内でも動作するウェブアプリケーションとして実装されている．

伝統的なウェブアプリケーションはページ全体をやり取りするという性質上，モバイルからの利用では回線の不安定さや帯域の狭さなどの影響を受けてしまい，快適な操作を実現することは容易ではない．また，端末固有のデバイスなどをアプリケーションから利用できない．しかしながらネットワーク接続が確立していればどこでも利用できる利便性があるのに加え，データがすべてサーバ内に保管されているため，端末を紛失してもデータを失ったり漏えいしたりする危険性は比較的小さい．

2.3　ブラウザ内で動作するウェブアプリケーション

本稿でいうブラウザ内で動作するウェブアプリケーションとは，ウェブブラウザ内で JavaScript によって記述されたプログラムを実行し，クライアントサイドでもある程度の処理を実行しながら，一つまたは複数のサーバから必要なデータだけを都度取得し，組み合わせるアプリケーションを指す．開発作業の多くは JavaScript でのプログラミングである．

この形態の特徴としては，伝統的なウェブアプリケーションの良さを継承しつつ，利用者との対話的な操作を実現している点があげられる．また，iPhone などに搭載されているブラウザは，JavaScript の実行が高速であると定評のある WebKit というブラウザのエンジンを搭載しているため，スクリプト言語である JavaScript で一般的に懸念される動作の遅さについては，実用上問題ないレベルである．なお，必要なデータだけを取得することで素早いレスポンスを可能としているが，複数のサーバから毎回データを取得すると，全体的な軽快性は薄れてしまう危険性がある．また，伝統的なウェブアプリケーションと同様に端末に搭載されたデバイスは利用できない．

3.　アプリケーション開発にあたっての構造の選択

前章で述べた通り，アプリケーションの構造はそれぞれに特色がある．システム全体の一部という位置づけでモバイル向けアプリケーションをとらえたとき，システム構築ベンダがどのような視点で評価すればよいかを本章にまとめた．なお，ネイティブアプリケーションに関しては 2010 年以降のプラットフォームの潮流を作っていくと考えられる iPhone と Android を具体的な事例として取り上げている．

3.1　機能実現の視点

機能実現の視点とは，求められている機能が実現できるかどうかという視点である．当然

個々の適用先のシステムごとに求められる機能要件は変わるが，モバイルという点で特徴的な機能要件は，端末を所持している利用者の現在位置を把握する機能，バーコードや RFID など物品に貼付されたデータを読み取る機能，また，通信網のない場所でも利用が可能な機能などがあげられる．これらの機能は，業務利用では倉庫や店舗での棚卸やピッキングの作業などが具体的な利用先として考えられる．

ネイティブアプリケーションの構造の場合は，各端末に搭載されたカメラや GPS などのデバイスをフル活用することができるが，ウェブアプリケーションの構造の場合はこれらのデバイスは利用できず，通信網のないオフラインの場所ではアプリケーションそのものの利用ができなくなる．

3.2　操作性の視点

操作性の視点は主にユーザインタフェースに関することである．モバイル端末では指やスタイラス（ペン状の操作道具）でのタッチパネル操作のほか，キーボードを含むハードウェアボタンによる操作もある．また，屋外や暗い場所で操作する状況も考慮しなければならない．

一般的には，画面の要素の視認性，アイコンや文字の表現，大きさなどのビジュアル面のほか，操作を受け付けたことのフィードバックなど，操作に直接影響する要素の精緻な考慮が求められる．この点では，プラットフォームが提供する開発環境に再利用可能な部品がそろっていたり，設計や実装のガイドラインが存在するため，ネイティブアプリケーションでもウェブアプリケーションでも大きな違いはないが，ネイティブアプリケーションの方が操作に対するフィードバックが軽快で感触が良い．

システム構築という観点では，反応速度も大きな要因となる．反応速度についてはアプリケーションだけではなくシステム全体として見たときの他の要素との連携と親和性が重要である．これに関してはウェブアプリケーションでもネイティブアプリケーションでも基本的な部品はそろっており，実現できることに大きな違いはない．

