スマートデバイスアプリケーションのための アーキテクチャに関する研究
—画面表示論理に関する考察—

2011SE045 平松和彦 2011SE203 小田航己 2011SE249 鈴木崇弘
指導教員：野呂昌満

1　はじめに

スマートデバイス向けアプリケーション開発において，開発者は開発対象とするプラットフォーム毎に異なる実行時環境と開発環境を用いる必要がある．複数のプラットフォームを開発対象とし，同様な機能を持つアプリケーションを開発するさい，それぞれの実行時環境や開発環境を用いる技術が必要であり，開発者の負担となる．

本研究室では，スマートデバイス向けアプリケーションのための共通アーキテクチャが提案されている．共通アーキテクチャは，Web アプリケーションとネイティブアプリケーションの統一的な開発を可能にすることを目的としている．これにより開発者は，共通アーキテクチャが説明可能としている実行時環境の一つを理解していれば，異なる実行時環境のアプリケーションを開発することが可能となる．

本研究では，共通アーキテクチャにおける画面表示論理に関する部分の妥当性確認を行なうことを目的とする．View 部分はアプリケーション開発における主要部分であり，View 部分の開発は画面構成の開発が大部分を占める．よって，画面表示論理に関する部分の統一的な開発を可能にすることで，アプリケーション開発のコスト削減につながると考える．これにより，共通アーキテクチャで説明可能な実行時環境であれば，同様な画面構築が容易となり，アプリケーション開発のコスト削減につながる．

共通アーキテクチャの View 部分をトップダウンとボトムアップの視点で，妥当性確認を行なう．トップダウンの視点として，一般的なアーキテクチャと比較を行なう．ボトムアップの視点として，共通アーキテクチャに基づき，既存の実行時環境を調査の対象にした画面表示論理に関する中間表現を定義する．

2　本研究の前提とするアーキテクチャ

本研究室では，スマートデバイス向けアプリケーションのための共通アーキテクチャが提案されている．共通アーキテクチャは，Web アプリケーションとネイティブアプリケーションの統一的な開発を可能にすることを目的としている．図 1 は，共通アーキテクチャを Component Connector で記述したものである．共通アーキテクチャは，各実行時環境ごとのアーキテクチャ間における中間表現として定義されている．

アプリケーション開発は，アーキテクチャを理解することで，開発方法が特定できる．よって，中間表現を定義することで，ある実行時環境の開発方法で他の実行時環境の開発を行なうことが出来る．

以下に ViewConcern におけるコンポーネントの責務を示す．

ViewContent　画面の構造及び内容

DisplayImageContent　画面部品の役割

Style　画面の見栄え

ViewConstructor　ViewContent と Model との対応付け

DisplayImageConstructor　構造，役割，見栄えの対応付け

構造，役割，見栄えについては，次のように定義されている．例を示すと，Web アプリケーションにおいては，HTML で定義しているものが構造と役割にあたり，CSS で定義しているものが見栄えにあたる．

構造　画面の構成．内容の相対的な位置関係．

役割　画面部品としての振舞いや機能．内容の表現方法．

見栄え　画面の装飾に関する情報．色や大きさなど．

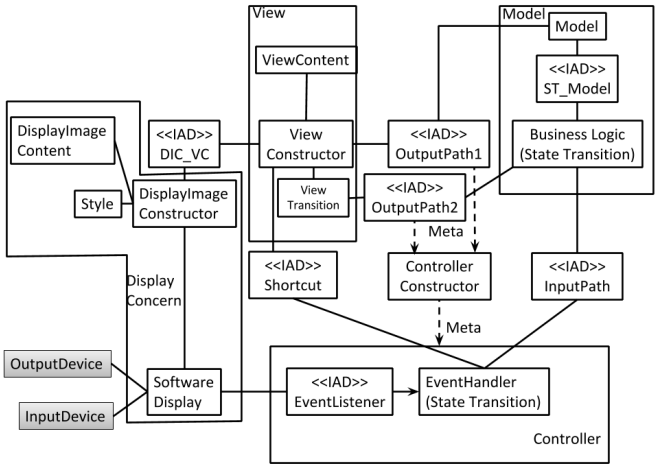

図 1　共通アーキテクチャ

3　研究手順

トップダウンの視点として，共通アーキテクチャを一般的なアーキテクチャと比較し，画面表示論理に関する部分の妥当性確認を行なう．K.Sokolova らの研究 [5] で提示している一般的なアーキテクチャを対象にし，比較を行なう．

- PAC(Presentation-Abstraction-Control)
- HMVC(Hierarchical-Model-View-Controller)

- MVP(Model-View-Presenter)
- MVVM(Model-View-ViewModel)

