スマートデバイスアプリケーションの自動生成に関する研究
―画面イメージの生成を題材として―

12se003 天野智裕　12se165 森崇行　12se204 長田康宜
指導教員：野呂昌満

1　はじめに

近年，様々なスマートデバイスが広く普及し画面サイズも多様化している．レスポンシブ Web デザイン [1] は異なる画面サイズに応じて，適切なビューを構築する技術である．

レスポンシブ Web デザインは想定されるデバイスの差異に合わせて，複数のページを予め定義する必要がある．普及している多様な画面サイズ全てに対して，それぞれに最適な画面を定義することは困難である．一つのビューを定義し，それを動的に変換することが出来れば複数のビューを定義する必要がなくなる．

本研究室で提案されているインタラクティブソフトウェアの共通アーキテクチャ [3] ではビューのオブジェクトモデル (以下，ViewModel) からビューを生成している．ViewModel は ViewContent, DisplayImageContent, Style から構成されている．ViewContent は表示内容，DisplayImageContent は役割，Style は見栄えを表している．

本研究の目的は，ViewModel の変換のパターン化である．すなわち，画面サイズやユーザ操作に応じてビューの構造の動的変換を実現する．これは ViewModel の特定の型から型への変換であると考え，具体的な事例から共通部分を抜き出し，変換規則をパターンとして定義する．パターンと ViewModel の組み合わせから，変換後のビューを動的に生成することができる．これにより，画面サイズ毎にページを静的に定義する必要が無くなり，開発の省力化に繋がる．

ビューの動的生成実現のために，ViewModel の型の変換をパターン化する．ビューの変換における横断的関心事を特定し分離する．関心事の分離にはデザインパターンを用いる．具体的な変換パターンの実現には各デザインパターンのコンポーネントの組み合わせを用いる．共通アーキテクチャは，任意の開発環境で開発したソフトウェアを，他の開発環境に対応する実行時環境で稼働できることを目標としている．共通アーキテクチャでは View の ViewConstructer が ViewModel の型とインスタンスを生成している．ビューの動的変換をするさいに，ViewConstructer で ViewModel の型を操作履歴に基づいて変換する．

本研究ではビューの具体例を用いて変換前のビューから変換後のビューを，操作履歴に基づいて変換可能であることを，具体的に設計し事例を用いて検証する．

2　背景技術

2.1　レスポンシブ Web デザイン

レスポンシブ Web デザインとは, 特定の実行時環境を想定してつくられる Web ページを，画面サイズに応じて適切に表示させるための技術のことである．メディアクエリ，フルードグリッド，フルードイメージの 3 つの技術的要素がある．レスポンシブ Web デザインは CSS の技術であり，多様な画面サイズのスマートデバイスに対して, 合わせたレイアウトを 1 つの HTML ソースで対応させる．メディアクエリはデバイスの画面サイズに対して任意にブレイクポイントを定め，適用する CSS ファイルを変更することである．フルードグリッドは画面のレイアウトをピクセルではなく割合によって定める事である．フルードイメージは画面内の画像などの固定のサイズを持つ要素に対して縦横の比率を維持しながら自動的にサイズを拡大または縮小して画面サイズに合わせて表示することである．

2.2　共通アーキテクチャ

共通アーキテクチャでは任意の実行時環境で稼働するアプリケーションを任意の開発環境を用いて作成する枠組みを築いている．共通アーキテクチャは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からなっている．共通参照アーキテクチャでは開発環境を，共通アプリケーションアーキテクチャでは実行時環境を説明可能としている．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では，複数の既存のアーキテクチャ [2] をすべて説明可能としている．あるアーキテクチャを説明可能とは，アスペクト指向アーキテクチャとして定義し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折り込み，そのアーキテクチャを生成できることを指す．

共通参照アーキテクチャを詳細化したアプリケーションアーキテクチャの例は図 1 である．共通アプリケーションアーキテクチャの View 部分のコンポーネントとして ViewContent, DisplayImageContent, Style がある．ViewContent は文字列や画像などの表示内容を表す．DisplayImageContent はボタンやセレクトボックスなどの役割を表す．Style は色やフォントなど見栄えに関する部分を表している．ViewConstructor では，この 3 つから構成されている ViewModel を生成している．DisplayImageConstructor では，ViewConstructor で生成した ViewModel を入力として DisplayImage を生成している．