3.3　学習・生産性の視点

各プラットフォームはネイティブアプリケーションを開発するためのフレームワークを提供している．ほとんどは，ユーザインタフェースの部品を配置してそれらが操作された際の処理を記述していくスタイルである．さらに統合開発環境による支援が利用できる．そのため，特別に凝った操作を追求しない限り，ユーザインタフェースを構築する際の生産性は高いと考えるが，アプリケーション・ロジックの開発となると，開発言語への慣れが大きく生産性に影響を与える．iPhone では Objective-C というエンタープライズ系の開発ではほとんど見られない言語を利用している．また，Android は Java で開発を行い，Java 標準ライブラリをできるだけ活用したフレームワークを用いるため，Java 技術者であれば初期の学習コストは比較的低い．

ウェブアプリケーションではアプリケーション・ロジックは従来通りのウェブアプリケーションの開発方法が踏襲できる．ブラウザ内のプログラムには JavaScript を用いるのが事実上の標準であり，多くの解説記事が参照できる．また，過去には JavaScript でのアプリケーション開発は，さまざまな種類のブラウザに対応することへの負荷が高いという問題があったが，開発を支援する各種のライブラリがブラウザの違いを吸収しつつ，短いプログラムで多く

の処理を行えるようになっており，生産性を高めつつある．

3.4　導入・保守性の視点

システムのライフサイクルを考慮すると，アプリケーションの導入や更新を速やかに行えるか否かは，システム全体の保守や機能拡張という点で大きなポイントとなる．

iPhone の場合，実機へのアプリケーションの導入は大きく二つの方法がある．一つ目は，iPhone Developer Program への有料登録をした開発者が，導入先の個々の端末用にアプリケーションパッケージをビルドする方法である．実機の利用者は配布されたアプリケーションパッケージを iTunes などを用いて自ら導入しなければならない．この方法では，導入先の端末のすべてを開発用として登録しなければならず，そのために，端末それぞれの固有識別子を集めて開発者用のウェブサイトに登録するという手順が必要となり，非常に煩雑である．さらに，開発用として登録できる端末数には上限がある．二つ目は，アプリケーションのマーケットプレースである App Store を通じて公開する方法である．これは，企業内で使用するアプリケーションには向かない．また，公開の申請をすると，Apple 社による審査があり，数週間から数カ月待たされることもある．このため，いずれの方法を用いても速やかなアプリケーションの導入や更新は難しいと考える．Android でも基本的には同じようなパターンであるが，オープンな場で開発が進んでいることもあってマーケットプレースで公開する場合に審査は不要である．また開発端末を登録する必要もない．

一方，ウェブアプリケーションの場合はブラウザで再読み込みするだけでよく，大きな優位点となる．

3.5　移植性の視点

実際に異なるプラットフォーム向けに移植する機会がどれほどあるかという点が焦点になるが，より多くの利用者を確保するには可能な限り多くのプラットフォーム向けにリリースできるのが望ましい．この点では，ウェブアプリケーション構造に優位性がある．

複数のプラットフォームに向けて統一的にネイティブアプリケーションを開発できるフレームワークはいくつか存在しており，PC 向けアプリケーションと共通化できる可能性のある Flex（Adobe Systems 社製の Flash Player や AIR で実行可能な開発フレームワーク）のスマートフォン向けの実行環境も Adobe Systems 社が開発中だが，これらのフレームワークはプラットフォームベンダが出している開発者規約の制限を受ける場合があるため，採用の際には注意が必要である．

3.6　考察

3.1 節から 3.5 節までの各視点の評価を表 1 にアプリケーション構造ごとにまとめた．これらをみると，各構造ともメリット・デメリットがあることが分かるが，エンタープライズ・システムの構築には，学習が容易で生産性が高く，導入・保守が速やかに行えることが重要であると考える．この点ではネイティブアプリケーションは適さない．また伝統的なウェブアプリケーションはブラウザ内動作ウェブアプリケーションよりも生産性は高いが，操作性に限界がある．総合的にはブラウザ内動作ウェブアプリケーションが最も望ましい．

表 1　アプリケーション構造ごとの各視点の評価

構造 / 視点	ネイティブアプリケーション	伝統的なウェブアプリケーション	ブラウザ内動作のウェブアプリケーション
機能実現	◎	△	△
操作性	◎	△	○
学習・生産性	△	◎	○
導入・保守性	×	◎	◎
移植性	×	○	○

