自動販売機システムのクラウド化に向けたアプリケーションアーキテクチャの設計

M2011MM040 小池由和
指導教員：野呂昌満

1 はじめに

クラウドコンピュティーングの浸透によって，ネットワークを通じた組込みシステムと外部サービスの連携が期待されている．スマートフォン等の携帯端末に配置したアプリケーションからクラウドにアクセスすることで，組込みシステムの位置透過な制御が実現可能となる．われわれは，自動販売機システムを事例として組込みシステムのクラウド化を目指している．本研究では，自動販売機システムのクラウド化における携帯端末アプリケーションのアーキテクチャ設計を扱う．

携帯端末アプリケーション開発に用いられるプラットフォームは携帯端末毎に提供されており，各プラットフォームのアーキテクチャは異なる．アプリケーション開発者は，選択した携帯端末に対応するプラットフォームで提供されているコンポーネントを用いてアプリケーション開発をおこなう．

携帯端末プラットフォームはアーキテクチャが異なることから，プラットフォーム毎にアプリケーションアーキテクチャを設計する必要がある．各携帯端末プラットフォームを用いて設計したアプリケーションアーキテクチャは，各プラットフォームに依存する．このことから，他の携帯端末プラットフォームへの移植が困難であり，アプリケーションアーキテクチャの再利用性が低下する．

本研究の目的は，携帯端末のアプリケーションアーキテクチャの標準化である．一般的に携帯端末のアプリケーションアーキテクチャ設計に用いられる Model-View-Controller[1](以下，MVC) を用いてアーキテクチャ設計をおこなう．MVC に基づき携帯端末のプラットフォームを用いて設計したアプリケーションアーキテクチャを比較することで，共通構造を定義することで，携帯端末のアプリケーションアーキテクチャの標準化をおこなう．本研究では携帯端末のプラットフォームとして，Android[2] と iOS[3] を扱う．

同様の機能を持つアプリケーションを Android，iOS 各プラットフォームで設計する．事例として，TwitterAPI・FacebookAPI・MIxiAPI を用いたアプリケーションを扱う．設計したアプリケーションアーキテクチャを MVC に基づいて比較し，携帯端末のアプリケーションアーキテクチャの共通構造を定義した．

2 背景技術

2.1 Android

Android は，スマートフォンやタブレット端末などの携帯端末を対象としたプラットフォームである．開発言語は Java である．

2.2 iOS

iOS は，Apple 社の携帯端末を対象としたプラットフォームである．開発言語には Objective-C を用いる．

3 アプリケーションアーキテクチャの設計

Android，iOS プラットフォームを用いてアプリケーションアーキテクチャの設計をおこなう．事例として，TwitterAPI・FacebookAPI・MixiAPI を用いたアプリケーションを扱う．

3.1 Twitter アプリケーションのアーキテクチャ設計

TwitterAPI を用いたアプリケーションのアーキテクチャ設計をおこなう．本研究では，Twitter にアクセスし指定したユーザのつぶやきを取得し画面に表示するアプリケーションを作成する．Twitter からの受信データは XML 形式である．Twitter から受信した XML データを解析し，ユーザ名とつぶやきを抽出する．抽出した情報を端末の画面に表示する．画面には更新ボタンを配置し，更新ボタンを押下された場合，つぶやきを再び Twitter から取得し，端末の画面に再び出力する．

Twitter アプリケーションのコンポーネントを定義する．画面にユーザ情報を表示する View，入力イベントを受け取る Controller，Twitter にアクセスし XML データを取得する XMLAcquire，XML データを解析する XMLParser，更新通知用インタフェースの Observer が Twitter アプリケーションで用いるコンポーネントである．

