Vol.2015-SE-187 No.32
2015/3/13

インタラクティブソフトウェアの共通アーキテクチャの提案

江坂 篤侍[1]　野呂 昌満[2]　沢田 篤史[2]

概要：スマートデバイスや Web ブラウザの多様化は，そのソフトウェアの実行時環境や開発環境の多様化を引き起こし，一人の技術者がこれら環境すべてを把握することは難しく，これが生産性向上の妨げとなっている．我々は，参照アーキテクチャは開発環境を規定し，アプリケーションアーキテクチャは実行時環境を定義するとの認識のもと，共通参照アーキテクチャを設計し，それを詳細化して共通アプリケーションアーキテクチャを定義した．これらを既存の参照アーキテクチャならびに実行時環境を規定する既存のアプリケーションアーキテクチャと比較し，それらの関係を考察した．さらに，任意の実行時環境で稼働するアプリケーションの任意の開発環境を用いた作成支援の可能性を考察した．

A Common Architecture for Interactive Software

Abstract: A number of software development environments and runtime environments for variety of smart devices increasingly coming out and that of Web browsers have different shapes one from the other. The variety and the increasing number of environments cause lower quality and/or productivity of interactive software running on the devices and the browsers. The inherent problem of this can be solved to define the problem as that of software architecture. That is, a reference architecture defines a development process and then the process prescribes a development environment. An application architecture is, in turn, reflected by a runtime environment. We have constructed the common architecture which is a set of the common reference architecture and the common application architecture for interactive applications. We also sorted the relationships between the common reference architecture and existing reference architectures. Correspondence between the common application architecture and existing application architectures is also considered. We concluded that there are possibilities for supporting the software production in a development environment for a runtime environment being for a different development environment. We discuss that how the relationships defined and sorted contribute the development support.

1. はじめに

スマートデバイスや Web ブラウザの多様化に伴い，それらの上で稼働するインタラクティブソフトウェアの実行時環境や開発環境は多岐にわたるようになってきた．開発用のプログラミング言語を例にとっても，Java，Ruby，C#, Objective-C と多岐にわたる．これら多岐にわたる環境を一人のソフトウェア技術者がその詳細にわたりすべてを把握することは非常に難しい．さらに，これらの開発において，そのような技術者を多数確保することは困難である．この問題に対して，ワンソース (One Source) 開発を可能にするクロスプラットフォーム開発環境 [1] や関連技術の標準化 [3], [5] 等の研究・開発が盛んに行われてはいるものの，依然，環境の差異は生産性向上の障壁となっている．

特定の開発環境で別の開発環境が前提とする実行時環境上で稼働するソフトウェアが自動生成できれば，これらソフトウェアの生産性は向上する．開発環境はソフトウェアアーキテクチャにより規定され [2]，アプリケーションのアーキテクチャは実行時環境を規定する [8]．すなわち，環境の差異に起因する問題はソフトウェアアーキテクチャに関連する課題として定義できる．

本研究の目的は，インタラクティブソフトウェアの共通アーキテクチャを提案することである．共通アーキテクチャを共通参照アーキテクチャとして定義し，開発プロセスと開発環境との関連を考察する．

参照アーキテクチャの設計においては，MVC アーキテクチャとその派生である既存のアーキテクチャ [7] を調査し，それらのアーキテクチャが分離を試みている横断的関

[1] 南山大学大学院数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
[2] 南山大学情報理工学部ソフトウェア工学科
Department of Software Engineering, Nanzan University

心事を特定することで，アスペクト指向アーキテクチャとして統合する．共通参照アーキテクチャを詳細化し共通アプリケーションアーキテクチャを設計する．これを既存の実行時環境上のアプリケーションアーキテクチャと比較し，その構造を検討する．共通参照アーキテクチャと既存の開発環境ならびに共通アプリケーションアーキテクチャと既存の実行時環境との関係を整理する．任意の実行時環境で稼働するアプリケーションの任意の開発環境を用いた作成支援の可能性をこれらの関係を用いて考察する．

2. インタラクティブソフトウェアの開発の現状および問題点と解決策

2.1 開発の現状

インタラクティブソフトウェアは，Web アプリケーションとネイティブアプリケーションおよびこれらの組み合わせに分類される．Web アプリケーションは，Web ブラウザ内で稼働するプログラムとサーバ上で稼働するプログラムの協調により動作する．ネイティブアプリケーションは Web ブラウザを利用せずにユーザインタフェースを取り扱い，Web アプリケーションと同様にサーバ上のプログラムと協調して動作することが多い．

近年，インタラクティブソフトウェアは MVC アーキテクチャに基づいて開発されるようになってきた [6]．MVC アーキテクチャは，プレゼンテーションロジックからビジネスロジックを分離し，それぞれの独立な変更を可能にする [7]．この View，Controller，および Model の開発に際し，多岐にわたる技術が利用されている．