4.　ネイティブとウェブアプリケーションを融合するハイブリッドアプリケーション

前章の分析からはブラウザ内動作のウェブアプリケーションが望ましいのだが，さらに端末に搭載されたデバイスの機能を使えるようにするために，JavaScript から端末固有の機能を利用するアプリケーションの構造を，iPhone を題材に検討した．その結果，ネイティブ・プログラムに実装されたデバイスの制御などの機能を JavaScript から呼び出すという構造を持つアプリケーションの構成（ハイブリッドアプリケーション）を導出した（図 1 右端）．この構造によってプラットフォーム固有の部分をネイティブ・プログラム部分に押し込め，デバイス制御などのプログラムを記述し，アプリケーション・ロジック自体はウェブアプリケーションや JavaScript の範囲内で記述できるように分離する．この構造は前章のアプリケーションの配布・導入というネイティブアプリケーションの不得意な点を受け継いでしまうが，ネイティブ・プログラム部分はアプリケーション・ロジックに左右されずに比較的安定しているため，頻繁に更新する必要がなく，インパクトはそれほど大きくない．

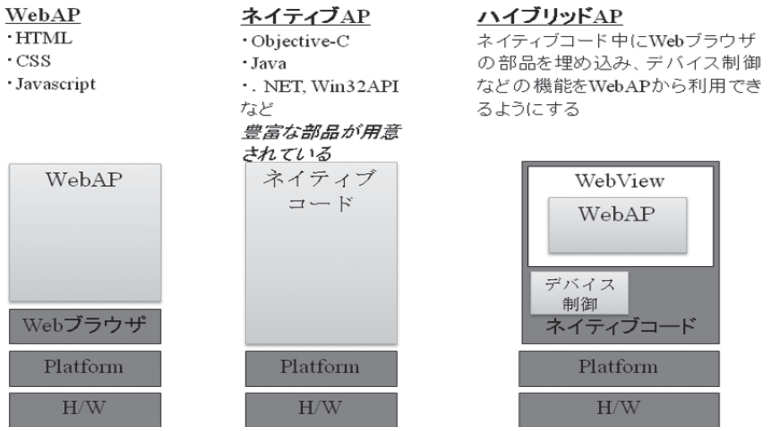

図 1　ハイブリッドアプリケーションの構造

4. 1　ハイブリッド構造のアプリケーションの実現

ネイティブと JavaScript とのハイブリッドアプリケーションは，iPhone の開発環境に用意されている UIWebView というブラウジング機能を持った部品の View 機能によって実現した．JavaScript が記述された HTML（インターネット上，ローカルファイル共に使用可能）を UIWebView にて読み込み，画面表示は HTML に依存した形で行っている．レンダリングエンジンには WebKit を使用しており，ブラウザは基本的に MobileSafari と同等なものとして扱える．

JavaScript からネイティブコードの機能を使用するために UIWebView のリクエストフック機能を用いる．ネイティブコードを呼び出すためのリクエスト URL のパターンを決めておき，UIWebView 内でフックしたリクエストの URL を解析し，パターンに合致した場合はネイティブ側の処理記述を呼び出すという形で処理する．

また，ネイティブ側から戻り値を返したい場合は，ネイティブ側の処理終了時に呼び出されるコールバック関数を JavaScript 側で定義し，ネイティブ側からそのコールバック関数を実行する．以下に例を挙げる．

■ JavaScript 側の記述例

```
// ネイティブの関数(getByNumber)をコール
function callNativeMetohd(param) {
     // キーワードとして「Keyword」を先頭に付与した処理命令をリクエストとして発行する
     document.location = 'Keyword:getByNumber:callbackFunc:' + param;
}
// コールバック関数にて戻り値を受け取る
function callbackFunc(returnValue) {
     // 戻り値を解析し処理を行う
     var result = returnValue;
...
}
```

