

## Build your own CAB Part #8 – Assigning Responsibilities in a Model View Presenter Architecture

Posted by [Jeremy Miller](#) on [June 10, 2007](#)

First, go catch up on what's come before:

1. [Preamble](#)
2. [The Humble Dialog Box](#)
3. [Supervising Controller](#)
4. [Passive View](#)
5. [Presentation Model](#)
6. [View to Presenter Communication](#)
7. [Answering some questions](#)
8. [What's the Model?](#)

### Where should this code go?

In a post last winter I said that a coder only becomes a true software craftsman when they start asking themselves “[where should this code go?](#)” It's a neverending question that you use to guide your designs and assign responsibilities. In a typical Model View Presenter architecture the screens are composed of 4 basic types of classes (“M”, “V”, “P”, and the Service Layer). When you're designing a screen along MVP lines, you need to assign each responsibility of the screen to one of these 4 pieces, while also determining the design constraints upon each piece. The question of “who does what?” isn't perfectly black and white, and there are many variations on the basic structure. That being said, here's my best advice on the duties and constraints of each of the four pieces. As a rule of thumb, try to put any given responsibility as close to the top of this list as possible without violating the design constraints of each piece.

### Service Layer

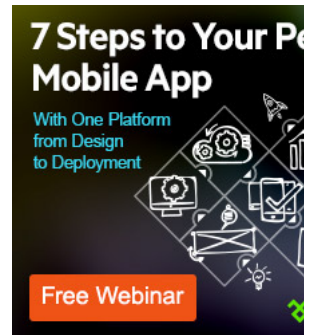
The [Service Layer](#) classes are generally a [Facade](#) over the rest of the application, generally encompassing business logic and data persistence. In general it's important and valuable to decouple the view layer from the rest of the application as much as possible to create [orthogonality](#) between the backend and the user interface. If you find the Presenter collaborating with more than a single Service Layer class you might think about putting a single Facade over the two services for that screen.

In terms of constraints, the Service Layer is not aware of anything specific to any one screen, but it is perfectly permissible for the Service Layer classes to be aware of the Model and perform work on the Model. If you've chosen to use actual [Domain Model](#) classes for the Model in the screen, the Service Layer is probably just some sort of [Repository](#) class. You might think of it like this. Say you're building a monolithic application today, but tomorrow you want to expose web services to interact with the system as an alternative to the user interface. The Service Layer classes should be able to work within the web service code without any change.

The Service Layer should encompass any type of business logic that is not specific or tightly coupled to a specific screen. In my current system there is going to be a validation on certain date values against a calendar service that compares the selected date against the business days for a particular country. If I were to embed that logic into the Presenter for my first Trade screen the logic wouldn't be very accessible to the following Trade screens. That calendar logic definitely belongs in the Service Layer.

### Model

The Model can be more than just a lump of data. For a multitude of reasons, I want my Model class to implement most of the business rules for the given business concept. The first reason is simply cohesion. As much as possible, I want all of my business rules for an Invoice or a Trade or a Shipment in the domain class for that particular domain concept. If I want to look up the business rules for an Invoice, I want a single place to go look. One of my primary design goals is to



**7 Steps to Your Perfect Mobile App**  
With One Platform from Design to Deployment  
**Free Webinar**

### Upcoming posts...

How I'm using StoryTeller to test FubuMVC  
Building a "Lookup" html converter FubuMVC  
FubuMVC's Configuration Mode Sauce  
Managing Script dependencies FubuMVC  
Authorization and FubuMVC Continuations  
Composing Views with FubuMVC  
Extensible Model Binding with FubuMVC  
Introducing "Bottles"  
Modular Packaging with FubuMVC  
Self-Installing Apps w/ FubuMVC  
Routing and Behavioral Conventions FubuMVC  
What Should I Learn?

### Blogroll

[Follow me on Twitter](#)  
[FubuMVC on GitHub](#)  
[My GitHub Page](#)  
[StoryTeller on GitHub](#)  
[StructureMap on GitHub](#)

## ASP.NET Re-Imagined

Touch-enabled web apps are a reality



**DevExpress FREE**

### Archives

[put code where you would expect to find it](#), and to me that means that invoicing business rules go in the Invoice class.

The second reason is reuse or elimination of duplication. If you make a Presenter or the View itself responsible for business rules you're much more likely to find yourself duplicating the same business rules across multiple screens. Those declarative validation controls in ASP.Net are sure easy to use, but there's a definite downside because the responsibility for validation is in the wrong place.