Web アプリケーションの View 定義技術として，HTML や CSS が代表例として挙げられる．ネイティブアプリケーションの View 定義には Swing や AWT などのライブラリが利用される．

Controller の実現においては，Web アプリケーションでは HTML 上で使用者イベントに対して起動するプログラムが指定できる仕組みが用意されており，CGI，PHP インタフェース，Servlet インタフェースがこれにあたる．ネイティブアプリケーションでは，Swing や AWT が使用者イベントに対してスクリプトやプログラムを実行する仕組みを提供している．

Web アプリケーションの Model の実現においては，CGI プログラム，PHP スクリプト，JavaServlet，JavaScript，ASP 等，多くの言語が用いられる．ネイティブアプリケーションの Model は，Java，Objective-C，C#等多くのプログラミング言語で実現されている．

Controller，View と Model 間の通信では，上記ライブラリやインタフェースが提供するプッシュ，プル通信だけではなく，非同期通信のための Ajax の LongPolling などの技術が利用されている．

AngularJS，Struts，Ruby on Rails，.Net や Spring 等のインタラクティブソフトウェアのためのアプリケーションフレームワークは，以上の Web 関連技術やネイティブアプリケーション用ライブラリ群を取捨選択して MVC アーキテクチャに基づくアプリケーション構築を支援している．

2.2 問題点

インタラクティブシステムの開発では，上述のような多様な開発技術について詳細にわたりすべてを把握することは難しい．HTML を用いた場合は View とユーザ操作による入力時の処理は同時に定義されるが，iOS アプリケーションフレームワークなどでは，これらが分離されている．HTML は，Web ブラウザのベンダが独自に拡張してタグを追加したり，JavaScript の言語仕様を変更してきた結果，プログラムのブラウザ間での互換性が保証できない場合が多い．オペレーティングシステムによって利用可能なプログラミング言語が制限されている場合には，利用可能なアプリケーションフレームワークやライブラリが異なる．このように，提案されている技術の互換性がないばかりか，通常，技術の選択権は技術者にはなく，どの技術を用いるかは，使用者や市場の要求さらには開発部門の方針によって決定される．

以上の問題に対してクロスプラットフォーム開発環境が実現，運用され，また Web 技術の標準化が行われている．クロスプラットフォーム開発環境は，異なるデバイスや OS 上で稼働するアプリケーションのワンソース開発を可能とするものである．Web 関連技術の標準化は，異なるブラウザに対して同様の表示や動作を可能にすることを目的としている．しかし，クロスプラットフォーム開発環境は，すべての実行時環境に対してワンソース開発を保証するものではなく，Web 標準が遵守されず．ブラウザベンダごとの方言ができることが，これまで幾度も繰り返されてきている．

要約すると，ある技術の組での開発に習熟した技術者は多数存在するが，多様な類似技術をすべて運用できる技術者は稀である．技術の多様性に起因して，特定のインタラクティブソフトウェアの開発にあたっては，一定以上の質の技術者を必要数確保することは簡単ではなく，結果として，生産性の低下を招いている．

2.3 解決策

環境の差異に起因する問題はソフトウェアアーキテクチャに関連する課題として定義できる．

一般に，ソフトウェアアーキテクチャは開発プロセスを含意している [2]．開発環境はこの開発プロセスを支援する環境として定義される．例えば，参照アーキテクチャである MVC アーキテクチャは，その構造から Model，View，Controller それぞれについて並行に開発が可能であることを示し，開発環境はこれを支援する．このように，ソフト

ウェアアーキテクチャは開発環境を規定するものである．

アプリケーションフレームワークは，アプリケーションアーキテクチャに基づいて特定の実行時環境の使い方を説明したコードを定義している．すなわち，フローズンスポットは実行環境の提供する API にしたがって実現されており，ホットスポットに埋め込まれるコードと実行時環境の隔たりを埋めるものである．

以上をまとめると，参照アーキテクチャと開発環境は関連し，アプリケーションアーキテクチャと実行時環境が関連している．参照アーキテクチャのプロセス側面は開発環境を構成する基本となり，アプリケーションアーキテクチャは実行時環境依存コードをフローズンスポットとして実現したものである．

開発環境が前提とするアーキテクチャについて，共通の参照アーキテクチャを定義し，それぞれのアーキテクチャと参照アーキテクチャの関係を整理することで，開発環境で用いられる技術間の対応関係を明らかにできる．実行時環境が前提とするアプリケーションアーキテクチャについても, 共通のアプリケーションアーキテクチャを定義し，それぞれのアプリケーションアーキテクチャと共通アプリケーションアーキテクチャの関係を整理することで，実行時環境間の対応関係を明らかにできる．この様子を図 1 に示す．すなわち，共通アーキテクチャを介して，環境の技術間の関係が明らかとなる．参照アーキテクチャの詳細化がアプリケーションアーキテクチャであるとの前提に立ち，これらの関係を整理することにより，任意の開発環境から，任意の実行時環境で稼働するアプリケーションを作成可能にする基礎が与えられる．

