Silverlight 4

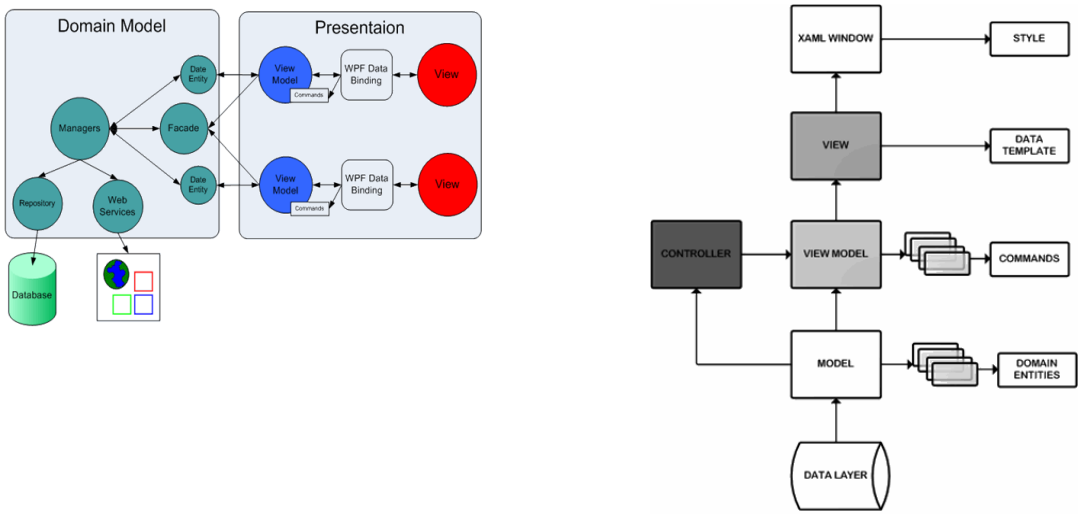
پژوهش و نگارش: وحید نصیری
Vahid_Nasiri@Yahoo.Com
بهار ۱۳۸۹

فهرست مطالب

فصل ۹ – معرفی الگوی M-V-VM

M-V-VM چیست؟

M-V-VM یا Model-View-ViewModel یکی از الگوهای محبوب طراحی رابط کاربر در WPF و Silverlight می‌باشد که توسط John Gossman از تیم WPF ایجاد شده است و قدرت خود را مدیون توانمندی‌های binding پیشرفته‌ی WPF و Silverlight است. به کمک آن می‌توان View (یا همان قسمتی از برنامه که کاربر با آن سر و کار دارد) را از کدهای مرتبط با داده‌ها و منطق برنامه مجزا ساخت. به این صورت در یک تیم، افرادی می‌توانند بر روی View در Expression Blend کار کرده و همزمان تعدادی دیگر در Visual Studio .NET مشغول تهیه قسمت ViewModel باشند.



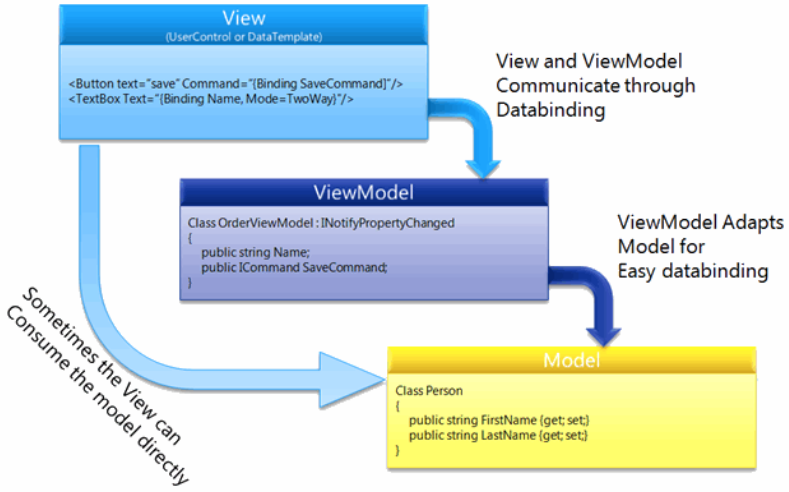
شکل ۱– نمایی از قرارگیری و نحوه‌ی تعامل لایه‌های مختلف یک برنامه در الگوی MVVM.