3.1.1 Android プラットフォームを用いた Twitter アプリケーションのアーキテクチャ設計

Android プラットフォームを用いて Twitter アプリケーションアーキテクチャの設計をおこなう．Andoird では，Activity を用いてユーザへの表示を管理する．Activity を継承したクラスを作成し，ユーザが操作可能な画面と操作に対する処理を記述する．本研究で作成する Twitter アプリケーションはユーザ名とつぶやきを表示するアプリケーションである．ユーザのつぶやきは複数存在し，Twitter アプリケーションではつぶやきをリスト状に表示する．Android では，複数のデータをリスト状に表示する際に ListActivity を用いる．このことから，ListActivity クラスを継承した MainActivity を作成する．Android では，画面のレイアウトを XML で記述する．XML で記述したテキストボックスやボタンに ID を設定し，Activity から ID を参照することでテキストボックスやボタンのオブジェクトを取得する．本研究で作成する Twitter アプリケーションでは，MainActivity にボタンリスナを実装する．Twitter アプリケーションで用いる View，Controller が MainActivity に相当する．XMLAcquire は，

HttpURLConnection クラスを用いて XML データを取得し XMLParser に XML データを送信する．XMLParser は，XmlPullParser クラスを用いて XML データを解析しユーザ名とつぶやきを抽出する．解析結果を参照するために getUserData メソッド，ユーザ情報取得処理の終了通知をおこなう notify メソッドを実装する．以上の結果から，Android プラットフォームを用いて設計した Twitter アプリケーションのアーキテクチャを図 1 に示す．

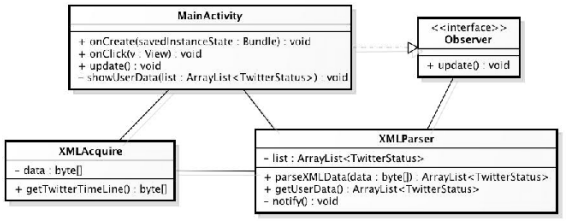

図 1 Android プラットフォームを用いて設計した Twitter アプリケーションのアーキテクチャ

3.1.2 iOS プラットフォームを用いた Twitter アプリケーションのアーキテクチャ設計

iOS プラットフォームを用いて Twitter アプリケーションアーキテクチャの設計をおこなう．iOS では，Android における Activity に対応する UIViewController を用いて View を制御する．複数データをリスト状に表示する際には UITableViewController を用いる．このことから，UITableViewController クラスを継承した RootViewController クラスを作成する．画面からの入力を受け取りは UITableViewController がおこなう．Android と同様に，View と Controller が UITableViewController に相当する．XMLAcquire では NSURLConnection クラスを用いてデータ取得，XMLParser では NSXMLParser クラスを用いて XML データ解析をおこなう．iOS では，Observer パターンの実現に NSNotification クラスを用いる．NotificationCenter に Observer となるオブジェクト，通知を受け取った際に実行するメソッド，通知元のオブジェクトを登録し，XMLParser が NotificationCenter に通知をおこなうことで実現する．以上の結果から，iOS プラットフォームを用いて設計した Twitter アプリケーションのアーキテクチャを図 2 に示す．

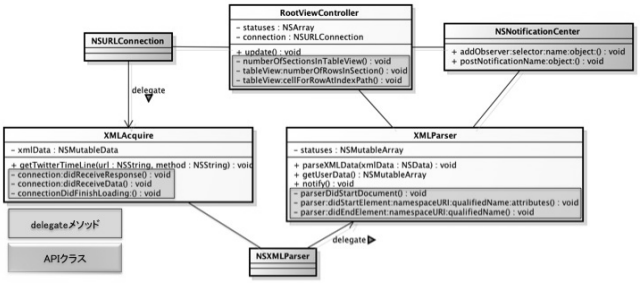

図 2 iOS プラットフォームを用いた Twitter アプリケーションのアーキテクチャ

3.2 Facebook アプリケーションのアーキテクチャ設計

FacebookAPI を用いたアプリケーションのアーキテクチャ設計をおこなう．本研究で作成する Facebook アプリケーションは，Facebook にログイン後ユーザ情報を取得し，端末の画面に表示する．Facebook へのログインのために端末の画面にダイアログを表示する．Facebook からの受信データは JSON 形式である．Facebook から受信した JSON データを解析し，ユーザ名・出身地・性別を抽出し結果を画面に出力する．