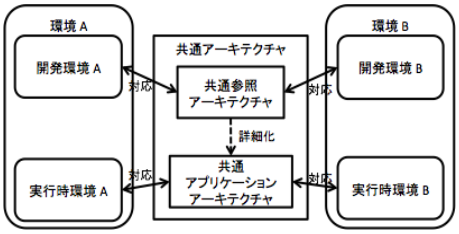

図 1 共通アーキテクチャと実行時環境および開発環境の関係

3. アーキテクチャ設計

ここでは，Clements らの複数のビューによるアーキテクチャ文書化に関する枠組み [4] に基づいてアーキテクチャを記述する．システムに対する関心事は複数存在し単一のビューだけで表現することは難しいことから，複数のビューによりアーキテクチャを記述する．この論文ではアーキテクチャの構造について議論しているので，モジュールビュータイプを用いて抽象ビューと具体ビューとして，それぞれ共通参照アーキテクチャと共通アプリケーションアーキテクチャを記述する．

共通参照アーキテクチャと共通アプリケーションアーキテクチャの設計は，

> “複数の既存のアーキテクチャをすべて説明可能とする．”

という設計思想に基づいて行なう．あるアーキテクチャが説明可能とは，アスペクト指向アーキテクチャとして定義された共通アーキテクチャにおいて，特定の横断的関心事のいくつかを指定して織込むことで，そのアーキテクチャが生成されることを指す．

共通アプリケーションアーキテクチャの設計は以下に示すコンポーネントの柔軟性を確保するという指針に基く．

- View
- Model
- Model，View，Controller 間の関連
- Controller

3.1 MVC アーキテクチャとその派生である既存のアーキテクチャの調査

共通参照アーキテクチャの設計にあたり，MVC アーキテクチャとその派生として AM-MVC，HMVC，MVP，PAC，MVVM について調査した [7]．

MVC アーキテクチャ (図 2) は，プレゼンテーションロジックからビジネスロジックを分離し，それぞれの独立な変更を可能にする．画面に表示される視覚的要素を扱う View，データとビジネスロジックを含むドメインモデルを扱う Model，ユーザ操作によるイベントの扱う Controller によって構成される．View と Model の関連は，変更を Model へのプッシュ通信または Model からのプル通信により認知し，画面を更新する Classic MVC と，View と Model の依存関係を無くし，Controller からのプッシュ通信を画面更新のきっかけとすることを前提とした Passive MVC がある．

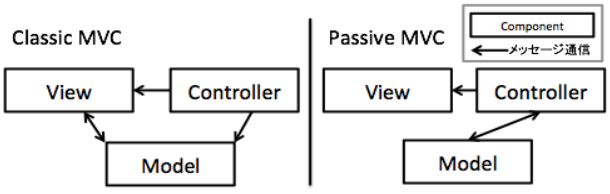

図 2 Model-View-Controller(MVC)

AM-MVC(図 3) は，MVC の View もしくは Controller に定義されるプレゼンテーションロジックを分離し，View と Controller を，画面出力とユーザ入力処理と画面構築の役割に分ける．MVC から分離したプレゼンテーションロジックの実行を役割としてもつ ApplicationModel を追加している．

HMVC(図 4) と PAC(図 5) は，階層構造を用いて記述することを目的としている．HMVC は，MVC の Controller

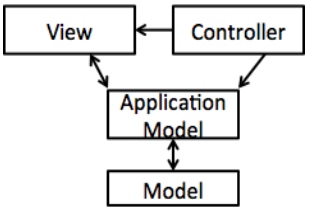

図 3 ApplicationModel-Model-View-Controller(AM-MVC)

同士で階層間の協調を実現する．PAC は，その構成単位を特定の機能を実現する Agent としている．Agent は，画面出力とユーザ入力処理を扱う Presentation，Agent の機能とデータを扱う Model，これらの間の協調と階層間の協調を実現する Control によって構成される．

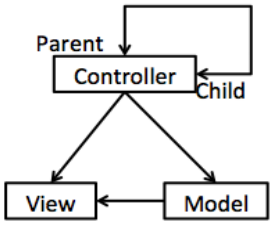

図 4 Hierarchical-Model-View-Controller(HMVC)

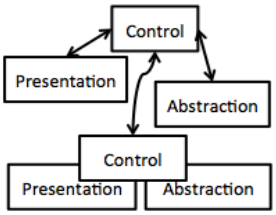

図 5 Presentation-Abstraction-Control(PAC)