روش متداول آموزش و کار با WPF و Silverlight همانند روزهای WinForms است یا دوران VB6 . تمام کنترل‌های بصری نامگذاری می‌شوند. سپس روال‌های رویدادگردان کلیک و امثال آن به صورت مستقیم تعریف شده و در code behind صفحه، منطق مرتبط با آن پیاده سازی می‌شود (در کدهای View ارجاعات مستقیمی را از Model خواهیم داشت). همچنین این پیاده سازی به همراه استفاده مستقیم از نام اشیاء و کنترل‌هایی است

که در View تعریف شده‌اند. حاصل این عملیات متداول، گره خوردگی کدهای صفحه با View است که امکان نوشتن آزمون‌های واحد آن را به صفر می‌رساند. همچنین از امکانات پیشرفته‌ی WPF و Silverlight مانند انقیاد دوطرفه (two way binding) و بسیاری موارد دیگر نیز استفاده نخواهد شد. به علاوه با توجه به استفاده‌ی مستقیم از نام کنترل‌های بصری در code behind ، تغییر یا جایگزینی یک کنترل نیاز به تغییرات وسیعی در کدهای ما خواهد داشت. برای رفع این مشکلات، الگوی M-V-VM ارائه شده است که توسط خود مایکروسافت جهت بهره گیری از تمامی امکانات WPF و Silverlight ابداع گردیده و در تهیه‌ی محصولات مهم آن شرکت مانند Microsoft Expression Blend نیز بکار گرفته شده است.

آشنایی با اجزای مختلف الگوی M-V-VM

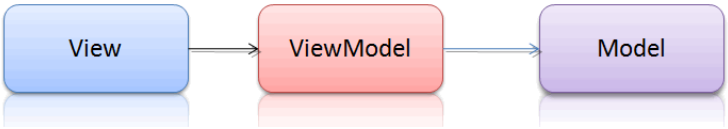
به صورت خلاصه هدف اصلی الگوی M-V-VM ، جداسازی مسایل مرتبط با رابط کاربر (View)، از اشیاء تجاری و حالات آن‌ها (ViewModels) و همچنین از لایه داده‌ها و دسترسی به داده‌ها (Model) است.



شکل ۲– نمایی از نحوه‌ی تعاریف و تعاملات اجزای مختلف در الگوی M-V-VM .

- **Model** : گروهی از موجودیت‌ها می‌باشند که ارائه خواهند شد. برای مثال کلاس مشتری به همراه خواص نام و شماره شناسایی او. مدل‌های برنامه از اینکه اطلاعات آن‌ها از کجا تهیه خواهند شد یا چگونه به کاربر نمایش داده می‌شوند اطلاعاتی ندارند. یک مدل خوب نباید نیازی به ارجاعاتی به PresentationFramework.dll ، System.Windows.Forms ، PresentationCore و امثال آن‌ها داشته باشد. همچنین باید بتوان به سادگی از آن حتی در یک برنامه‌ی کنسول نیز استفاده کرد. البته باید در نظر داشت که پیاده سازی‌های متفاوتی از مدل یا سایر قسمت‌های این الگو مشاهده می‌شود و الزامی به رعایت حتمی تک تک این موارد نیست.

- **View** : همان کدهای Xaml ایی هستند که جهت نمایش رابط گرافیکی تهیه می‌شوند و نه بیشتر. View می‌تواند شامل کنترل‌های بصری، پویانمایی و غیره باشد به همراه تعاریف binding به اطلاعات؛ اما با این شرط که View نمی‌داند این اطلاعات در کجا قرار دارند و یا چگونه تهیه خواهند شد. عموما از UserControl ها و یا Data Templates برای تهیه View استفاده می‌کنند. یکی از اهداف الگوی M-V-VM داشتن View هایی است که دارای Code behind نیستند و هیچ نوع منطقی را پیاده سازی نمی‌کنند.
- **ViewModel** : مدلی آینه‌ای از View است به شکل یک کلاس (Model of a View)؛ که توسط آن می‌توان برای کل View بجای تعامل مستقیم با آن، آزمون واحد (Unit test) تهیه نمود. ViewModel محل اتصال View با دنیای خارج است (به آن meta-view هم گفته می‌شود). توسط این کلاس است که اطلاعات binding و Commands در اختیار View قرار می‌گیرد. ViewModel بیانگر حالت و رفتار View (های) متناظر آن است.