Facebook アプリケーションのコンポーネントを定義する．ログイン用のダイアログを表示する LoginView，ユーザ情報を表示する ShowUserDataView，ユーザのログイン情報を受け取る Controller，Facebook にアクセスし JSON データを取得する JSONAcquire，JSON データを解析する JSONParser，更新通知用インタフェースの Observer が Facebook アプリケーションで用いるコンポーネントである．

3.2.1 Android プラットフォームを用いた Facebook アプリケーションのアーキテクチャ設計

Android プラットフォームを用いて Facebook アプリケーションのアーキテクチャ設計をおこなう．Twitter アプリケーションと同様に，Activity を用いて端末の画面への表示，入力の受け取りをおこなう．Activity は画面と 1 対 1 で対応することから，Facebook アプリケーションではログイン用の画面を表示する LoginActivity と，ユーザ情報を表示する ShowUserDataActivity を作成する．Facebook アプリケーションで用いる LoginView，Controller が LoginActivity に，ShowUserDataView，Controller が ShowUserDataActivity に相当する．Android では画面遷移のためのコンポーネントとして Intent が提供されている．Intent に遷移先の画面に対応する Activity を登録することで，画面遷移を実現する．Intent は画面間でデータの送受信にも用いられる．解析結果を Intent に設定することで画面間のデータの送信を実現する．JSONAcquire は，Facebook に対して HTTP リクエストを実行しプロフィールを取得し JSONParser に送信する．JSONParser では JSON データを解析し，ユーザ情報を取得する．解析結果を参照するために getUserData メソッド，ユーザ情報取得処理の終了通知をおこなう notify メソッドを実装する．Android プラットフォームを用いて設計した Facebook アプリケーションのアーキテクチャを図 3 に示す．

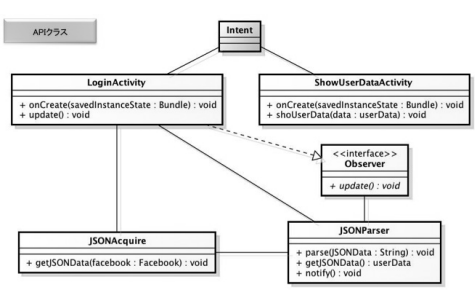

図 3 Android プラットフォームを用いて設計した Facebook アプリケーションのアーキテクチャ

3.2.2 iOS プラットフォームを用いた Facebook アプリケーションのアーキテクチャ設計

iOS プラットフォームを用いて Facebook アプリケーションアーキテクチャの設計をおこなう．Twitter アプリケーションと同様に，UITableViewController クラスを継承したクラスを作成する．UITableViewController を継承したクラスと画面は 1 対 1 で対応することから，Android と同様にログインするための画面を表示する LoginViewController とユーザデータを表示する ShowUserDataViewController を作成する．iOS で画面遷移の実現は StoryBoard を用いる．Storyboard は，画面間の遷移を管理するために提供されているコンポーネントである．Storyboard に遷移先の ViewController を設定することで，画面遷移を実現する．JSONAcquire では，Facebook にアクセスし指定したユーザのプロフィールを取得する．JSONParser は，JJSONAcquire から受けとったユーザデータを解析し，必要な情報抽出する．LoginViewController に対してデータを渡す getUserData メソッドを実装する．Twitter アプリケーションと同様に，Observer パターンの実現には NSNotificationCenter を用いる．iOS プラットフォームを用いて設計した Facebook アプリケーションのアーキテクチャを図 4 に示す．

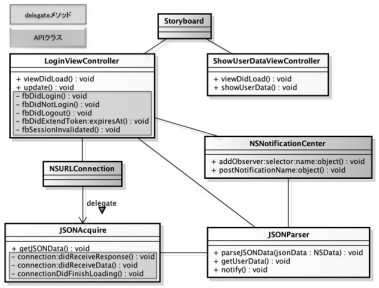

図 4 iOS プラットフォームを用いて設計した Facebook アプリケーションのアーキテクチャ

3.3 Mixi アプリケーションのアーキテクチャ設計

MixiAPI を用いたアプリケーションのアーキテクチャ設計をおこなう．作成する Mixi アプリケーションは Mixi にログイン後，入力されたつぶやきを投稿する．ログインするためのダイアログを端末の画面に表示し，ログイン後端末につぶやき入力画面を表示する．