MVP(図 6) は，リアクティブシステムに特化した派生である．画面に表示される視覚的要素とユーザ操作によるイベントを扱う View，プレゼンテーションロジックを扱う Presenter，ドメインモデルを扱う Model によって構成される．MVP の View はユーザ操作によるイベント処理と画面出力を行なうことから，MVC の View と Controller に対応する．MVP の Presenter は，MVC の View と Controller に横断する．Presenter と Model，View と Model 間はすべてイベント通知で協調する．これら要素間の関連も，MVC と同様に Classic タイプと Passive タイプがある．

MVVM(図 7) は，プレゼンテーションロジックとビジネスロジックの分離を目的とした，MVC とは異なる分割によるアーキテクチャである．画面に表示される視覚的要素とユーザ操作によるイベントを扱う View，ドメインモデルを扱う Model，View と Model を関連づけ外部表現とドメインモデルを結合した View-Model によって構成される．Abstraction は，特定の機能に関連する MVC の Model に対応する．

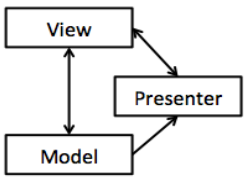

図 6 Model-View-Presenter (MVP)

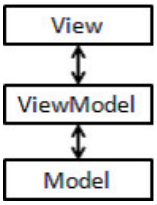

図 7 Model-View-ViewModel (MVVM)

3.2 共通参照アーキテクチャの設計

上述の調査結果に基づき，MVC アーキテクチャとその派生から以下の横断的関心事を識別した．

- MVC コンサーン
- UI(ユーザインタフェース) コンサーン
- 表示ロジックコンサーン
- 外部内部 View コンサーン
- 階層化コンサーン
- 通信コンサーン

UI コンサーンは，MVP，PAC，MVVM にみられるように，MVC における Controller と View を UI コンポーネントとして定義する．表示ロジックコンサーンは，AM-MVC にみられるように，プレゼンテーションロジックを分離する．外部内部 View コンサーンは，MVVM にみられるように，画面の外部表現とドメインモデルの中間表現である画面の内部表現を分離する．階層化コンサーンは，HMVC，PAC にみられるように，システムを階層的に捉えて分割する．通信コンサーンは，ClassicMVC，PassiveMVC，MVP にみられるように，要素間の通信の方向性についてを決定する．

以上の横断的関心事を二つの次元に分類することで，各次元の組み合わせで，MVC アーキテクチャおよびその派生のそれぞれを記述できる．

次元 1：MVC コンサーン，UI コンサーン
次元 2：表示ロジックコンサーン，外部内部 View コンサーン，階層化コンサーン

例えば，AM-MVC は MVC に対して表示ロジックコンサーンを分離していることから，MVC コンサーンのビューと表示ロジックコンサーンのビューの組み合わせによって，AM-MVC が記述される．

識別した横断的関心事を考慮し，アスペクト指向アーキテクチャとして共通参照アーキテクチャを設計する．共通参照アーキテクチャの MVC コンサーンビューを図 8(a) に示す．Model，View，Controller それぞれをアスペクトとし，IAD により協調する，UI コンサーンに着目した共通参照アーキテクチャのビューを図 8(b) に示す．MVC の View と Controller を結合した UI コンポーネントアスペクトと Model アスペクトによって構成される．図 9 以降の <<any>>は，この MVC もしくは UI ビューのいずれかに織込まれることを示す．

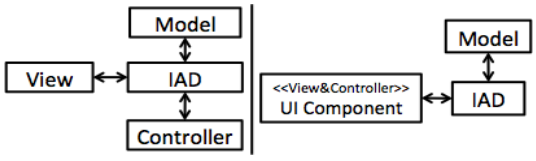

図 8 (a)MVC コンサーンビュー，(b)UI コンサーンビュー

図 9 は，表示ロジックコンサーンに着目した共通参照アー

キテクチャのビューを示す．MVC コンサーンによって規定される分割から，画面構築を実現する PresentationLogic アスペクトを分離している．

図 10 は，外部内部 View コンサーンに着目した共通参照アーキテクチャのビューを示す．MVC コンサーンによって規定される分割から，画面の内部表現を扱う ViewModel アスペクトを分離している．

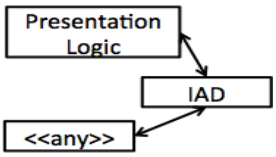

図 9 表示ロジック コンサーンビュー

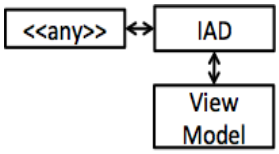

図 10 外部内部 View コンサーンビュー

図 11 は，階層化コンサーンに着目した共通参照アーキテクチャのビューを示す．階層毎に分割し，IAD により階層間の協調を実現する．

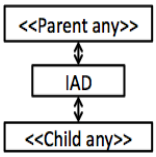

図 11 階層化コンサーンビュー

通信コンサーンは，アスペクト間記述 (以下，IAD) 内に実現することが自然と考えた．IAD はプッシュ通信とプル通信の切り替えと，メッセージの経路の決定を行なう．