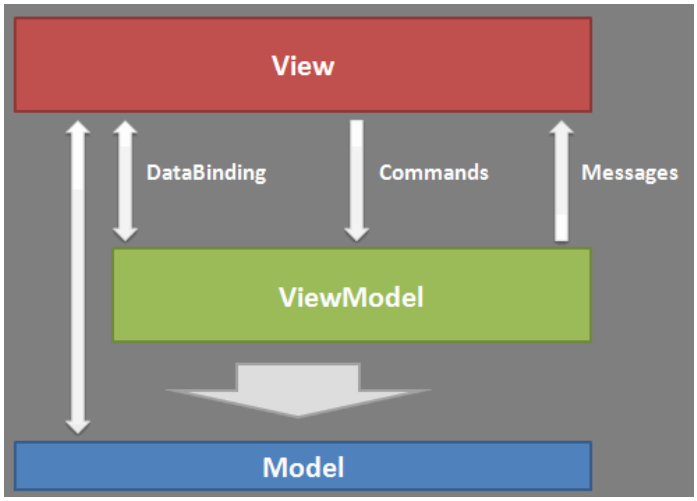


شکل ۳– نحوه‌ی ارتباط بین اجزای مختلف الگوی M-V-VM .

در اینجا هر خاصیتی که قرار است در binding شرکت داشته باشد باید اینترفیس INotifyPropertyChanged را پیاده سازی نماید. در این الگو حرکات و تعاملات کاربر مانند کلیک بر روی یک دکمه به صورت Commands ارائه می‌گردند. در زمان ایجاد ViewModel ، تنها ساختار View است که باید مد نظر قرار گیرد و نه ظاهر کلی آن. همچنین جهت بالابردن قابلیت آزمون‌های واحد آن این کلاس نباید هیچگونه ارجاعی را به کنترل‌های بصری داشته باشد.

در الگوی M-V-VM همانطور که در شکل ۳ نیز مشخص شده است، View تنها از ViewModel اطلاعات لازم را دریافت می‌کند و هیچگونه اطلاعاتی از Model ندارد. ViewModel اطلاعات مورد نیاز خود را از Model تهیه می‌کند اما اطلاعاتی از View ندارد و Model تنها خودش را می‌شناسد. در شکل ۴ ارتباط مستقیم View با Model نیز نمایش داده شده است. گاهی از اوقات اگر ViewModel شما هیچگونه منطقی را پیاده سازی نمی‌کند، لزومی هم به ایجاد آن نیست؛ در غیر اینصورت حتما کار وفق دادن اطلاعات Model با View را از طریق ViewModel پیاده سازی نمائید.

View به خواص تعریف شده در ViewModel بایند (مقید) خواهد شد (توسط DataContext هر View). با توجه به پیاده سازی اینترفیس INotifyPropertyChanged ، هر تغییری در خواص ViewModel بدون نیاز به کد نویسی، در UI و View برنامه منعکس می‌گردد. بنابراین تغییرات بر روی داده‌ها تنها در ViewModel باید صورت گیرد و نه در View .



شکل ۴– نمایی از نحوه‌ی ارتباط بین ViewModel و View در الگوی MVVM .

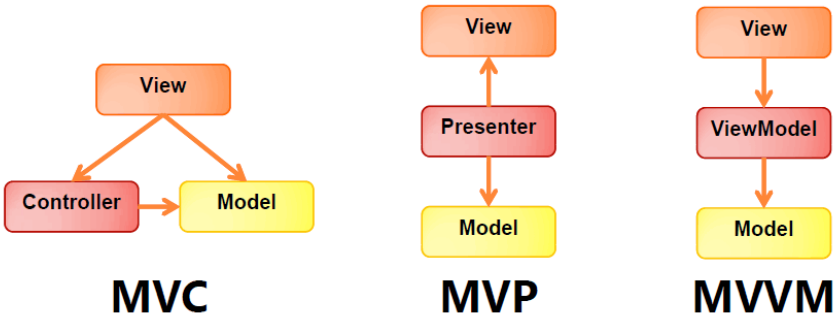
به صورت خلاصه نحوه‌ی تعامل بین View و ViewModel به شکل زیر است:

- ارتباطات بین View و ViewModel از طریق data-binding ، فراخوانی متدها، خواص، رخدادها و پیغام‌ها است.
- ViewModel علاوه بر ارائه‌ی خواص ضروری Model به View ، می‌تواند یک سری حالات و یا Commands را نیز در اختیار View قرار دهد.
- رخدادهای یک View از طریق Commands به ViewModel منتقل می‌شوند.
- خواص ارائه‌ی شده‌ی ViewModel به View از طریق binding دو طرفه به روز رسانی می‌شوند.

