# Microservicesのモバイルアプリケーション適用事例

齋藤　暢郎[1]

**概要**:モバイルアプリケーション開発においてMicroservices相当の実装を行うことで、複数の開発者が同時に着手可能な作業を拡大することを目的とする。実装や企業の戦略から見た利点、実現方法について紹介する。Microservicesはマーチン・ファウラーらが2014年に記事を公開して大きく知られたプログラミングアーキテクチャである。Webサービスにおいてはバックエンドでの採用例が増えており、フロントエンドにおいてもWebComponents、Microfrontendsとしてで同様の考え方が実現されている。モバイルアプリケーションにおける前例としてはUber社のRIBsがあげられる。Microservices: https://martinfowler.com/articles/microservices.html RIBs: https://github.com/uber/RIBs

**キーワード**：プログラミング・シンポジウム，夏，Microserices, モバイルアプリケーション

## 1. はじめに

モバイルアプリケーション開発においてMicroservices[1]相当の実装を行う事で、複数の開発者が同時に着手可能な作業量を増やし、より多くの開発者が一つのアプリケーション、一つの画面を開発することが可能になる。従来のモバイルアプリケーション開発ではレイヤー分けを行い分担するものが主流であったが、一つの画面が巨大な仕様を持つケースにおいては十分な選択肢ではなかった。そこで一つの画面を複数の細かな単位、コンポーネントに分割し、それぞれのコンポーネントに責務を移譲することで目的の達成を目指した。実装したプラットフォームはiOS/UIKitである。

## 2. 提案手法

UIKitにおいて画面を構成するクラスはUIViewControllerである。UIViewControllerは、画面の見た目を決定するクラス、UIViewのラッパーであり、画面遷移やイベントハンドリングの責務を持つ。従来のiOSアプリケーション開発においては、このUIViewControllerの実装をレイヤー毎のクラスに分離して行うものが主流である。例えばレンダリングを担当するクラス、ロジックを決定するクラス、表示するデータを定義するクラスに分離する。(図1) しかし一つの画面そのものが巨大である場合、各レイヤー間の情報伝達が肥大化する、或いは開発者の人数を増やすことが、レイヤーの数に依存して上限が定まるという問題が発生する。提案手法では、単一の画面を単一のUIViewControllerとして開発するのではなく、より細かいUIコンポーネント毎のUIViewControllerの集合として開発することで、情報伝達の肥大化や開発者の上限を解決する。UIViewControllerがDOMのように親子関係を持つことが出来る点に着目し、一

