スマートデバイスアプリケーションのアーキテクチャに関する研究
複数プログラミング言語への対応

2012SE039 林 史也　2012SE054 池田 昌史　2012SE152 三矢 大貴
指導教員：野呂 昌満

1　はじめに

スマートデバイスの普及に伴って，それらの実行時環境や開発環境は多岐にわたる．開発用のプログラミング言語を例にとっても，HTML，Java，Swift など，様々なものが利用されている．一人の技術者がこれらすべての言語の詳細を把握するのは困難であり，そのような技術者を多数確保することは難しい．本研究ではその問題を自動生成という形で解決する．

本研究の目的は，モデル駆動型アーキテクチャ (以下 MDA) に基づいて，HTML，Java，Swift の View の自動生成を行うことである．本研究室で提案されている共通アーキテクチャ [1] に基づくアプリケーションフレームワークを前提とする。

モデル駆動型アーキテクチャに基づく自動生成を行なう．プラットフォームを言語とする．PIM として内部表現である View の ObjectModel を用いる．ViewObjectModel の要素として，DisplayImageContent が役割を，ViewContent が中身を，Style が見た目を持つ．これらは言語に依存しない形で定義されている．PSM として外部表現の DisplayImage を定義する．PIM から PSM の変換を行うために DisplayImageConstructor を用いる．PIM と特定のプログラミングでの実装 (PSM) との対応付けを行なうことで入力として必要となる情報を整理する．今回は DisplayImage を出力する DisplayImageConstructor を作成する．言語に依存しない形での DisplayImage を定義した ViewModel を，プラットフォームを言語とした PIM とする．ViewModel は木構造であり，その木を Interpreter-Pattern を用いて走査する．VisitorPattern を用いて読み取った要素に対応した外部表現を生成することで MDA に基づく自動生成を行なう．本研究では PSM による複数言語での DisplayImage としての外部表現の出力を目標とする．今回は HTML，Java，Swift の 3 つをプラットフォームとする．

2　背景技術

2.1　モデル駆動型アーキテクチャ

MDA とはモデルを利用したソフトウェア開発手法であり，プラットフォームに独立なモデル（PIM）を設計する．それをプラットフォームに依存したモデル（PSM）の生成を行なう．設計を実装から独立させることで，プロダクトの可搬性，相互運用性，可搬性の向上を狙う．[3]

2.2　共通アーキテクチャ

共通アーキテクチャとは，多くのプラットフォームが存在する現状において，開発支援を目的とされているアーキテクチャである，共通アーキテクチャ自身と様々な環境のアーキテクチャとの対応関係により，様々な環境のアーキテクチャの説明を可能とする．これによってアプリケーション間の変換が可能となる．

2.2.1　共通参照アーキテクチャ

共通参照アーキテクチャは MVC アーキテクチャとその派生が分離を試みている関心事を特定する．これをアスペクトとして分離し，アスペクト指向アーキテクチャとして定義されている．共通アプリケーションアーキテクチャは，共通参照アーキテクチャを詳細化し図 1 のように設計されている．

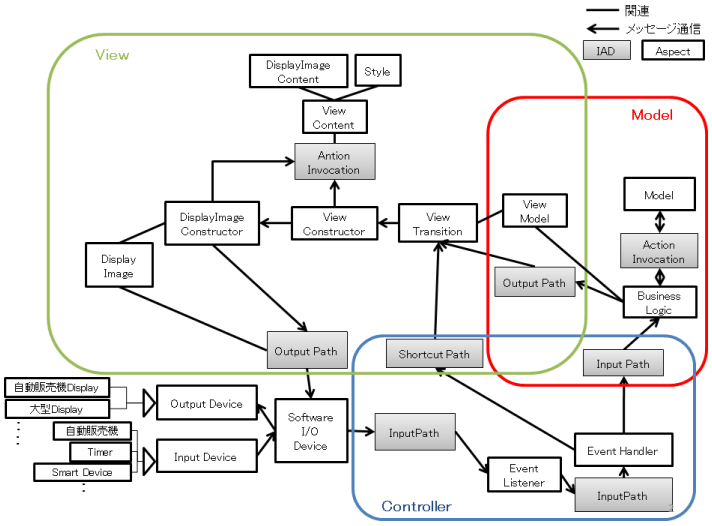

図 1　共通アプリケーションアーキテクチャ

2.2.2　ViewModel

プラットフォームに依存しない形での DisplayImage を定義している．これを入力とすることで，DisplayImage-Constractor によりそれぞれのプラットフォームに合わせた実装を出力することができる．