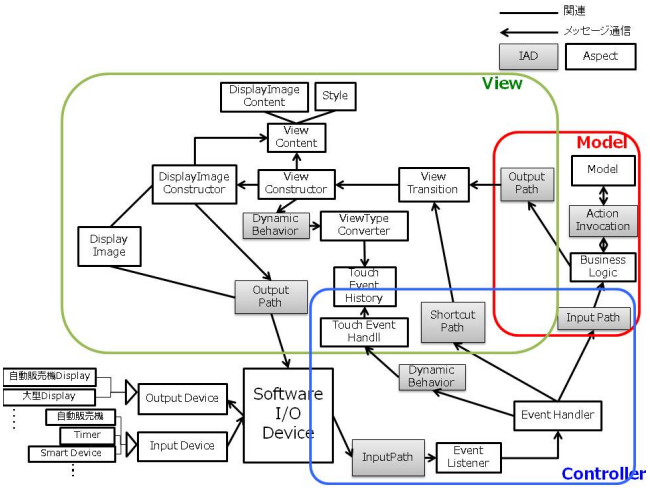

図 1　共通アーキテクチャ

3　設計

3.1　ビュー変換の概要

レスポンシブ Web デザインでは画面を表示するために，使われるデバイスを複数想定してあらかじめ画面を想定に合わせて複数定義する必要がある．多様な画面サイズに対してその全てにとって最適な画面を定義することは困難である．そこで画面をコンテキストによって動的に変化させることであらかじめ定義する画面を一つにすることができる．コンテキストをユーザに関するものにすることで，デバイスに適した画面ではなくユーザに適した画面を生成できる．網羅的に全てのビューを定義することなくビューを動的に変換するために，コンポーネントの変換をパターンとして定義する．ViewModel は木構造で定義されているので，ViewModel の型は構造型と要素型で表すことが出来る．図 2 では操作履歴を参照し，その状態に応じて ViewModel の型を変換する．変換後の ViewModel は Model を取得し変換後のインスタンスを生成する．

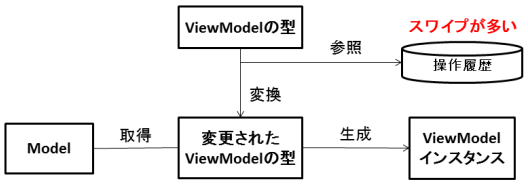

図 2　ViewModel 生成

たとえばビューを画面サイズによって変換するさい，画面の内容が違っても同じ手続きで ViewModel を変換することがある．その手続きをパターンとして定義する．

同じ手続きの例として二次元の表を一次元の表で表示したい場合を考える (図 3)．二次元表，一次元表をそれぞれ構造型とし，それぞれの行数，列数，表題，見出し要素を要素型とする．アーティストレコードという内容と二次元表の ViewModel の型から，アーティストレコード二次元表ページが生成される．その二次元表を一次元表として表示したい場合，二次元表の型を一次元表の型に変換することで，アーティストレコード一次元表ページが生成される．これと同様の変換をムービーランキングページでも行なわれている．複数のページで行なわれている二次元表から一次元表への変換部分をパターンとして定義する．これにより，二次元表の要素がどのようなものであってもパターンを適用させることで一次元表が生成される．すなわちパターン化することで変換可能な内容について適用したい複数の型をあらかじめ定義する必要がなくなる．

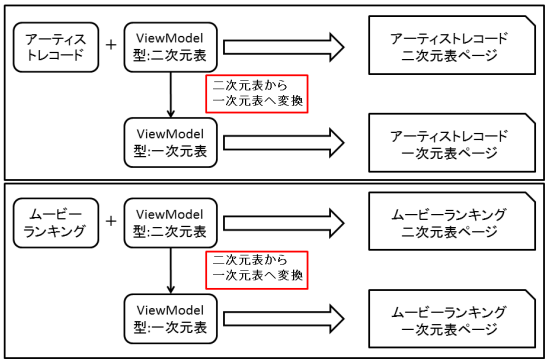

図 3　ViewModel の変換：具体例

3.2　ViewConstructer の設計

ViewConstructor では ViewModel の型の変換と ViewModel のインスタンス生成を行なう (図 4)．型変換では最初に変換前の型である二次元表のインスタンスを生成します．次に操作履歴にあたる TouchEventHistory を参照し，履歴の状態に応じて ViewTypeConverter で型の変換を行ない，一次元表を生成する．ViewModel のインスタンス生成では，Model を取得し ViewModel の型との対応関係から一次元表のインスタンスを生成する．

