MVVM と Xamarin
-多様化する IT 環境への応用技術の取り組み-

ソリューション本部　開発一部

永井 啓介

1. はじめに

複雑な操作を伴うソフトウェアの開発は UI から処理部へのパラメータの引き渡し、または内部状態の UI への反映等、UI と処理部の「同期」が必要となる。「同期」のために処理側と画面側をつなげる必要があり、情報にアクセスするための I/F を作成しなければならない。

さらには直接お互いにアクセスするようなコードを書かなければならないこともあり、プログラムが面倒で煩雑になりがちである。

こういった煩わしさを解決するために過去に UI と処理を分離させるための様々なデザインパターンが生み出された。

有名なものは MVC（Model View Controller）であるが、近年では更に発展した MVVM（Model View ViewModel)デザインパターン（以降 MVVM と呼ぶ）といったものも出てきた。MVVM は開発・仕様変更・メンテナンスにおいて MVC よりも効率が良いとされているが、現在のところ JavaScript や TypeScript にも広がりはあるものの「ほぼ」C#専用になっている。

MVVM は Silverlight に採用されていたが、残念ながら Silverlight 自体がメジャーにはならなかった。現在は WPF（Windows Presentation Framework）の標準的なデザインパターンとして活躍している。

一方、各種 ICT 端末/CPU の進化に伴い、PC/スマートフォン/タブレットなど多様な媒体・マルチプラットフォームに対応したソフトウェアの開発要件が増えている。それに伴い、異なるプラットフォームにおいて統一されたデザインパターンで開発を進めることが困難になってきている。

動作対象が増えることは

① 各プラットフォームに合わせた学習コスト/協力会社が必要。
② 過去の資産が活用しにくい。
③ ユーザーも間接的にコストが掛かる。

といった問題にも繋がっている。

本稿では当社の得意とする「図形表示・操作」を行うアプリケーション開発において、MVVM と Xamarin（後述）で、どのように前述の問題に対処しているかを述べる。

2. システムの形態と特徴

まず、一般的に「システム」はブラウザ上で動作するアプリケーションなのか、ネイティブで動作するアプリケーションなのかで二分される。ブラウザアプリケーションは、インストールが不要で幅広い動作環境をサポートするというメリットがある反面、実行速度や、機能の充実度においてネイティブで動作するアプリケーションに劣る。一方ネイティブのアプリケーションは豊富な機能を備えることができる反面、動作環境が限定される面がブラウザ上で動作するアプリケーションに劣る。近年では PaaS（Platform as a Service）や SaaS(Software as a Service)の発達により、

一部はローカルで、一部はクラウド上で実行されるといったことも珍しくない。

様々な環境の中で、我々は目的に沿った形態のシステムを構築する必要がある。

3. デザインパターンの統一

3. 1 デザインパターンの選定

システムを開発する上で、デザインパターンを統一することは、長期的に見ると開発コストを大きく下げることに繋がる。これは次々に出てくる新しい技法への学習コストを低減するのみでなく、統一することにより「間違い」が少なくなり修正等に掛かる後工程のコストが小さくなるからである。当社もこの必要性を感じていたが、社内/協力会社のメンバーに圧倒的に C#人口が多いことから、MVVM を採用することにした。但し、MVVM は WPF に特化しているため、WinForm で開発してきたプログラマにとっては、作法が異なりすぎて少々敷居が高いのはわかっていた。

3. 2 WinForm と MVVM のプログラム構造の違い

WinForm のプログラム構造は図 1 に示すように、UI に紐づけされた各コントロールのイベント処理やボタン等の実行可否を制御するプログラム、内部データと画面表示の同期処理等を UI のクラス側に記述したり外部クラス側に記述したりとお互いの結合性が高く、そのため構造が煩雑になりがちである。