2.2.3　DisplayImageConstructor

DisplayImageConstructor(以下 DIC) とは、内部表現を入力することで外部表現を出力する。内部表現を構成する ViewContent，DisplayImageContent，Style から外部表現である DisplayImage の生成を行なう．DIC を特定の外

部表現毎に定義することができれば，DIC を選択することである内部表現から複数の外部表現の生成が可能となる．

3　設計

3.1　PSM

任意のプログラミング言語での実装．本研究では，HTML，Java，Swift の 3 つとする．Java の場合はレイアウト情報を記述した java ファイル，Swift の場合は ViewController.swift となる．

3.2　PIM

ViewModel．View の ObjectModel で，木構造である．View の中身 (ViewContent)，役割 (DisplayImageContent)，見た目 (Style) をまとめたものである．

3.3　DIC

DIC を Java，Swift で記述する場合，DIC をオブジェクトとして作成，ViewContent はオブジェクトの中身となる。ViewContent の対応を調べ，オブジェクトかどうか分類したとき，Style はすべてのオブジェクト内で対応可能である．Style はオブジェクトに付随する．

4　事例検証

ViewModel と DIC は，デザインパターンで生成できる．ViewModel を Visitor，DIC を Acceptor とした Visitor パターンと，ViewModel の内容の ViewContent を Iterator，複数の表示方法を Aggregate とした Iterator パターンの二つを使用することで可能となる．
Java+Swing　オブジェクト指向→ UML 記述
-要素に対応するライブラリの展開をそれぞれ記述
Swift+UIKit　オブジェクト指向→ UML 記述
-Import　UIKit
要素に対しての明確な指定先が無い，指定先はあるがその先がパッケージなのかメソッドなのかクラスなのかは判断することができない．そのため，呼び出されても，単体でもいいのか，他のクラスが必要になるのか，インポートが必要なのかは，別定義する必要がある．

4.1　ViewModel

今回考察した画面のサンプルは図 2 である．Java，Swift で記述した際に表示される画面とサンプルを比較する．図 3 が，その ViewModel となる．

4.2　ViewModel と各言語の対応

以下が ViewModel と各言語の対応表である．比較対象は HTML，Java，Swift である．

4.3　生成したサンプルコード

4.3.1　HTML で表現した外部表現

HTML を生成する場合，図 4 のような走査を行う．行き掛け順に走査し，立ち寄ったときに対応する要素の開始

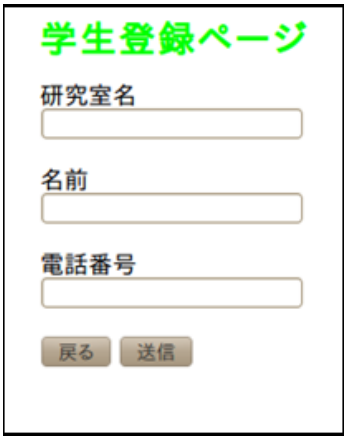

図 2　画面サンプル (html での実装)

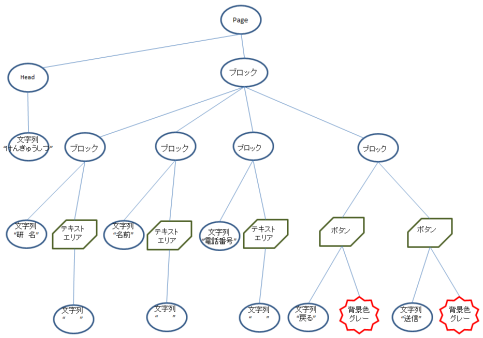

図 3　ViewModel

DIC	HTML	Java(Swing)	Swift
ボタン	Button	JButton	UIButton
TF	<input type=text>	JTextField	UITextField
画像	<img>	ImageIcon	UIImageView