ネイティブ側の動きとしては，UIWebView に定義されているリクエストのフック処理に，ルールに従った処理分岐を記述する．以下に例を挙げる．

■ネイティブ側の記述例（Objective-C）

```
// リクエストフックメソッド（エラー処理は割愛）
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:
         (NSURLRequest *)request
         navigationType:(UIWebViewNavigationType)navigationType {
   NSURL *url = [request URL];
   NSString *urlString = [url absoluteString];
   // リクエストがリンク押下の場合，リクエストの文字列を解析
   if(navigationType == UIWebViewNavigationTypeLinkClicked) {
     // リクエスト URL があらかじめ定めたパターンに合致する場合，ネイティブへの処理命令として解釈
     NSArray *parts = [urlString componentsSeparatedByString:@":"]; // コロンで分解
     if([(NSString *)[parts objectByIndex:0] isEqualToString:@"Keyword"]) {
       // キーワードで始まる文字列の場合
       NSString *funcName = (NSString *)[parts objectAtIndex:1];
       NSString *callbackFunc = (NSString *)[parts objectAtIndex:2];
       NSString *param = (NSString *)[parts objectAtIndex:3];
       if([[funcName isEqualToString:@"getByNumber"]) {
         NSString *result = [self getByNumber:param];
         // コールバックとして呼び出す JavaScript の文を文字列で作成
         NSString * jsCallback =
           [NSString stringWithFormat:@"%@('%@')", callbackFunc,result];
         // UIWebView に処理命令を発行
         [webView stringByEvaluatingJavaScriptFromString:jsCallback];
       }
       return NO;    // リクエスト処理をブラウザに戻さない場合，NO を返却
     }
   }
   return YES;       // 通常のリクエストとして処理する場合，YES を返却
}
```

4. 2　生産性を高める JavaScript ライブラリの活用

JavaScript でのアプリケーション開発は，ユーザインタフェースの制御や動的なデータのローディングがおもな処理となる．Ajax（Asynchronous JavaScript and XML）が技術として浸透してきたのに伴い，Ajax を使って動的に HTML を書き換えるためのフレームワークとして，様々な JavaScript ライブラリがオープンソースのものを中心に出回っている．これらを使うことで，例えばリンクタグをクリックしたらその直後の DIV 要素に指定の URL をアクセスした結果を埋め込むといった処理が，少ない記述で実現できる．また画面要素をアニメーションさせながら移動，変形，表示させるような視覚効果を簡単に出す部品や，カレンダーなどのユーザインタフェース部品も揃っている．これらのライブラリを十分に活用することが，アプリケーションをハイブリッド構造にする目的の一つである．iPhone の UIWebView は JavaScript の実行に制限がないので，目的を十分満たしている．

代表的な JavaScript ライブラリを以下に列挙した．いくつかは企業の支持を受けている．

・jQuery，jQuery UI（Microsoft，Nokia などが支持）
・DoJo（IBM などが支持）
・Mootools
・Yahoo! UI Library
・prototype.js

4. 3　JavaScript から使える標準的な API としての実装

JavaScript からネイティブプログラムの機能を呼び出す API を定義する際に，世界的な標準 API を参考にすることで，開発者の負担を減らし，将来的にその API がどのブラウザでも利用可能となった時に円滑に移行できるのが望ましい．

たとえば Geolocation API[2] は HTML から地理的な位置情報を取得するための統一的なインタフェースで，W3C が仕様を策定中である．次章で述べる例ではハイブリッドモジュールを使い，GPS を Geolocation API に則った API として制御できるように実装した．

5.　検証

今回のハイブリッド構造の実現性を検証するために，東京ユビキタス計画[3]実証実験向けのアプリケーションを検討し実装した．

5. 1　アプリケーションの概要

このアプリケーションは街中で RFID や QRcode を読み取り，埋め込まれた ucode（ユビキタス ID センターが推進しているユニークな ID を割り当てるためのコード体系）[4] に関連づけられた位置情報を取得することで現在地を把握し，適切なコンテンツを利用者に提示するというものである．以下のような点を技術的なポイントとしてアプリケーションの機能を策定し実装した．

・カメラを制御し QRcode を読み取る．
・GPS から値を取得できるように実装した Geolocation API を用いて，Google Maps API と連携させ，現在位置周辺の地図を常に表示する．
・「actlogPOM(P)」（アクトログポンプ）[*2] [5]や公開されているウェブ API からデータを取