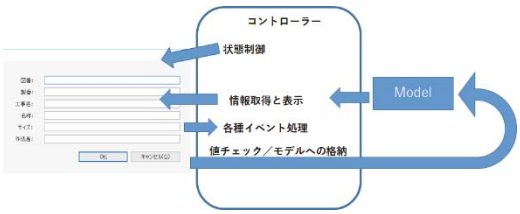


図 1 WinForm のプログラム構造

一方、MVVM のプログラム構造は図 2 に示すように、View、ViewModel、Model 間は疎結合となっており、処理側は画面の情報・状態を意識することなくプログラムの記述を行えるため、構造がスッキリとまとまりやすい。

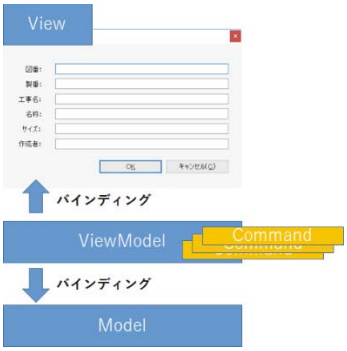


図 2 MVVM のプログラム構造

MVVM に対応するために、初期の学習コストはある程度掛かってしまうが、将来の「資産活用」、「メンテナンス性」、前述の「同期処理」の煩わしさからの解放を考えるとここで学習コストをかけてもデザインパターンを統一しておくことは価値があると判断した。

4. 開発環境の統一

MVVM を採用したもう一つの理由として Xamarin の存在がある。(正確には Xamarin.Forms の存在が大きい。)

Xamarin が発表された 2013 年頃、日頃 Windows のデスクトップ用アプリケーションの開発でお世話になっている C#で iOS/Andoroid のクロスプラットフォーム開発が出来ると知り大きな関心を持ったが、UI については各プラットフォームの作法に従う必要があると知り、その学習コストや開発コストを考えると当時は採用には至らなかった。(MVVM で構築できるのは魅力であったが。)

しかし、Xamarin にはいつの間にか各プラットフォームにおいて共通で使うことのできる UI の Xamarin.Forms が出来ていた。

Xamarin.Forms では記述は共通だが、実行時にコントロールを各プラットフォームのものに置換される。これを知り、これは使えるかもという思いになった。

Xamarin.Forms が提供するコントロールはリッチではなく各プラットフォームが必ず持っている共通のコントロールであり複雑なコントロールは使えないため、プロトタイプ開発を Xamarin.Forms で行い、最終的にはプラットフォーム毎に UI を綺麗に作るのが一般的のようであった。

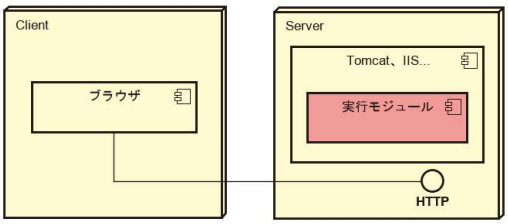
しかし我々の開発するシステムは CAD 系のものが多く、コントロールを駆使して複雑なシステムを組むことより、図形等の配置されている canvas の操作がメインとなるものが多いため、提供されるコントロールがリッチでなくとも「canvas さえなんとかなれば問題ないのでは？」と考えた。

尚、当社(一般的にも同じだが)の CAD 系の案件には大きく次の3つのパターンがある。

4. 1 アドインの開発

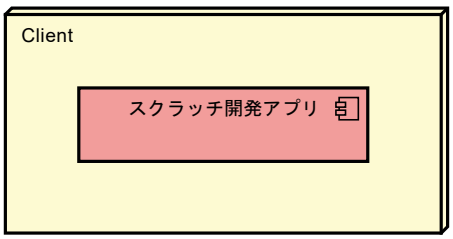
市販の CAD システムのアドインとして専用の機能を追加するパターン。CAD のコマンドとして専用の機能が追加される。