و نحوه‌ی تعامل بین ViewModel و Model به صورت زیر است:

- ViewModel می‌تواند تمام خواص یا تعدادی از خواص مفید Model را جهت data-binding در اختیار View قرار دهد.
- ViewModel می‌تواند شامل اینترفیس‌هایی به سرویس‌ها یا تنظیماتی خاص جهت دریافت و اعمال تغییرات بر روی اطلاعاتی باشد که به View ارائه می‌دهد.

شکل بعد روابط اجزای سه معماری *MV را بیان می‌کند و با توجه به آن کاملا مشخص است که یکی از اهداف اصلی الگوی MVVM ، سهولت هرچه بیشتر انجام آزمون‌های واحد است و از هر دو معماری MVP و MVC قابلیت آزمایش پذیری بهتری را ارائه می‌دهد.

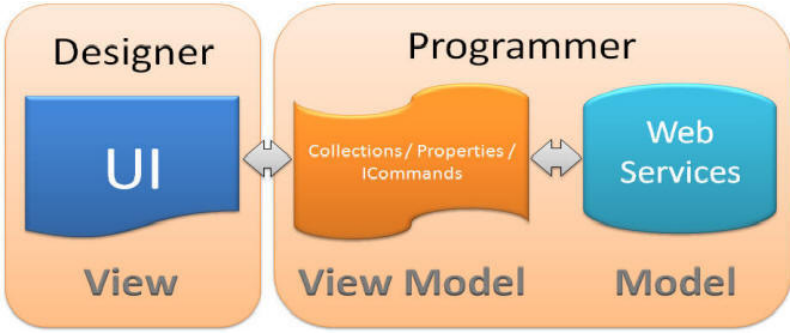


شکل ۵– مقایسه‌ای از روابط اجزای مختلف سه معماری متفاوت *MV .

مزایای استفاده از الگوی M-V-VM

- جدا سازی لایه‌ها از یکدیگر: به این صورت تغییرات صورت گرفته در هر کدام از لایه‌ها به علت عدم تنیدگی در دیگری، با سهولت بیشتری صورت خواهد گرفت و نگهداری برنامه را در دراز مدت ساده‌تر خواهد کرد.
- امکان فعالیت تیمی بهتر : با توجه به جداسازی لایه‌ها، طراح رابط کاربر و برنامه نویس‌ها همزمان می‌توانند کارهای خویش را انجام دهند (شکل ۶).
- تهیه آزمون‌های واحد : همیشه تهیه‌ی آزمون واحد برای UI و کدهای event driven کاری بسیار مشکل است. اما در اینجا ViewModel را به سادگی می‌توان با کمک آزمون‌های خودکار بررسی کرد و میزان کدهای View به حداقل ممکن می‌رسد.
- پشتیبانی بهتر از چندین View : با توجه به جدا سازی لایه‌های مختلف برنامه، امکان تهیه برنامه‌هایی که به سادگی قابل تبدیل به WPF و یا Silverlight هستند را خواهید داشت. همچنین می‌توان برای مثال دو View مختلف را ارائه داد به صورتیکه از یک ViewModel استفاده می‌کنند و تفاوت آن‌ها در نحوه‌ی نمایش و قالب بندی اطلاعات ارائه شده توسط ViewModel است، بدون اینکه نیازی به تکرار کدها در برنامه وجود داشته باشد. به این صورت تغییر و یا تعویض یک View بدون تغییری در کدهای برنامه میسر خواهد شد.
- سادگی استفاده از کنترل‌های WPF و Silverlight: کنترل‌های بصری WPF و Silverlight اساسا جهت کار با این الگو طراحی شده‌اند و بسیاری از قابلیت‌های آن‌ها با کمک الگوی M-V-VM بهتر نمایان خواهند شد.

- Blendability : به قابلیت ایجاد و ویرایش داده‌های آزمایشی در زمان طراحی رابط کاربر در Expression Blend (و یا حتی طراح Visual Studio) جهت مشاهده‌ی بهتر و نزدیک به واقعیت View تولید شده گفته می‌شود که توسط این الگو بهتر از پیش میسر خواهد شد.