得し表示する．

作成したコンセプトアプリケーションの全体像と iPhone 版のスクリーンショットを図 2 および図 3 に挙げる．

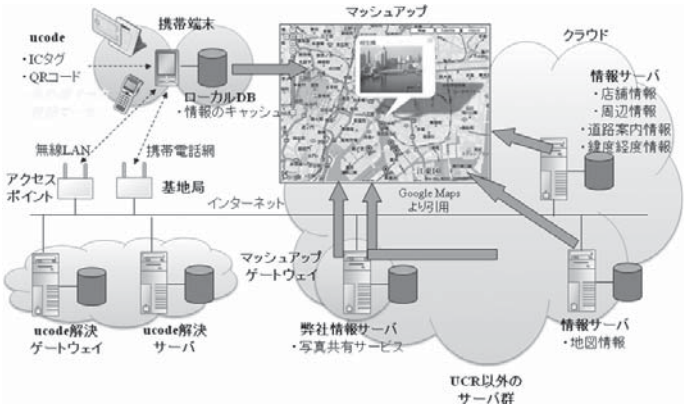

図 2　東京ユビキタス計画 2009 参加のためのコンセプトアプリケーションの全体像

図 3　作成した iPhone 版アプリケーションのスクリーンショット

5. 2　結果

ハイブリッド構造により，携帯端末に搭載されているカメラや GPS などのデバイス制御を JavaScript の API として整備し，情報を取得してマッシュアップを行う JavaScript アプリケーションを記述することができた．構築したアプリケーションの規模は，HTML が約 80 行，JavaScript が約 1000 行（うち Geolocation API の実装は約 80 行），ネイティブ・プログラムが約 1300 行（うち主要部分は約 300 行．自動生成コードは除く）であり，iPhone アプリケーション開発未経験の技術者 2 名が約 2 ヶ月で開発した．

6. お わ り に

本稿は2008年度の研究プロジェクトの成果を報告したものであり，2010年5月現在は状況が異なっている部分もある．たとえば，2009年6月にリリースされたiPhone OS 3.0ではMobileSafariにGeolocation APIが実装され，今回構築したアプリケーションの機能の一部はブラウザのみで実装できるようになっている．また，2009年9月にはNovell社により，iPhone上でC#を使った開発ができる「MONO Touch」フレームワークがリリースされた．一方Apple社は2010年4月に開発者規約を改訂し，クロスコンパイラを禁止したため，Adobe Systems社が用意していたFlashからiPhoneネイティブアプリケーションへのバイナリ変換ソリューションは，日の目を見ることはなくなった．このように，技術の進展とそれを取り巻く状況変化のスピードは速く，スキルや知識の有効期間は短い．

最後に，研究プロジェクト関係者各位と，本稿の執筆に関して多大なるご指導ならびにご協力をいただいた皆様に深く感謝申し上げる．

＊1 米国Apple社のアプリケーション販売サービス．
＊2 携帯電話のGPS機能，カメラ機能，メール機能と，PCのリッチクライアント技術を組み合わせ，利用者がGPS携帯電話で撮影した画像/位置情報をWeb地図上に自動的に配置・表示・管理するソフトウェア．日本ユニシス・ソリューション(株)が開発した．

参考文献

［1］ “iPhone and Android app discovery and usage August 2009”,AdMob, 2009
http://metrics.admob.com/wp-content/uploads/2009/08/AdMob-Mobile-Metrics-July-09.pdf
［2］ Andrei Popescu, “Geolocation API Specification W3C Working Draft 07 July 2009”, W3C, 2009
http://www.w3.org/TR/2009/WD-geolocation-API-20090707
［3］ 東京ユビキタス計画，「東京ユビキタス計画」実験事務局，2009年
http://www.tokyo-ubinavi.jp/
［4］ ユビキタスIDセンター，2009年 http://www.uidcenter.org/japanese/uid.html
［5］「GPS携帯電話で撮影した画像を地図上に自動マッピングできる『actlogPOM(P)™（アクトログポンプ)』」，Club Unisys +PLUS，日本ユニシス，Vol.04，2006年5月
上記参考文献に記載のURLは，2010年6月10日時点での存在を確認．

執筆者紹介 渡 邉 充 隆（Mitsutaka Watanabe）
2000年日本ユニシス(株)入社．モバイル，ユビキタス・コンピューティング，Web2.0関連のミドルウェアの研究開発に従事．現在，共通利用技術部Webビジネス技術室に所属．