4. 2 ブラウザ上のアプリ開発



ブラウザ上で動作する CAD 系のシステム。HTML5 や WebGL によりかなり高機能なことが実現できている。

4. 3 スクラッチ開発



UI も含めた独自開発のシステム。複雑な CAD の機能が必要でない場合はこのパターンが適用される。

アドイン開発やブラウザ上で動作するアプリの開発は既に確立されたデザインパターンがあるため、

それらを捨ててまで、新たな技術を採用ことは難しいが、スクラッチ開発については MVVM＋Xamarin 開発が比較的スムースに適用できると考えた。

但し、肝心の図形の操作をどうにかする必要があった。Xamarin では WPF のようなリッチな Geometry やシェイプは使用できない。Xamarin.Forms には代用コントロールはない。かといって、プラットフォーム毎に独自の開発をすると開発コストが膨み、メンテナンス性も悪くなる。

この問題については Microsoft の MVP 保持者に相談やネットでの調査の結果、Google や mono の提供する描画コンポーネントで代用出来そうなことが分かり、開発環境についても、統一化の目途が立った。

5. 制限

技術的な問題はクリアしたものの、やはり完全に同じプログラムソースのみによるクロスコンパイルが出来るわけではなかった。ファイル等の外部資源アクセスの概念がセキュリティを含めプラットフォーム毎に異なるため、その処理はプラットフォーム毎に開発する必要はある。今のところどうしようもなさそうである。iOS や Android では特定の場所の資源しか使用できない。

6. 異なる入力方式の吸収

もう一つの問題は、入力方式の違いである。パソコンであれば一般的にはマウスやキーボードだが、タブレッドやスマートフォンはタッチ操作になる。

Xamarin.Forms は入力デバイスにあった入力コントロールが自動的に採用されるが、canvas 部分の操作はそうならない。画面のズームなどはパソコンではマウスホイールで行うのが一般的だが、タブレッドでは指2本を広げるといった操作が一般的なので、両方のイベントに対応する必要がある。

しかしその先の「処理内容」は同じである。イベントに対する処理自体は MVVM の Command が受け持つため、タッチのイベントパラメータをマウスイベントのパラメータに変換し Command に渡すことで処理の共通化を図ることが可能になった。

7. 事例

スクラッチ開発のレイアウト配置システムに MVVM＋Xamarin を適用した。詳細な内容を書くことができないが、ある領域の中に要素を最適配置するのを支援するためのシステムである。このシステムでは図形の操作に加え、近年注目されているジェネレーティブデザイン[1]の要素があったため、膨大な幾何計算が必要となった。高速なプログラミング言語で構築することが有利なため、インタプリタ型ではない D 言語（DMD）,C,C++,C#等が開発言語の候補となったが、タブレッドにも対応する必要があったため、ここで初めて C#による MVVM+Xamarin での開発に踏み切った。

不慣れであったため構築には手間取ったが最終的には図 3 のように View がプラットフォーム毎に分離して、その他は共通で使用するプログラム構造となった。

---

[1] 定められた条件を元に可能性のあるソリューションを全て見つけ出し、設計案を素早く生成する技法

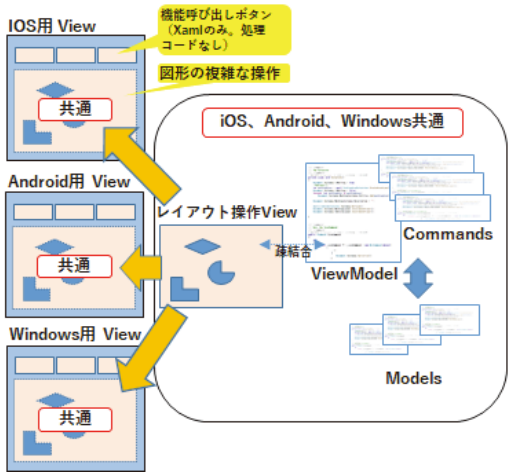


図 3 MVVM＋Xamarin を適用したシステム構成例

結局プラットフォーム毎に異なる View となってしまったが、これは各 OS によって Effect を変えるためである。

レイアウトの操作は分離された View の中に共通の View を埋め込んだ。ここには前述の Google の提供する描画コンポーネントを使用している。

処理は MVVM の ViewModel、Model、Command が司っておりプラットフォーム共通のソースとなる。ここのソースの共有率は全体の9割以上である。