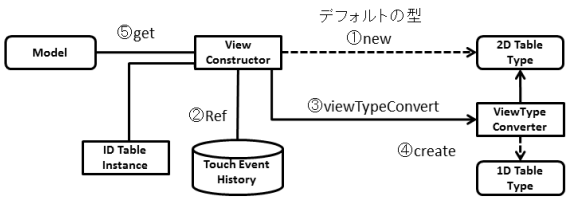

図 4　ViewConstructor のふるまい

変更手続きの例として，二次元の表から一次元の表への変換を取り扱う．ビューを変換するさいは，変換前の ViewModel のインスタンスを走査し，生成されている変換後の ViewModel のインスタンスを生成する．最初に変換前の表題を読み込んだときは，変換後の表に表題を生成する．次に変換前の表から見出しの行を読み込んだときは，変換後の表に見出しの数だけ表要素を生成する．最後

に変換前の表から行を読み込んだときは，変換後の表に行に対する列要素を生成する．

3.2.1 変換のためのプログラムパターン

ViewModel の型の変換は以下の 2 工程により実現される．

- 構造変換
- 要素変換

構造変換とは ViewModel の型の構造を変換することである．要素変換とは ViewModel の要素群を異なる要素群に変換することである．

二次元表から一次元表への変換の処理に PrototypePattern を用いる (図 5)．型変換処理のさいに ViewModelType レポジトリから変換後の ViewModel の型を取得する．取得する型は総称型であるので ViewModel の生成に必要なパラメータを与える．これに表題や行数，見出し要素を与える事で ViewModel を変換することができる．

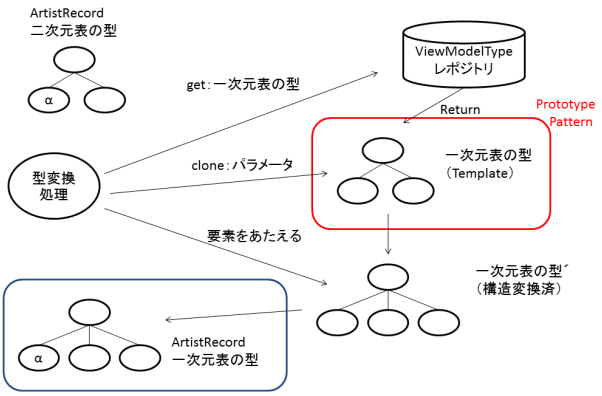

図 5 PrototypePattern を用いた一次元表の生成工程

構造の変換では TouchEventHistory を参照し，操作履歴を取得する．変換前の型と操作履歴に応じて，FactoryMethodPattern を用いて変換後の型を取得し ViewModel の構造を生成する (図 6)．要素の変換では ViewModel の型を走査しながら，変換前の型の要素を変換後の型の要素に変換する．ViewModel の走査手順は ViewModel の要素に InterpreterPattern を用いる．各要素の変更手続きは IteratorPattern を用いる．