Mixi アプリケーションのコンポーネントを定義する．ログイン画面を表示する LoginView，つぶやきの入力画面を表示する InputMixiVoiceView，ユーザのログイン情報，つぶやきを受け取る Controller，つぶやきを投稿する MixiVoiceWriter，更新通知用インタフェースの Observer が Mixi アプリケーションで用いるコンポーネントである．

3.3.1 Android プラットフォームを用いた Mixi アプリケーションのアーキテクチャ設計

Android プラットフォームを用いて Mixi アプリケーションアーキテクチャの設計をおこなう．Twitter・Facebook アプリケーションと同様に，Mixi アプリケーションにおいても Activity を用いる．Mixi アプリケーションでは，ログインするための画面とユーザがつぶやきを入力する画面が必要となる．このことから，ログインするための画面として LoginActivity，つぶやきを入力する画面として InputMixiVoiceActivity を作成する．Mixi にログイン完了後，ユーザがつぶやきを入力する画面に遷移する．画面遷移には，Facebook アプリケーションと同様 Intent を用いる．MixiVoiceWriter では，Mixi にアクセスしつぶやきを投稿する．Observer パターン実現のために Observer インタフェースを作成し，つぶやき投稿が完了したことを終了通知をおこなう notify メソッドを MixiVoiceWriter に，InputMixiVoiceActivity に update メソッドを実装する．Android プラットフォームを用いて設計した Mixi アプリケーションのアーキテクチャを図 5 に示す．

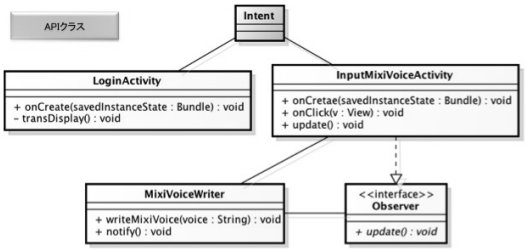

図 5 Android プラットフォームを用いて設計した Mixi アプリケーションのアーキテクチャ

3.3.2 iOS プラットフォームを用いた Mixi アプリケーションのアーキテクチャ設計

iOS プラットフォームを用いて Mixi アプリケーションアーキテクチャの設計をおこなう．Mixi アプリケーションにおいても，UIViewController を用いて View の制御をおこなう．Mixi アプリケーションでは，Mixi にログインするための画面とつぶやきを入力する画面が必要となることから，UITableViewController を継承した LoginViewController と InputMixiVoiceViewController を作成する．Facebook アプリケーションと同様に，LoginViewController から InputMixiVoiceViewController への画面遷移には StoryBoard を用いる．MixiVoiceWriter では Android プラットフォームを用いて設計した Mixi アプリケーションと同様に実現する．Observer パターンの実現には NSNotificationCenter を用いる．iOS プラットフォームを用いて設計した Mixi アプリケーションのアーキテクチャを図 6 に示す．

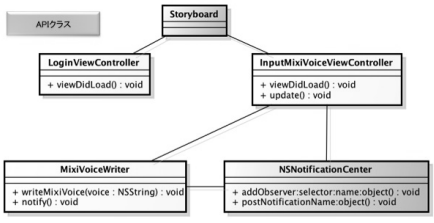

図 6 iOS プラットフォームを用いて設計した Mixi アプリケーションのアーキテクチャ

4　考察

4.1　Twitter アプリケーションアーキテクチャの比較

MVC における View について比較をおこなう．画面のレイアウトは Android，iOS プラットフォーム共に XML ファイルを用いて記述する．XML で記述したレイアウトに対して，Android プラットフォームでは MainActivity から，iOS プラットフォームでは RootViewController から表示するデータの処理をおこなう．このことから，Android プラットフォームでは MainActivity，iOS プラットフォームでは RootViewController が MVC における View に対応すると考える．