ボトムアップの視点として，共通アーキテクチャに基づき，既存の実行時環境を調査の対象にした画面表示論理に関する中間表現を定義を行なう．調査の対象を，一般に広く使われている Android，iOS 及び Web アプリケーションとして Ruby on Rails とする．画面表示論理に関する中間表現を定義するためには，実行時環境間で画面部品を対応付けるために，画面部品の分類を行なう必要がある．画面部品には多くの場合，同様の動作をする画面部品でも，実行時環境によって異なる名称がつけられており，名称によって分類することはできない．共通アーキテクチャは，構造と役割による分離によって，画面部品の特定を可能にすることを考慮し，設計されている．これに基づき中間表現を定義することで，構造と役割に基づく分類により画面部品が適切に対応付けられることを確認する．

4 一般的なアーキテクチャとの比較

共通アーキテクチャが一般的なアーキテクチャを説明可能であることを示すために，いくつかのアーキテクチャと比較を行なう．説明可能であるとは，アーキテクチャのコンポーネントの機能とその間の関係を，共通アーキテクチャのそれに対応付けられることを指す．

4.1 PAC との比較

PAC は，MVC のモジュール性を向上させることを目的に作られている．個々の機能を担うまとまりをエージェントとして定義し，アプリケーションをエージェントの階層構造で表現する．PAC のエージェントは，Presentation，Abstraction，Control の 3 つのコンポーネントから成る．Abstruction は，エージェントとしての機能とそれに関わる情報を保持する．Presentaion は，Abstruction の情報を表示するコンポーネントであり，UI の構造とプレゼンテーションロジックを担う．Control は，Presentation と Abstruction との連携や他のエージェントとの連携を担う．これにより，機能単位でのモジュール性を向上させている．

PAC における Presentation は，UI の構造及び外観，プレゼンテーションロジックを含有している．対して共通アーキテクチャの View 部分では，プレゼンテーションロジックは ViewConstructor に記述する．UI の構造は ViewContent，DisplayImageContent で記述し，外観は Style に記述する．よって，PAC は共通アーキテクチャで説明可能である．

4.2 HMVC との比較

HMVC は，画面構成の画面部品単位での階層関係をアプリケーションの構成に反映させることで，画面部品単位でのモジュール性を高めたアーキテクチャである．画面部品単位で Model,View，Controller から成る層を作り，層同士は Controller を通じて通信する．

HMVC では，View は UI とプレゼンテーションロジックを含有している．対して共通アーキテクチャでは，UI の構造を ViewContent で記述し，画面部品単位での階層関係を保持する．また，プレゼンテーションロジックは ViewConstructor で記述する．よって，HMVC は共通アーキテクチャで説明可能である．

4.3 MVP との比較

MVP は，MVC をイベント駆動システムに適用させたアーキテクチャとして提案されており，画面表示とプレゼンテーションロジックを分離した構造をもつ．View は画面表示とユーザからのイベントの振り分けを担い，Presenter はプレゼンテーションロジックのみを担う．共通アーキテクチャでは，プレゼンテーションロジックを ViewConstructor 実現を行なう．また，画面表示とイベントを振り分けるコンポーネントとして，DisplayConcern における Software Display と Controller Concern における，Event Handler で表現している．また MVP では，Presenter と View とで UI とプレゼンテーションロジックを分離している．共通アーキテクチャでも，UI の定義とプレゼンテーションロジックを分離している．UI の定義は，ViewContent，DisplayImageContent 及び Style で記述し，プレゼンテーションロジックは ViewConstructor で記述する．よって，MVP は共通アーキテクチャで説明可能である．

4.4 MVVM との比較

MVVM は，プレゼンテーションロジックとビジネスロジックを分離するためのアーキテクチャである．View に UI の外見と構造を分離し，ViewModel にはプレゼンテーションロジックと View の状態を分離している．Model にはビジネスロジックとデータを分離している．View に入力があった場合は，View から ViewModel へ通信を行い，ViewModel が Model を変更する．Model に変更があった場合は，Model が ViewModel に通知を行い，ViewModel から View への通知により View が更新される．

MVVM では，ビューの表示を View で記述し，ViewModel にビューの状態を分離している．また MVVM では，コマンドによりビューとモデルを疎結合にしている．対して共通アーキテクチャでは，ビューの表示は ViewContent と Style で記述する．また，OutputPath1 によりビューとモデルを疎結合にしている．よって，MVVM は共通アーキテクチャで説明可能である．

4.5 考察

共通アーキテクチャが比較対象としたアーキテクチャ全てについて説明可能であることが確認できた．共通アーキテクチャが一般的なアーキテクチャの持つ利点を含有していることを確認できた．MVP において，Presenter と View により UI とプレゼンテーションロジックの分離を実現している．また MVVM おいては，コマンドによりモ