3.3 共通アプリケーションアーキテクチャの設計

図 1 に示した通り，参照アーキテクチャの詳細化がアプリケーションアーキテクチャであると定義した．共通アプリケーションアーキテクチャの設計では，共通参照アーキテクチャによって定義されたアスペクト群に対してさらに横断する関心事を識別し，これをアスペクトとして分離する，識別した横断的関心事を表 1 に示す．

Model，View，Controller による通常のモジュール分割に対する横断的関心事として，ビジネスロジックコンサーン，データアクセスコンサーン，イベント管理コンサーンを識別する．Model に対して，特定の機能に関するビジネスロジックコンサーンが横断していることから，BusinessLogic アスペクトとして分離することで，新たな機能追加や修正を可能とする．また，DB に依存するデータアクセス処理に関するデータアクセスコンサーンが横断していることから，DataAccess アスペクトとして分離することで，DB の変更を独立して行なえるようにする．Controller に対して，イベント管理コンサーンが横断していることから，EventListener アスペクトと EventHandler アスペクト

表 1 共通参照アーキテクチャのビュー識別した横断的関心事

共通参照アーキテクチャのビュー	識別した横断的関心事
MVC コンサーンビュー (UI コンサーンビュー)	ビジネスロジックコンサーン
	データアクセスコンサーン
	イベント管理コンサーン
表示ロジックコンサーンビュー	画面構築コンサーン
	画面遷移コンサーン
内部外部 View コンサーン	表示コンサーン

に分割することで，外部イベントと内部イベントの組とその通知先を柔軟に変更できるようにする．

表示ロジックコンサーンによって規定される PresentationLogic アスペクトに対する横断的関心事として，画面構築コンサーンと画面遷移コンサーンを識別する．PresentationLogic に対して，画面構築の手続きに関連する画面構築コンサーンが横断していることから，ViewConstructor アスペクトとして分離することで，画面部品を再利用可能にする．さらに，画面遷移の管理に関連する画面遷移コンサーンが横断していることから，ViewTransition アスペクトとして分離することで，画面毎の再利用を可能とする．

外部内部 View コンサーンによって規定される ViewModel アスペクトに対する横断的関心事として，表示コンサーンを識別する．内容と役割と見栄え関連する ViewContent，DIContent，Style アスペクトとして分離することで，色やサイズなどの見栄えや，表やリストなどその表現方法を内容から独立して変更可能となる．

共通参照アーキテクチャと同様に，それぞれのコンサーンに着目した各ビューを共通アプリケーションアーキテクチャとして記述する．本稿では，各々のビューについては，ページ数の制限の都合上省略する．図 12 は MVC ビューの例であり，MVC コンサーンのみ指定し，その他のについては Model，View，Controller それぞれに織込まれる．

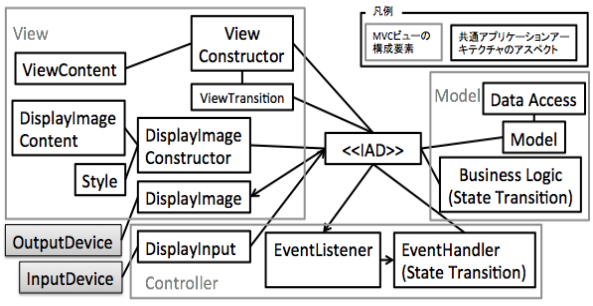

図 12 共通アプリケーションアーキテクチャ

4. 共通アーキテクチャと既存の参照アーキテクチャおよび既存の環境との関係

共通参照アーキテクチャと MVC およびその派生，共通アプリケーションアーキテクチャと既存の環境の前提とする参照アーキテクチャおよびアプリケーションアーキテクチャそれぞれについて比較し，その関係を議論する．

4.1 既存の参照アーキテクチャとの関係

前述のとおり，共通アーキテクチャは，特定の横断的関心事のいくつかを指定し，織込むことで，MVC とその派生が生成される．4.2 節で説明する既存の環境が前提とする参照アーキテクチャである PassiveMVC，MVVM と共通参照アーキテクチャの関係を例として挙げる．

PassiveMVC は，MVC コンサーンによって規定される分割を前提として Controller を介して Model と View は協調する．したがって，MVC コンサーンのみ指定し，その他のコンサーンについては Model，View，Controller それぞれに織込まれる．結果として図 8(b) と同じ記述になる．

MVVM は，UI コンサーンによって規定される分割に対して横断する外部内部 View コンサーンを分離している．したがって，UI コンサーンと外部内部 View コンサーンを指定し，その他のコンサーンについては UI Component，Model，ViewModel に織込まれる．図 13 は，図 8(b) と図 10 の組み合わせによって，MVVM アーキテクチャが記述されることを示している．