Controller について比較をおこなう．Android プラットフォームでは MainActivity が更新ボタンからの入力を受け取る．iOS プラットフォームでは，RootViewController が更新ボタンの入力イベントを受け取る．このことから，Android プラットフォームでは MainActivity，iOS プラットフォームでは RootViewController が Controller に対応すると考える．また，View の結果をふまえると，Android プラットフォームでの MainActivity，iOS プラットフォームでの RootViewController が ViewController となると考える．Model の要素である XMLAcquire と XMLParser は，共に Android・iOS 各プラットフォームで同様に実現することができた．

Observer パターンについて比較をおこなう．Android プラットフォームでは，Java を用いて Observer パターンを実現した．iOS プラットフォームでは，NSNotificationCenter を用いて Observer パターンを実現しており，NSNotificationCenter が Observer となるオブジェクトを保持する．Android プラットフォームでは，XMLParser が Observer を保持する．Android・iOS 各プラットフォームでは，Observer を保持するクラスに違いがあるがアーキテクチャには影響しないと考える．

4.2　Facebook アプリケーションアーキテクチャの比較

MVC における View と Controller に関しては，Twitter アプリケーションと同様に Android プラットフォームでの LoginActivity・ShowUserDataActivity，iOS プラットフォームでの LoginViewController・ShowUserDataViewController が ViewController となると考える．Model の要素である JSONAcquire と JSONParser，Observer パターンは Android・iOS 各プラットフォームで同様に実現することができた．

画面遷移の実現について考察する．Android プラットフォームでは，Intent を用いて画面遷移を実現した．Intent オブジェクトに遷移先の画面に対応する Activity を登録することで，画面遷移が実現可能となる．iOS プラットフォームでは，画面遷移の実現に Storyboard を用いた．Storyboard に遷移先の ViewController を設定することで，画面遷移を実現する．このことから，Android・iOS 各プラットフォームにおける画面遷移の実現は同様の構造で実現できると考える．

4.3　Mixi アプリケーションアーキテクチャの比較

MVC における View と Controller に関しては，Twitter・Facebook アプリケーションと同様に，Android プラットフォームでの LoginActivity・InputMixiVoiceActivity，iOS プラットフォームでの LoginViewController・InputMixiVoiceViewController が ViewController となると考える．Model の要素である MixiVoiceWriter，Model の更新通知・画面遷移 Android・iOS 各プラットフォームで同様に実現することができた．

4.4　アプリケーションアーキテクチャの共通構造定義

Twitter，Facebook，Mixi のアプリケーションアーキテクチャの比較結果を用いて，携帯端末アプリケーションの共通構造を定義する．各アプリケーションにおいてひとつの画面に対してひとつの ViewController が存在する．画面遷移を実現する場合，Android では Intent，iOS では Storyboard を用いている．実現するコンポーネントに違いはあるが構造は同様であることから DisplayTransitionComponent とする．ViewContorller と DisplayTransitionComponent の関係は多対一である．Model の更新通知も同様の構造で実現できた．以上のことから，携帯端末アプリケーションの共通構造を図 7 に示す．

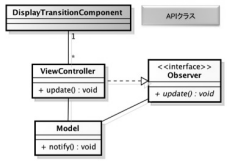

図 7　携帯端末アプリケーションの共通構造

携帯端末アプリケーションのアーキテクチャを設計する際には，図 7 に示した共通構造をもとにアプリケーションの設計をおこない，その後各携帯端末プラットフォームを考慮して詳細設計をおこなうことが可能となると考える．

5　おわりに

本研究では，Android・iOS プラットフォームを用いて，アプリケーションアーキテクチャの設計をおこなった．設計したアプリケーションアーキテクチャを比較することで共通構造を明確にした．今後の課題として，複数のアプリケーションが関連する事例を用いたアーキテクチャの比較が挙げられる．

参考文献

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture A System of Patterns,* John Wiley, 1996.

[2] Google Inc,“ Android Developers,” http://developer.android.com/, 2012.

[3] Apple Inc,“ iOS Developer Program,” https://developer.apple.com/, 2012.