デルとビューを疎結合にしている．対して共通アーキテクチャでは，UI を ViewContent と DisplayImageContent 及び Style に記述し，プレゼンテーションを ViewConstructor に記述することで分離を実現している．また，OutputPath1 によりモデルとビューを疎結合にしている．したがって，共通アーキテクチャはこれまでに提案されている Web アプリケーションとネイティブアプリケーションのアーキテクチャを説明可能であると言える．

5　共通アーキテクチャの適応可能性確認

共通アーキテクチャにおける画面構成部分に関して，詳細化を行なうことで適応可能性の確認を行なう．共通アーキテクチャにおける ViewContent と DisplayImageContent のデータ構造の提案を行なう．データ構造を提案する上で，実行時環境の画面部品が持つ内容と役割を特定する必要がある．特定した内容と役割から実行時環境間の画面部品を対応付け，同じ機能を持つ画面部品として整理し，分離方法の妥当性を確認する．

5.1　画面部品の分類

画面部品の内容と役割に基づき，いくつかの実行時環境を対象として画面部品の分類を行なった．調査の対象とする実行時環境を Android，iOS，Ruby on Rails とし，各実行時環境で提供されている画面部品の内容と役割を特定した．画面構成の記述は，Android，iOS においては，それぞれで提供されているライブラリ [4][2] を使用し，Ruby on Rails では，HTML[6] を用いる．内容と役割が一致するものを同じ動作をする画面部品と考え，分類を行なった．対応表の一部を表 1 に示す．これにより，内容と役割による分類が適切にできたと考える．

表 1　画面部品の分類 (一部)

入出力	役割	内容	HTML	iOS	Android
入力	単体	文字列	<input type="button">	UIButton	Button
		画像	<audio><video>で controlsを指定	AVFoundation	MediaControler
		文字列	1行は<input type="text">, 複数行は<textarea>	UITextField	EditText
	選択／単体選択	真偽	<input type="radio">	UISwitch	ToggleButton
	選択／複数選択	真偽	<input type="checkbox">	UISwitch	Switch
		真偽	<input type="checkbox">	UIButton	CheckBox
出力	単体	文字列	x	1行はUILabel, 複数行はUITextView	TextView
		画像	<img>	UIImageView	ImageView
		動画	<video>	MPMoviePlayerController	VideoView

5.2　ViewContent

ViewContent は画面の構造と内容を表現するコンポーネントである．画面の構造は画面部品の相対的な関係により表現可能である．これは HTML が木構造として画面の構成を行なっていることから流用可能であることを判断した．Composite パターンを適応することで，木構造を表現可能にした [3]．また内容については，画面部品の分類から保持可能な内容を木構造の葉として定義する．以下が，画面部品が保持すべき内容である．

表 2　画面部品が保持する内容

文字列	開発者が表示させたい文字，入力される文字列等
数値	入出力をする数値
真偽値	スイッチの ON/OFF など
画像	ボタンの画像など
音声	音楽，効果音等
動画	画面部品の挙動
プログラム	動的な処理

これらの内容の集合により，日付や時刻などの内容を定義することが可能となる．ViewConcern のデータ構造を表すオブジェクトモデルとして図 2 に示す．

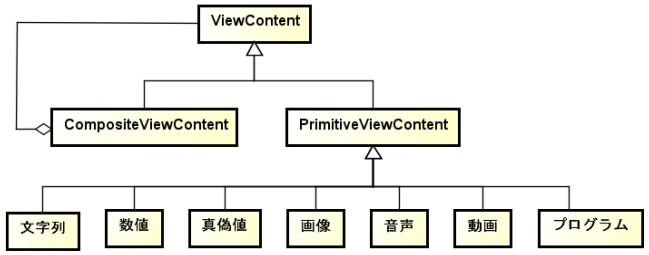

図 2　ViewContent のデータ構造

5.3　DisplayImageContent

DisplayImageContent は，ViewContent が保持する内容に付与する役割を表現するコンポーネントとして定義されている．ここでの役割とは，画面部品としての機能のことを指す．実行時環境の画面に内容をどのような形式で表示させるかを意味し，内容を表現する ViewContent に付与される．画面部品の整理結果をもとに，データ構造を提案する．内容の表現方法について，入出力による分類ができると考えた．入力に関しては，開発者があらかじめ選択肢を与えておくものと，ユーザが直接入力を行なうものがある．また，選択肢を与えておく場合，ユーザが選択する際，選択肢から 1 つを選択する場合と複数選択させる場合がある．また出力に関しては，リストや表として一覧と表示するものと単一で表示を行なうものがある．これらを基に画面部品の分類を行ない，データ構造を考案した．DisplayImageContent のデータ構造を表すオブジェクトモデルを図 3 に示す．

5.4　考察

本研究で対象としていない実行時環境が，共通アーキテクチャで説明可能であるかを確認する必要がある．確認の対象とするアーキテクチャとして Apache Struts[1] を挙げる．Apache Struts と共通アーキテクチャの View 部分との対応関係を示し，本研究で提案した分類方法による画面