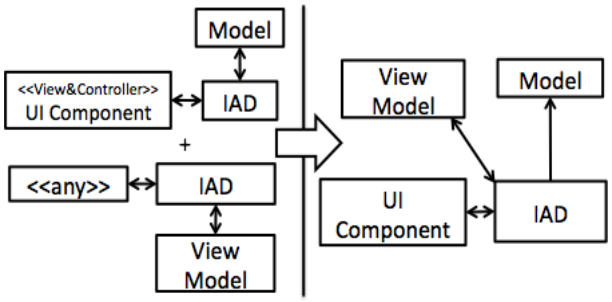

図 13 UI コンサーンビューと外部内部表現 View ビューの結合によって記述される MVVM アーキテクチャ

4.2 既存の環境が前提とするアプリケーションアーキテクチャおよび参照アーキテクチャとの関係

インタラクティブシステムを分類した場合，ネイティブアプリケーションと Web アプリケーションに分類され，別の観点では開発を支援するアプリケーションフレームワークを用いて実現されたアプリケーションとクロスプラットフォーム開発環境を用いて実現されたアプリケーションに分類される．共通アーキテクチャと環境との関係の例として，典型的に対称なものを挙げる．Web アプリケーションの環境の例として Ruby on Rails，ネイティブアプリケーションの環境の例として PhoneGap を挙げる．

4.2.1 Ruby on Rails との関係

参照アーキテクチャ間の関係

Ruby on Rails は PassiveMVC を前提としている．前述の通り，図 2(b) は Ruby on Rails の前提とする参照アーキテクチャである．Controller は，HTTP コマンドの外部イベントを Ruby で定義された内部イベントに変換し，この内部イベントに応じて Ruby で定義された Model へメッセージを送信または，View に対してレンダリングするテンプレートを指定する．Model ではアクション完了後，Controller を経由して View にメッセージを送信する．View は指定されたテンプレートから画面を構築し，これを出力する．共通アーキテクチャは，前節で説明した通り，PassiveMVC コンサーンのみ選択し，それ以外のコンサーンについては，Model，View，Controller それぞれに織込まれることで，Ruby on Rails の前提としている参照アーキテクチャと同じ図 2(b) が生成される．したがって，共通参照アーキテクチャは Ruby on Rails の参照アーキテクチャを説明可能である．

アプリケーションアーキテクチャ間の関係

図 14 は，Ruby on Rails の前提とするアプリケーションアーキテクチャである．そのアプリケーションアーキテクチャは PassiveMVC に横断する画面構築コンサーン，画面遷移コンサーン，データアクセスコンサーン，イベント管理コンサーンを分離している．ERB Engine は Template を用いて画面を構築することから，画面構築コンサーンによって規定された要素である．これによって画面部品の再利用が容易となる．ActionView は通知されるイベントに応じて利用する Template を決定することにより画面遷移を実現しているので，画面遷移コンサーンによって規定される要素である．これによって画面単位での再利用が容易になる．ActiveRecord は DB アクセスに関連する手続きを実現することから，データアクセスコンサーンによって規定された要素である．これによって，DB の変更を独立して可能になる．ActionDispatch は，ブラウザからの外部イベントを内部イベントに変換し，ActionController は，イベントに応じて BusinessLogic または ActionView へのルーティングを実現することから，イベント管理コンサーンによって規定された要素である．これによって，外部イベントに対応する内部イベントの組とその通知先を柔軟に変更可能になる．

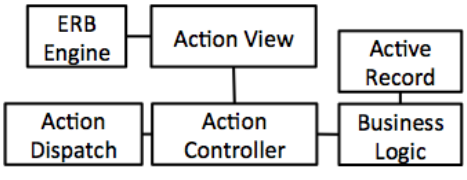

図 14 Ruby on Rails の前提とするアプリケーションアーキテクチャ

共通アプリケーションアーキテクチャと Ruby on Rails のアプリケーションアーキテクチャとの対応関係について考察する．Ruby on Rails は，PassiveMVC に横断する画面構築コンサーン，画面遷移コンサーン，データアクセスコンサーン，イベント管理コンサーンを分離した構造を持つことから，共通アプリケーションアーキテクチャのこれら

のコンサーンによって規定されるアスペクト群と対応する．これらのコンサーンを指定した共通アプリケーションアーキテクチャの構造は，3.3 節で述べたように，画面構築コンサーンによって規定される ViewConstructor アスペクト，画面遷移コンサーンによって規定される ViewTransition アスペクト，データアクセスコンサーンによって規定される DataAccess アスペクト，イベント管理コンサーンによって規定される EventListener アスペクト，EventHandler アスペクトから構成される．それ以外のコンサーンについては，これらのアスペクトに織込まれる．図 15 に，この生成されたビューのアスペクトと Ruby on Rails の構成要素との対応関係を示す．Ruby on Rails ビューを生成できたことから，共通アプリケーションアーキテクチャは，Ruby on Rails のアプリケーションアーキテクチャを説明可能である．