表 1　DisplayImageContent の対応表

ViewContent	HTML	Java(Swing)	Swift
文字列	テキスト	テキスト	UITextView
ブロック		JPanel	UIView

表 2　ViewContent の対応表

タグを，そこから親に戻る際に終了タグを生成することで外部表現を生成する．

ソースコード 1　HTML(一部抜粋)

```
1　<div>
2　 <label>研究室名</label><br>
3　 <input type="text" name="field1">
4　</div><br>
5　<div>
6　 <label>名前</label><br>
7　 <input type="text" name="field2">
8　</div><br>
```

Style	HTML	Java(Swing)	Swift
フォントサイズ	font-size	setFont	UIFont
レイアウト	Potion	Layout	layer
色	color	color	UIColor

表 3　Style の対応表

4.3.2　Java(Swing) で表現した外部表現

実行結果は図 6 のとおりである．Java を生成する場合，図 5 のような走査を行う．行き掛け順に走査し，立ち寄ったときに対応する要素を生成，そこから親に戻る際に親の要素に add することで外部表現を生成する．

ソースコード 2　Java(一部抜粋)

```
1    JPanel panel2 = new JPanel();
2    JLabel label2 = new JLabel("研究室名");
3    panel2.add(label2);
4
5    JTextField field1 = new JTextField();
6    panel2.add(field1);
7    panel1.add(panel2);
8
9    JPanel panel3 = new JPanel();
10   JLabel label3 = new JLabel("名前");
11   panel3.add(label3);
12
13   JTextField field2 = new JTextField();
14   panel3.add(field2);
15   panel1.add(panel3);
```