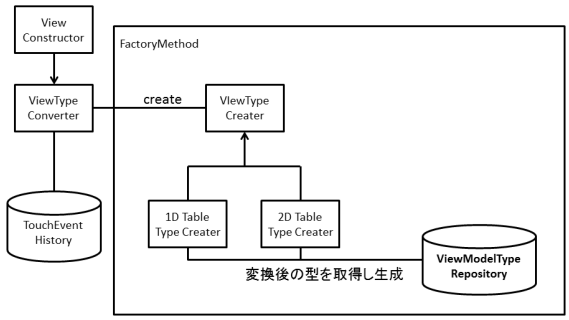

図 6 ViewModel の構造の変換

ここでは，変換のためのプログラムパターン (以下，変換パターン) の例として二次元表から一次元表への変換パターンを説明する (図 7)．

二次元表から一次元表の要素の変換には，変換前の見出し行と行を繰り返し走査し，変換後の要素に変換する必要がある．したがってこの実現に IteretorPattern を用いる．変換前の型のインスタンスから変換後の型のインスタンスを生成する FactoryMethodPattern と，表題や行数，列数を規定する見出し要素を変換後の要素に変換する IteretorPattern がある．これらが二次元表から一次元表への変換パターンである．どのような二次元表であっても，この FactoryMethodPattern と IteretorPattern に入力として与えることで一次元表を生成できると考えている．

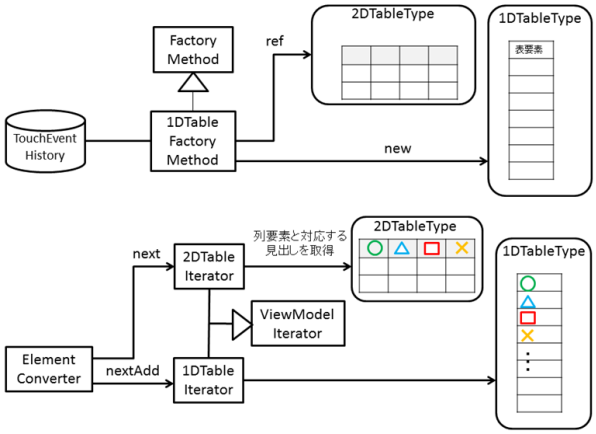

図 7 二次元表から一次元表への変換パターン

3.2.2 ViewModel のインスタンス生成

ViewModel のインスタンスを生成するために次の 3 ステップを行なう (図 8)．

- ViewModel の型を走査
- Model 要素を取得
- 対応関係から ViewModel のインスタンスを生成

ViewModel の型走査では，InterpreterPattern を用いることで Tree や List 等，構造毎に走査手順を定義する．Model 要素の取得では IteratorPattern を用いることで，Model の構造によらず順にアクセスし，一つずつ要素を取得する．ViewModel のインスタンス生成では VisiterPattern を用いることで，Model と ViewModel の型の対応関係から各ノードのインスタンスを生成していく．

4 事例検証

本章では第 3 章での設計を基に変換前のビューから変換後のビューを操作履歴に基づいて生成できることについて検証する．変換前のビューを二次元表とし，変換後のビューを一次元表，ユーザの操作履歴は横方向のスワイプ操作とする．表はアーティストレコードページというもの

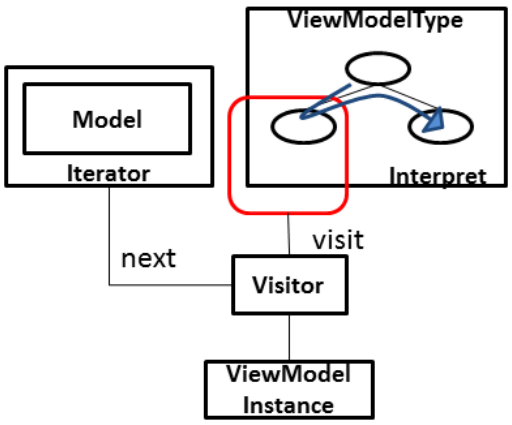

図 8　ViewModel のインスタンス生成

で検証を進めていく．図 9 の変換前のビューである二次元表は表題，見出し行，行の要素から成り立っており，これらの情報を読み取ることで図 9 の変換後のビューである一次元表が生成されると考える．