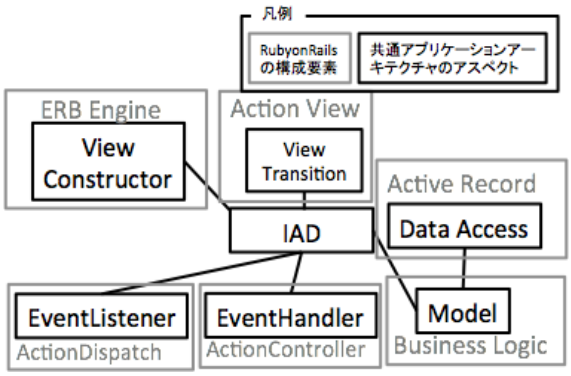

図 15 共通アプリケーションアーキテクチャと Ruby on Rails のアプリケーションアーキテクチャの関係

4.2.2 PhoneGap との関係

参照アーキテクチャ間の関係

既存のネイティブアプリケーションの開発環境として PhoneGap を例に挙げる．PhoneGap は，HTML と JavaScript などの Web 標準を用いたネイティブアプリケーションのクロスプラットフォーム環境である．

PhoneGap は参照アーキテクチャとして MVVM を前提としている．図 16 は PhoneGap の前提とする参照アーキテクチャである．View and EventHandler は，MVVM の View に対応し，画面出力とユーザ操作によるイベント処理を HTML と JavaScript で実現する．ViewModel は，MVVM の ViewModel に対応し，View の内部表現とその操作を DOM 木と JavaScript による操作で実現する．Business Logic は，MVVM の Model に対応し，ドメインモデルを JavaScript で実現する．図 16 は，この PhoneGap の前提とする参照アーキテクチャと MVVM の関係を示す．

共通参照アーキテクチャと PhoneGap の参照アーキテクチャとの対応関係について考察する．PhoneGap は MVVM に基づくことから，共通アーキテクチャ上の UI コンサーン

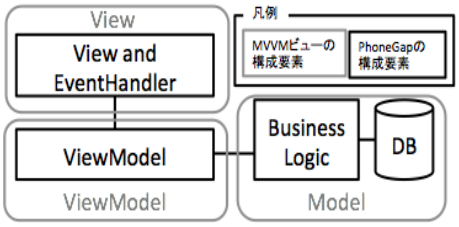

図 16 PhoneGap の前提とする参照アーキテクチャ

と外部内部 View コンサーンによって規定されるアスペクト群と対応する．この横断的関心事を指定することで共通アプリケーションアーキテクチャのビューは図 13 になる．図 17 に，このビューに記述されるアスペクトと PhoneGap の構成要素との対応関係を示す．PhoneGap ビューを生成できたことから，共通参照アーキテクチャは PhoneGap の参照アーキテクチャを説明可能である．

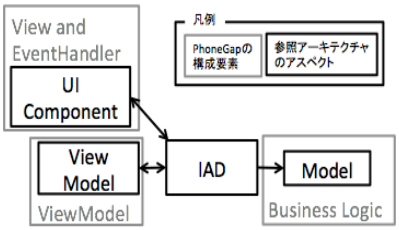

図 17 共通参照アーキテクチャと PhoneGap の参照アーキテクチャの関係

アプリケーションアーキテクチャ間の関係

PhoneGap のアプリケーションアーキテクチャと共通アプリケーションアーキテクチャの関係について考察する．PhoneGap は図 17 を詳細化し，これに横断するイベント管理コンサーンと画面構築コンサーンを分離していると考えた．HTML と JavaScript によってアプリケーションを構築するさいに，EventHandler や DOM 木の操作はそれぞれ JavaScript によってオブジェクトとして定義される．図 18 に，PhoneGap のアプリケーションアーキテクチャを示す．

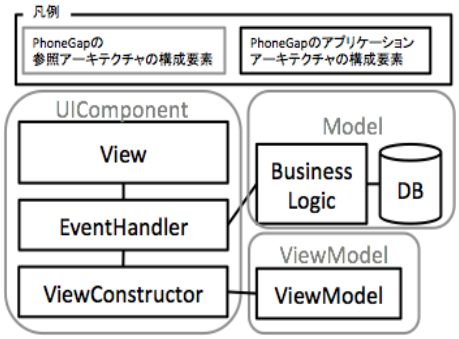

図 18 PhoneGap の前提とするアプリケーションアーキテクチャ

PhoneGap は，イベント管理コンサーンと画面構築コンサーンを分離した構造を持つことから，共通アプリケー

ションアーキテクチャのこれらのコンサーンによって規定されるアスペクト群と対応付く．これらのコンサーンを指定したアプリケーションアーキテクチャは，イベント管理コンサーンによって規定される EventHandler アスペクト，画面構築コンサーンによって規定される ViewConstructor アスペクト，View に対応する表示コンサーンによって規定される ViewContent，DIContent，Style アスペクトから構成される．図 19 に，この共通アプリケーションアーキテクチャのビューに記述されるアスペクトと PhoneGap の構成要素との対応関係を示す．PhoneGap のビューを生成できたことから，共通アプリケーションアーキテクチャは，PhoneGap のアプリケーションを説明可能である．