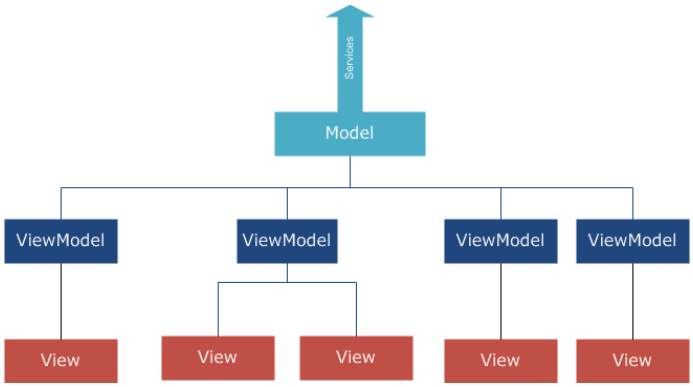


شکل ۶– یکی از مزایای الگوی M-V-VM امکان کار همزمان طراح رابط کاربر برنامه و برنامه نویس‌ها در یک تیم است.

اصول کاری و بایدها و نبایدهای الگوی M-V-VM

بایدها و نبایدهای یک View

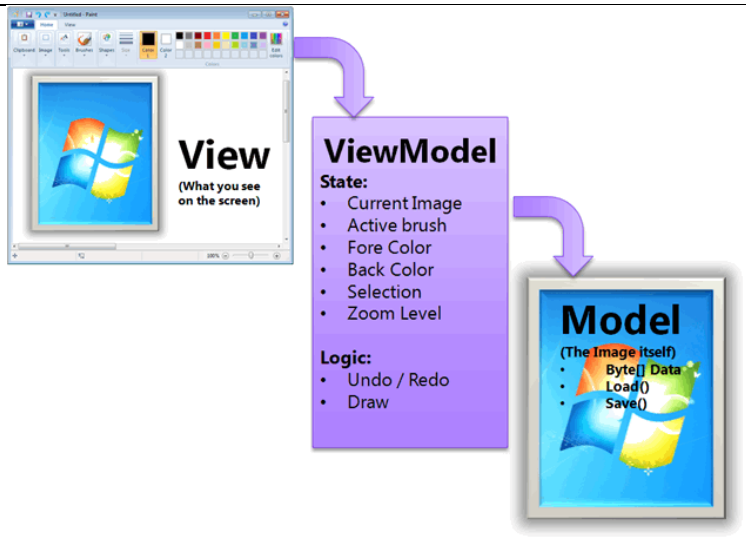
- وظیفه‌ی آن نگهداری حالات خود نیست. ViewModel است که این وظیفه‌را به عهده می‌گیرد.
- نباید در آن کدهایی که نیاز به آزمون واحد دارند، پیاده سازی شوند. در حالت کلی، کد نویسی در Code behind یک View منعی ندارد اما اگر برای مثال در View قسمتی جهت انتخاب نام یک کشور و سپس قسمتی دیگر جهت نمایش مناطق مختلف آن کشور وجود دارد، این مورد نیاز به نوشتن آزمون واحد داشته و نباید در Code behind پیاده سازی شود.
- نباید در Code behind آن روال‌های رخداد گردان کلیک بر روی یک دکمه و مانند آن مشاهده شود.
- می‌تواند یک user control و یا Data Template باشد.
- View را تا حد امکان ساده نگه دارید، هر چند استفاده از Data Trigger ، Value Converter و موارد دیگر منعی ندارند.
- اگر عنصری به سادگی، قابلیت bind نداشت برای آن attached property تدارک ببینید.
- تنها باید از اطلاعات ViewModel استفاده کرده و نباید به صورت مستقیم هیچگونه تعاملی با Model داشته باشد.
- مهم‌ترین هدف و وظیفه‌ی یک View باید قالب بندی، بازآرایی و همچنین نمایش بصری اطلاعات باشد.



شکل ۷– نمایی از روند کاری و ارتباطات قسمت‌های مختلف یک برنامه مبتنی بر الگوی M-V-VM .