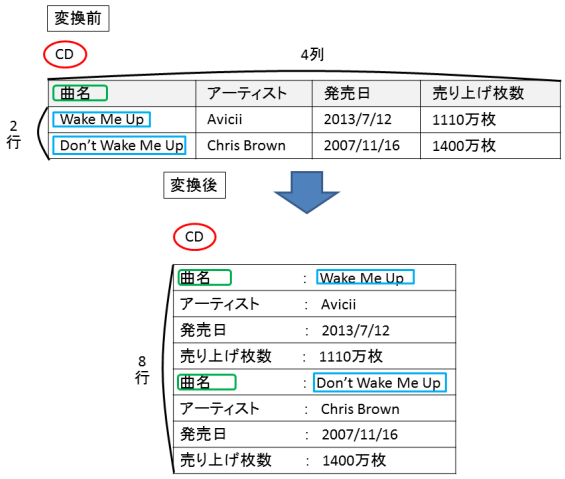

図 9　変換前と変換後の表の対応関係

見出し行の列要素は項目名であり，データ行の列要素はそれに対応する要素である．またデータ行はレコードを表す．一次元表は，二次元表の見出し行の列要素とデータ行の対応する列要素の組を列とした表である．二次元表を一次元表に変換するさい，二次元表のデータ行数×見出し要素数の行をもつ一次元表を作成する．二次元表の各データ行について，列要素に対応する見出し要素を一次元表の見出し要素として設定する．ViewModel の変換処理の具体的な設計は図 10 と図 11 に示す．

5　考察

本研究では，ビューの動的変換を実現するために，ViewModel の型の変換をパターン化した．またパターン化するために，ビューの変化を定義する型における横断的関心事を特定し，デザインパターンを用いて分離した．具体的な変換パターンの実現は各デザインパターンのコンポーネントの組み合わせを用いた．二次元表と一次元表というビューの具体例を用いて，変換前の ViewModel の型から変換後の ViewModel の型を操作履歴に基づいて変換可能であることを具体的に設計し検証した．これにより変換後のビューが生成可能であること，またコンテキストからビューの変換が可能であることが分かった．

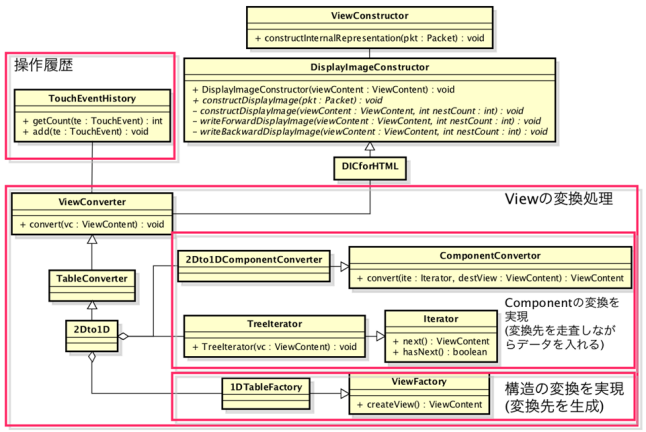

図 10　クラス図

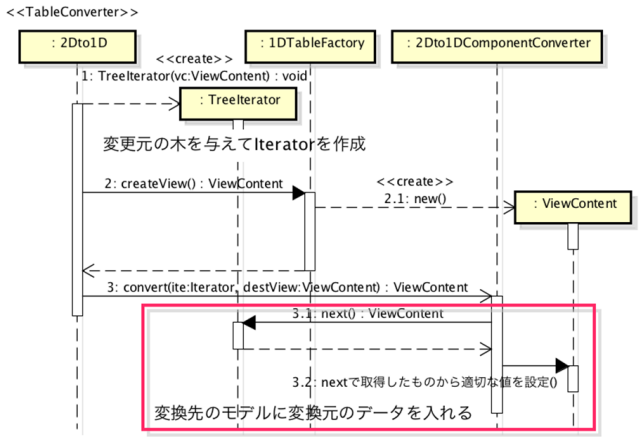

図 11　シーケンス図

今後の課題としていくつかの例において，変換パターンを定義し，その変換を共通部分を抜き出すことで標準変換パターンの定義を目指す．

参考文献

[1] E. Harb, P. Kapellari, S. Luong, and N. Spot, “Responsive Web Design,” Dec. 2011

[2] K. Sokolova, M. Lemercier, and L. Garcia, “TowardsHigh Quality Mobile Applications: Android PassiveMVC Architecture,” International Journal On Advancesin Software, vol. 7, no. 2, pp. 123-138, 2014.

[3] 江坂篤侍，野呂昌満，沢田篤史，“ 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，” 情報処理学会研究報告. ソフトウェア工学報告，vol.2015-SE-187, no.32，pp.1-8，May. 2015.