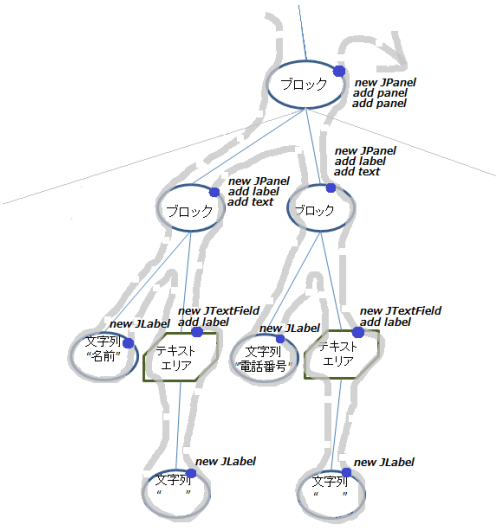

図 5　生成の走査-Java-

図 6　実行結果-Java(Swing)-

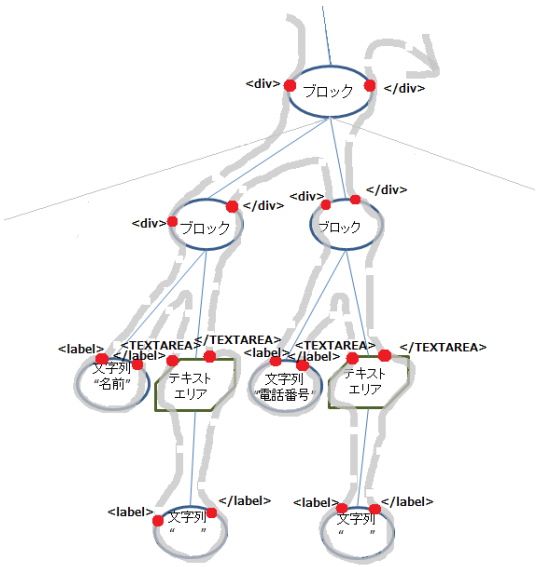

図 4　生成の走査-HTML-

4.3.3　Swift で表現した外部表現

Java と同じアルゴリズムにより生成を行なうので詳細は省略する．実行結果は図 7 のとおりである．

図 7　実行結果-Swift-

5　考察

本研究では、プラットフォームを言語とした MDA に基づき View の自動生成について考察した．DIC は PIM である View のオブジェクトモデルを入力とし，PSM である特定のプログラミング言語で記述された DisplayImage を生成する．HTML，Java，Swift について DisplayImage の生成を実現した．これらの言語ではそれぞれ生成手順が異なる．HTML はタグの入れ子で，木構造を表現している．ViewModel を行き掛け順に走査した場合，ノードに対する行きで対応する開始タグを生成する．親の要素に戻る際に終了タグを生成する．Java や Swift のようなオブジェクト指向プログラミング言語の場合，View は木構造で表現されるが，この木構造は包含関係で実現する．子のインスタンスを生成し，次に親のインスタンスを生成，親に子を追加する必要がある．行き掛け順の場合，帰りにインスタンスを生成する．これらの違いは言語仕様の違いである．我々は生成対象が増えた場合に対応可能となるように図 8 で表記されている生成系の生成系を実現することを考えた．プラットフォームはプログラミング言語である．PSM としては特定の言語を生成する DIC である．この DIC は ViewModel を行き掛け順に走査した際，各ノードで DisplayImage を生成する Visitor として実現される．Visitor の PIM はクラス図，シーケンス図 (図 9 図 10) で表現する．生成系の生成系に対して言語仕様を与えることにより MDA による自動生成を実現する．それにより View 定義形式をプラットフォームとした View のオブジェクトモデルからの MDA による自動生成を行うことができるようになる．縦の自動生成で言語の構造に関する情報を，横の自動生成で使用するライブラリに関する情報を与えることで View のオブジェクトモデルから View 定義形式の自動生成を行うことができるようになる．

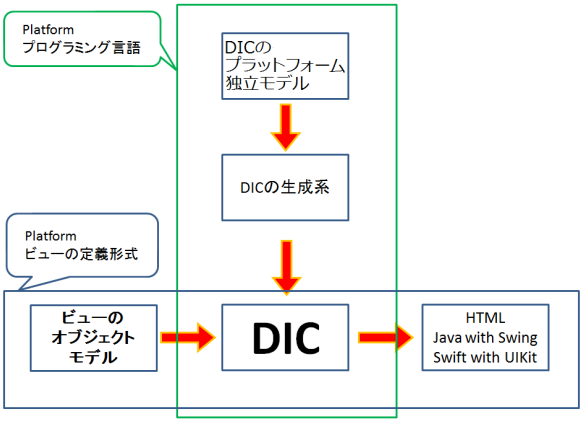

図 8　2 段階 MDA

6　おわりに

多岐にわたる実行時環境のそれぞれを詳細にいたるまで把握することが困難であることを問題とした．これはプログラミング言語が多岐にわたり、また新規プログラミング言語が提案されることに起因すると考えた．多岐にわたるプログラミング言語それぞれについて説明可能な共通アプリケーションアーキテクチャの ViewModel をプラットフォームを言語とした PIM とした．ViewModel の要素と特定の外部表現との対応付けを行い，MDA に基づいて特定のプログラミング言語での実装を行う DisplayImageConstractor を作成した．環境によって画面の描画は変化する．それぞれの環境に適した出力をすることで，アプリケーションをそれぞれ用意する負担の軽減になったと考える．HTML，Java，Swift を例に，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と考えた．今後の課題として言語仕様の定義があり，できれば負担の軽減に繋げることができると考える．

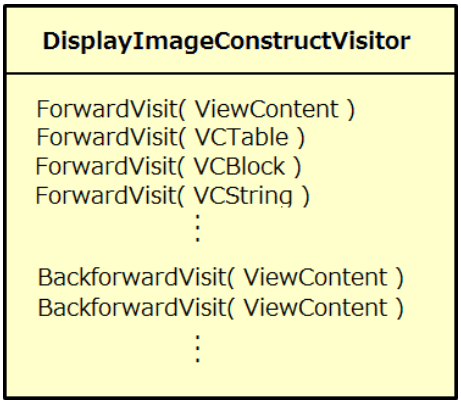

図 9　DICV クラス図

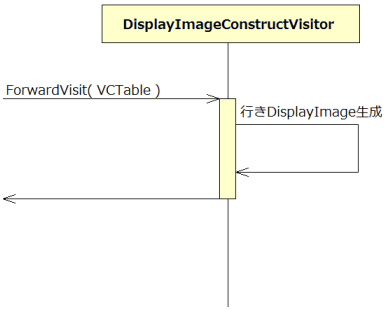

図 10　DCIV シーケンス図

参考文献

[1] 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. 1-8，2015.

[2] 張 漢明，沢田 篤史，野呂 昌満, “E-AoSAS++ における振舞い検証の枠組み”， 電子情報通信学会技術研究報告， ソフトウェアサイエンス研究会， vol. 109，No. 456, SS2009-64，pp. 97-102, 2010.

[3] 戸川 望，組み込みシステム概論，CQ 出版社，2008.