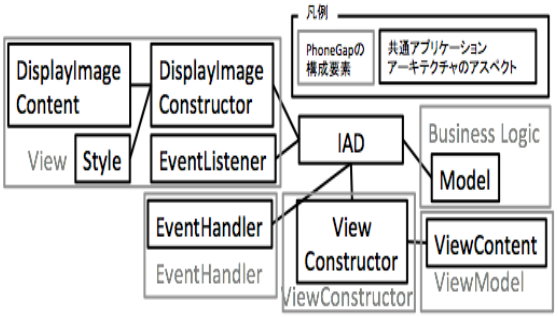

図 19 共通アプリケーションアーキテクチャと PhoneGap のアプリケーションアーキテクチャの関係

5. 考察

図 1 に示すように，共通アーキテクチャと既存の参照アーキテクチャおよび既存の環境の前提とするアプリケーションアーキテクチャが対応付いた．この関係を用いて異なる環境の実現技術間の関係が明確になると考える．したがって，ある開発環境を用いた開発に習熟した技術者が異なる開発環境に移行するさいに，共通アーキテクチャを介して，今まで利用していた開発技術に対応する移行先の開発環境で用いられる技術を理解したり，移行先の開発環境で特定の構成要素を実現することが，移行前の開発環境を用いた開発と照らし合わせて，何を意味するのか理解することができる．例えば，Ruby on Rails について詳細にわたり把握している開発者が，開発環境として PhoneGap を用いて開発をするさいに，ActionController や BusinessLogic，ActiveRecord で実現していた内容を，JavaScript で EventHandler や BusinessLogic を実現するオブジェクトとして実現すべきことがわかる．また，HTML 内で記述されている内容が，ActionDispatch や ActionView で実現している内容と同じであることがわかる．

本稿では共通参照アーキテクチャを介して，既存の参照アーキテクチャ間の関係，共通アプリケーションアーキテクチャを介して既存のアプリケーションアーキテクチャ間の関係が明らかにした．今後，環境毎にアーキテクチャの構成要素に対して用いられる実装技術を対応付けることにより，開発環境で用いられる技術間の関係，実行時環境で用いられている技術間の関係が明らかとなり，上述のような技術者の環境移行に対する支援が可能になると考える．

6. おわりに

既存の開発環境間の差異がソフトウェアの生産性向上の妨げになることを問題とし，これはアーキテクチャの差異に起因すると考えた．多岐にわたる環境のアーキテクチャそれぞれについて説明可能な共通アーキテクチャを設計し，関係を整理した．さらに，任意の実行時環境で稼働するアプリケーションの任意の開発環境を用いた作成支援の可能性を考察した．

謝辞 本研究の一部は，JSPS 科研費 (基盤研究 (C)24500049)，および 2014 年度南山大学パッヘ奨励金 I-A-2 の助成による．ここに謝意を表する．

参考文献

[1] Allen, S., Graupera, V. and Lundrigan, L.: *Pro smartphone cross-platform development: iPhone, blackberry, windows mobile and android development and distribution*, Apress (2010).
[2] Bass, L.: *Software architecture in practice*, Addison Wesley (2007).
[3] Bos, B., Celik, T., Hickson, I. and et al: Cascading Style Sheets, W3C (online), available from ⟨http://www.w3.org/TR/CSS2/⟩ (accessed 2015-02-16).
[4] Clements, P., Bachmann, F., Bass, L. and et al.: *Documenting Software Architectures Views and Beyond Second Edition*, Addison Wesley (2010).
[5] Hickson, L., Berjon, R., Faulkner, S. and et al: HTML, WHATWG, W3C (online), available from ⟨http://www.w3.org/TR/html5/⟩ (accessed 2015-02-16).
[6] Krasner, G. E. and Pope, S. T.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system, *object oriented programming*, Vol. 1, No. 3, pp. 26–49 (1988).
[7] Sokolova, K., Lemercier, M. and Garcia, L.: Towards High Quality Mobile Applications: Android Passive MVC Architecture, *Advances in Software*, Vol. 7, No. 2, pp. 123–138 (2014).
[8] Vliet, H. V.: *Software engineering: principles and practice.*, Wiley (2007).