[1] 株式会社メルカリ

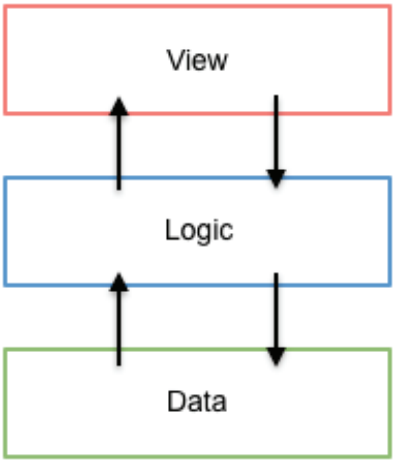


図 1　一つの画面におけるレイヤー分離の例

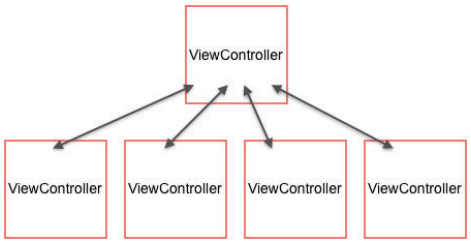


図 2　複数のコンポーネントとして一つの画面を分離

つ一つの UI コンポーネントを UIViewController として定義、画面遷移やイベントハンドリングの責務を UI コンポーネントに分割、移譲した。(図 2) また、UI コンポーネントの入出力のインターフェイスと、UIViewController の親子関係の定義を隠蔽したフレームワークを定義し、入出力のインターフェイスを Proxy することで、任意の UI コンポーネントを UIViewController として定義することが可能になった。UI コンポーネントの定義において UIViewController を利用しているため、UIKit が提供する機能をすべて直接利用することが出来る。画面内における複数の UI コンポーネントのが独立して動作、開発が可能になることで、同時に同一の画面における作業可能な開発者の上限を大きく増やすことが出来る。提案手法は MicroViewController と呼称し、簡易に実装するためのライブラリを公開 [2] した。MicroViewController は、UIViewController をコンポーネントとして定義し、責務を分割移譲する、という点のみを提案し、アプリケーション全体のアーキテクチャを指定するものではない。UI コンポーネントを全て UIViewController として提供するため、データを定義するクラスにおいて従来の Dependency Injection に代表される手法ではテストや実装時の開発者の負担が大きくなる。そこでアプリケーションのデータ層において、単一の Gateway を提供し、すべてのデータへのアクセスを関数によって管理する。Swift において protocol 内に定義されたオーバーロードされた関数は、実装時に単一の Generic 関数で代替可能な性質がある。(図 3) これを利用することで、Generic な関数によるデフォルトの Stub 実装を提供し、テストコードの記

```
protocol SomeProtocol {
  func foo() -> String
  func foo() -> Int
}

class SomeImpl: SomeProtocol {
  func foo<T>() -> T { ... }
}
```

図 3 Swift における Generics によるオーバーロードの解決

述を簡易化した。

## 3. 成果

提案手法の実装前後において単一アプリケーションのリポジトリにおいて以下の成果を得られた。実装以降 5 月から 8 月において月当たりのコード追加行数が 3 万〜4 万行、PullRequest の数は 300〜400、Commit の数は 1000〜2000 であった。週あたりの Commit 数のピークは約 250(作業者 13 人) から約 480(作業者 20 人) へ増加した。また特定の画面において、レンダリングが 1.5 秒から 0.5 秒に改善される結果が得られた。これは複雑な画面におけるレンダリングの最適化が複数人の作業では困難であったものが、コンポーネントが分割され作業が分割されたことにより簡易になったことが原因であると考える。

## 4. 関連技術の紹介

iOS における先行実装の例として、Uber 社の RIBs[3] が上げられる。これは提案手法とは異なり、アプリケーション全体のアーキテクチャに言及するものである。また Web フロントエンドの技術としては WebComponents[4] や Micro Frontends[5] の提案がある。

## 5. まとめ

モバイルアプリケーション開発における Microservices 相当の実装を提案した。一つの画面を細かなコンポーネントの集合として設計することで、複数の開発者による作業効率の向上を実現した。iOS における実装を MicroViewController と定義し、ソースコードを公開した。

## 6. 質疑応答

Q. コミュニケーションによるオーバーヘッドはないのか?
A. プロジェクトの企画段階でコンフリクトしていることもある。今の所避けれていない。提案手法の導入もコストが高い、課題はある。
Q. パフォーマンスの話があるが、元のやり方が悪かったのでは?
A. 画面全体を巨大なコンポーネントとして開発すると、どうしても関連するレイアウトのルールで破綻が出てきてしまう。細かく作ることでそれを避けれる効果があるのではないか。
Q. 開発効率は 2 倍程度なのか
A. 一人あたりの効率よりも全体で効率を上げられるようになったという価値があると考えている。
Q. コンフリクトが減ったのは評価できない?
A. そもそもコンフリクトが起きないように数は抑えていたと思う、比較はできない。
Q. 親からプロパティを引っ張ってくる仕組みが他のプラットフォーム (Swing) にはある、そういうのはないか。
A. ユーザの行動によって書き換えたり書き換えなかったりといったことが発生するので、明示的にコピーしたほうが良いと考えている。
Q.Generics によって Stub を作っているのは開発効率のため?
A. 開発効率のため。また Generics というよりは型クラスの用途、Request と Response のペアを定義したい。
Q.View を細分化するレベルについての知見はない?
A. ユーザがコントロールする UI レベルで分割するとうまくいくと考えている。操作の必要のないラベル単位で分割するとうまくいくかはわかってない。
Q.2 つのコンポーネントが連動する場合は?
A. 親が管理するようにしている。

参考文献

[1] Microservices https://martinfowler.com/articles/microservices.html
[2] Mew https://github.com/mercari/Mew
[3] RIBs https://github.com/uber/RIBs
[4] WEBCOMPONENTS.ORG
https://www.webcomponents.org
[5] MicroFrontends https://micro-frontends.org