بایدها و نبایدهای `ViewModel`

- وظیفه‌ی آن شکل دهی به اطلاعات، مرتب سازی آن‌ها و سپس ارائه‌ی این داده‌ها به View است. برای مثال Model اطلاعات تاریخ را به فرمی خام نگهداری می‌کند، ViewModel به آن‌ها فرمت قابل ارائه به کاربر را بخشیده و سپس View این اطلاعات نهایی را نمایش می‌دهد.
- هیچگونه ارجاعی از View نباید در آن وجود داشته باشد.
- تا حد ممکن قابلیت آزمون واحد پذیری آن را درنظر داشته باشید. برای مثال فراخوانی کلاسی از نوع Singleton در آن نباید وجود داشته باشد.
- نباید در آن اثری از کنترل‌های بصری View دیده شود. در غیر اینصورت علاوه بر مشکل شدن تهیه آزمون‌های واحد برای یک ViewModel ، به محض تغییر View باید ViewModel ما نیز کلا تغییر کند. بنابراین ارجاعی از PresentationFramework در آن نباید وجود داشته باشد. به عبارت دیگر طراحی آن باید logical باشد و نه Visual .
- تغییرات را باید از طریق پیاده سازی INotifyPropertyChanged به View منعکس کند (الگوی Observer).
- نوع خواص تعریف شده در آن باید محدود به یک ViewModel دیگر یا نوع‌های اصلی و پایه و یا مجموعه‌ای از آن‌ها باشد.
- یکی از مکان‌هایی که می‌توان در آن پیاده سازی تعیین اعتبار ورودی کاربر را انجام داد ViewModel مرتبط است؛ زیرا لایه‌ای است میان Model و View برنامه. همچنین در این لایه بسیاری از اعمال تبدیلی را جهت حذف تبدیل‌کننده‌های WPF و Silverlight نیز می‌توان مد نظر داشت.



شکل ۸– نحوه‌ی تعریف، حیطه‌ی عملکرد و ارتباطات اجزای الگوی M-V-VM .

بایدها و نبایدهای Model

- وظیفه‌ی آن مدیریت و ارائه‌ی اطلاعات به ViewModel است. مخزنی است از اطلاعات اما وظیفه‌ی آن دریافت اطلاعات از یک دیتابیس یا سرویس و یا تغییرات بر روی آن‌ها نیست. منطق تجاری باید از مدل مجزا شده و در کلاس‌های دیگری نگهداری شود.
- نباید ارجاعی از View و یا ViewModel در آن وجود داشته باشد. برای مثال نباید در یک مدل ارجاعی به PresentationFramework و موارد دیگری که پیشتر از آن‌ها یاد شد، مشاهده شود.
- نباید در سایر لایه‌ها، نیازی به کلاس‌های دیگر برای کار با آن وجود داشته باشد. به بیان دیگر هر نوع عملیاتی که نیاز است بر روی داده‌ها صورت گیرد باید توسط این کلاس‌ها انجام شده و سایر کلاس‌ها نباید به صورت مستقیم سر و کاری با بانک اطلاعاتی و ساختار آن و یا ذخیره یا بازیابی اطلاعات داشته باشند.
- زمانیکه نیاز به درخواست بازخورد از کاربر در آن وجود داشت باید از Events استفاده گردد.
- یک مدل می‌تواند با پیاده سازی IDataErrorInfo قسمتی از کار تعیین اعتبار داده‌ها را نیز به عهده بگیرد. به بیان دیگر تهیه‌ی مدلی که هیچگونه منطق تجاری را پیاده سازی نمی‌کند در اکثر مواقع میسر نیست و در این موارد بهتر است منطق تجاری اعمالی بر روی مدل، در کلاس‌های مجزایی نگه‌داری شوند.

مروری بر معایب الگوی M-V-VM

- با توجه به اینکه این الگو توسط تیم WPF ارائه شده است اما هنوز استاندارد سازی نشده و همچنین روش‌ها یا ابزارهای استاندارد و کاملا مشخصی نیز برای پیاده سازی آن ارائه نشده است. به همین

جهت امروزه شاهد چندین Framework توسعه یافته توسط برنامه نویس‌های مختلف در این زمینه هستیم و هر روز نیز تعداد آن‌ها افزایش می‌یابد (مانند Onyx ، Prism ، Microsoft WPF Toolkit، Crack.net ، Caliburn ، MVVM Light Toolkit و ...).

- مطابق نظر خالق این الگو، John Gossman ، الگوی M-V-VM برای برنامه‌های کوچک شاید مناسب نباشد و پیچیدگی‌های بی‌جهتی را به آن‌ها تحمیل کند.

هر چند مورد اول ذکر شده را می‌توان نوعی مزیت نیز برشمرد. به این صورت افراد مختلف با توانایی‌های متفاوت می‌توانند Framework درخوری را بیابند.

و در پایان باید در نظر داشت که برای توسعه‌ی هر محصولی باید از ابزار متناسب با آن بهره جست. برای مثال در توسعه و طراحی یک بازی نوشته شده با WPF و یا Silverlight الگوی MVVM شاید انتخاب مناسبی نباشد اما در سیستم‌های مبتنی بر داده و فرم‌های ورود اطلاعات، انتخاب اول به شمار می‌رود.

شکل ۹ – تصویری نمادین از قابلیت تغییر و نگهداری ساده‌تر برنامه‌های مبتنی بر MVVM .