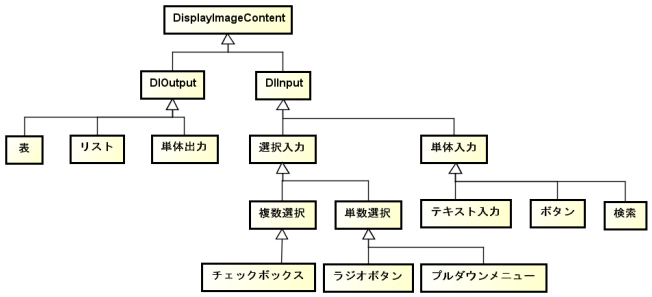

図 3 DisplayImageContent のデータ構造

部品の分類を行なうことで，説明可能かどうかを確認する．Struts のアーキテクチャにおいて，View 部分を担うコンポーネントは JSP と CustomTag である．JSP は画面構成を担うコンポーネントであり，開発者はこのコンポーネントに記述することで画面を作成する．CustomTag は，画面構成を作る際に開発者の開発を補助するためのタグライブラリである．コンポーネントの定義から共通アーキテクチャと対応させる．対応関係を図 4 に示す．

本研究で提案された分類方法に基づいた Struts で提供されている画面部品の分類を表 3 に示す．以上のことから本研究で対象としていない実行時環境についても共通アーキテクチャで説明可能であると考えた．

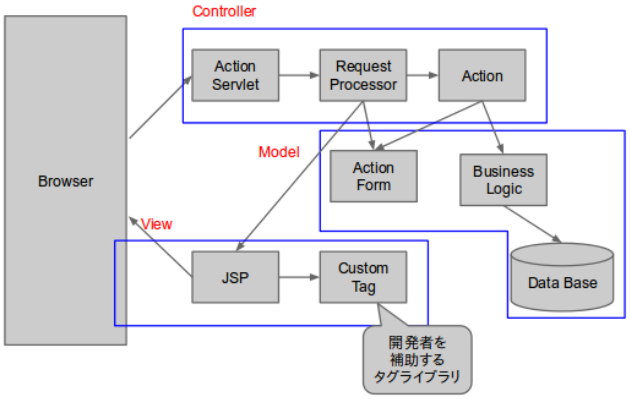

図 4 Struts のアーキテクチャと共通アーキテクチャとの対応関係

6 おわりに

本研究では共通アーキテクチャにおける画面表示論理に関する部分の妥当性をトップダウン，ボトムアップの視点から確認を行なった．トップダウンの視点から，一般的なアーキテクチャと比較を行なうことで，共通アーキテクチャが各アーキテクチャの特性を有することが確認でき，妥当性を示した．ボトムアップの視点からは，対象とする実行時環境を Ruby on Rails，iOS，Android とし，提供されている画面部品を内容と役割により整理，分類を行なった．画面部品の分類から各実行時環境の対応関係を示した表を作成し，これを基に ViewContent と DisplayImageContent のデータ構造を提案した．また，

表 3 Struts で提供されている画面部品の分類 (一部)

入出力	役割	役割	Apach Struts
入力	単体	文字列	<html:button>
		画像, 文字列	<html:image>
		文字列	<html:textarea>
	選択／単体選択	真偽	<html:radio>
	選択／複数選択	真偽	<html:checkbox>
		真偽	<html:checkbox>
出力	単体	画像	<html:img>
		動画	<html:video>

Struts を例に挙げ，本研究で対象としていない実行時環境でも適応が可能であることを確認した．これらのことから，共通アーキテクチャは Web アプリケーションとネイティブアプリケーションを説明するのに充分な抽象度であると結論づける．

今後の課題として，View 以外の部分を対応付ける必要があると考える．Controller 部分も，View 部分と同様に実行時環境や開発環境の差異が存在する．それにより，異なる実行時環境で同様な機能を持つアプリケーションを統一的に開発することが出来ない．よって，Controller 部分の統一的な開発を可能にすることで，アプリケーション開発のコスト削減につながると考える．

参考文献

[1] Apache Struts, “*Apache Struts2 Documentation Guides*,” https://struts. apache. org/, 2006.

[2] Apple Inc, “*iOS Developer*,” https://developer. apple. com/, 2014.

[3] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] Google Inc,“ *Android Developers*, ”http://developer. android. com/, 2014.

[5] K. Sokolova, M. Lemercier, and L. Garcia, “ Towards High Quality Applications: Android Passive MVC Architecture, ”*International Journal on Advances in Software*, vol. 7, no. 1&2, pp. 123-138, 2014.

[6] w3c, “ *HTML4.01 Specification*, ” http://www. w3. org/, 1999.