Xamarin＋MVVM でシステムを構築し気付いたことは、「WPF での開発工数とあまり変わらない」ことであった。無論 iOS、Android 用にタッチの I/F 等を作成する必要はあるが、これは Windows だけでもタッチ対応のシステムを作る手間と同様である。

もう一つ、「実行速度が速い」と感じた。特に UI において、シングルプラットフォームをターゲットとした専用のソフトに比べると不利かもしれないと予想してたが、そんなことはなかった。（むしろ早いかもと感じたくらいである。）

デバッグ面においても、開発中は全てのプラットフォーム用にビルドする必要がなく、テストを環境用のモジュール用のみビルドすれば良かったため、効率は悪くなかった。

その他のメリットとして、先述で膨大な幾何計算が必要と記したが、ローカルデバイスの計算能力では処理が難しい場合、その処理をクラウド上の強力なサーバ上で実行させることができる。SaaS の更に上の階層にあたる FaaS（Function as a Service）または BaaS（Backend as a Service）に該当する概念であるが、MVVM でシステムを構築しておくと処理部である Command はもとより View に対し疎結合であるため、サーバ上に再配置することは「容易い」（これは.Net 自体の恩恵も大きい。）

これにより非力なスマートフォンやタブレッドで動作させるシステムでもその性能をあまり気にせずに設計や構築が行えるようになった。

このようにして、当社は Xamarin＋MVVM を用いることにより、クライアントのみでなくクラウドも含め多様化している環境に対応したシステム作りができるようになった。

8. 終わりに

Xamarin、MVVM と Microsoft 発信の技術ばかり論じてしまったが、もう一つ Microsoft のサービスを紹介したいと思う。

前述のクラウドに対応したシステムになると、モジュールの配置が煩わしく、デグレーションを引き起こしてしまう可能性も伴う。

そこで開発から運用までをサポートしてくれるのが、Microsoft の提供するクラウドサービスである Azure DevOps である。（「DevOps」自体は Microsoft 固有の技術ではなく、ソフトウェア開発手法の名称である。）

Azure DevOps はリポジトリ管理、タスク管理、テスト計画等多くのサービスを提供するが、

クラウド上でビルドを行い、Deploy（実行環境の展開）を行う Pipeline というサービスがある。こういったサービスを利用することで、迅速に、且つ安全にクラウド上に実行環境を配置することができるため、ネイティブアプリケーションもブラウザ上で動作するアプリケーションのような運用に近くなってきたように思える。機会があれば是非使ってみたいと考えている。

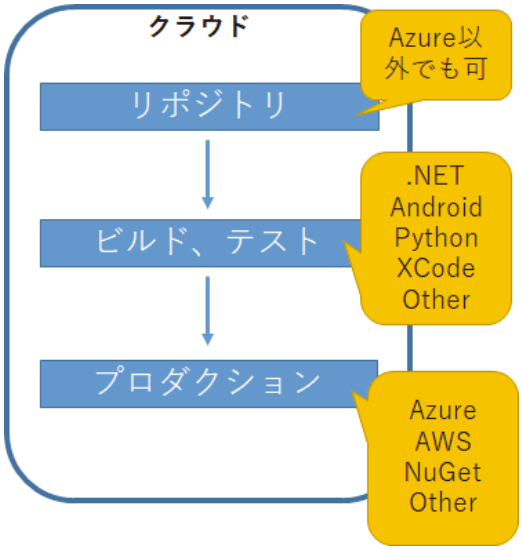


図 4Pipeline のイメージ