For example, the Model classes in my current system are responsible for:

- **Validation rules.** The validation rules for a domain concept should definitely be kept in the Domain Model. Validation logic is most definitely a business rule. Besides, it's just so common to have multiple screens acting upon the same classes. You don't really want to have to duplicate validation rules from a "Create Trade" screen to the parallel "Edit Trade" screen do you? If nothing else, putting validation rules in the Domain Layer makes these rules very simple to test compared to the same rules living in either the View or even the Presenter. I've got a lot more to say on this subject in the next post on the Notification pattern. In the meantime, Jean-Paul has a good post on [Validation In The Domain Layer – Take One](#) and again [Validation In The Domain Layer – Take Two](#).
- **Default values.** My last two projects have included quite a bit of logic around assigning default values. The normal scenario is that the user selects one value, and from that one value you can determine logical defaults for one or more other fields. I have a little business rule to code tomorrow morning that will automatically set the values for two date fields to 2 days after the Trade Date as soon as the Trade Date is selected for the first time. That rule is going right into the appropriate Trade subclass, both because it's where I'd expect to find that code and because it makes that business rule very easy to drive with Test Driven Development. Because the rule is wholly implemented in the Model class and the Model class is completely decoupled from the View and Service Layer, I can test these rules by applying purely state based testing. For this approach to work I **do** need the screen to automatically synchronize itself with the Model when the Model changes, but the WinForms tooling makes that kind of [Observer Synchronization](#) relatively simple.
- **Calculated values.** Again, this is dependent upon using Observer Synchronization, but I place the responsibility for calculating derived values into the Model class. It's a business rule, and the Model class has all of the data, so it's the natural place for this logic. As with defaulted values, the calculation can be relatively simple to test in isolation.

The constraint on the Model is that whether or not we allow the Model classes to call out to any kind of service, and if so, how much coupling do we allow with other services? In my current project I don't allow any direct coupling from my Model classes to any external system. If a business rule for defaulting or calculating a value requires information or a service from outside the Model, I would partially move that responsibility out into the Presenter to keep the Model decoupled from infrastructure. The Model classes don't know how they're displayed nor how they're persisted and updated. In other systems you might opt for more of an [Active Record](#) approach that puts some form of service communication responsibility into the Model classes.

The Presentation Model approach isn't really an exception to these constraints because the Presentation Model itself usually wraps the actual Model.

#### View & Presenter

The screen itself is split between two main actors, the View and the Presenter. As we've seen in the previous posts on the Supervising Controller, Passive View, and Presentation Model, there's a couple different ways to split responsibilities between the View and Presenter. Rather than rehash those posts, I just want to add a few more thoughts:

- Don't allow the View to spill out into the Presenter. There might be exception cases, but don't allow any reference to any Type in the System.Windows.Forms namespace from the Presenter class. Any WinForms mechanics in the Presenter is like letting water splash out of the tub and rot the floor.
- Keep the View as thin as possible. Always ask yourself if a given piece of code could or should live outside of the View. My advice is to move anything out of the View that doesn't have to be there.
- Don't allow the Presenter to become too big. It's far too tempting to use the Presenter as a receptacle for any random responsibility. If you see parts of the Presenter that don't seem to be related to the rest of the Presenter, do an Extract Class refactoring to move that code to a smaller, cohesive class.
- There is no ironclad rule that says 1 screen == 1 view + 1 presenter. It's often going to be valuable to reduce the complexity of either the Presenter or View by breaking off pieces of the screen into smaller pieces. There may still be one uber-Presenter and one uber-View for the entire screen, but don't hesitate to make specific Presenter or View classes for a part of a complicated screen. There's also no ironclad law that says n views = n presenters as well.

January 2012  
September 2011  
July 2011  
June 2011  
May 2011  
April 2011  
March 2011  
January 2011  
December 2010  
September 2010  
August 2010  
July 2010  
June 2010  
May 2010  
March 2010  
February 2010  
January 2010  
December 2009  
November 2009  
October 2009  
September 2009  
August 2009  
July 2009  
June 2009  
May 2009  
April 2009  
March 2009  
February 2009  
January 2009  
December 2008  
November 2008  
October 2008  
September 2008  
August 2008  
July 2008  
June 2008  
May 2008  
April 2008  
March 2008  
February 2008  
January 2008  
December 2007  
November 2007  
October 2007  
September 2007  
August 2007  
July 2007  
June 2007  
May 2007  
April 2007

Getting back to first causes, it all comes down to two things:

1. Is each piece of the code easy to understand?
2. Is the code easy to test?

Answer these two questions in the affirmative and I think your design is effective.

#### Where Next?

That's the end of the 100/200 level material on "Build your own CAB." Now we get to move on to the cool parts -which will undoubtedly prove more difficult to write. The next post will be on Domain-centric validation with the Notification pattern. After that I think the consensus is to go into building the Application Shell (2-3 posts), then my MicroController stuff (2-3 posts), and automated testing (???). Bringing up the rear will be event aggregation and synchronization with the backend. I am leaving Pluggability until last, but that's because I'm going to slip in a StructureMap 2.1 release at the end of June and demonstrate a couple of new features.

I'm hoping to wrap this up no later than the first week in July.



#### About Jeremy Miller

Jeremy is the Chief Software Architect at Dovetail Software, the coolest ISV in Austin. Jeremy began his IT career writing "Shadow IT" applications to automate his engineering documentation, then wandered into software development because it looked like more fun. Jeremy is the author of the open source StructureMap tool for Dependency Injection with .Net, StoryTeller for supercharged acceptance testing in .Net, and one of the principal developers behind FubuMVC. Jeremy's thoughts on all things software can be found at The Shade Tree Developer at <http://codebetter.com/jeremymiller>.

[View all posts by Jeremy Miller →](#)

This entry was posted in [Build your own CAB](#), [Design Patterns](#). Bookmark the [permalink](#). Follow any comments here with the [RSS feed for this post](#).

– Put Code Where You Would Expect to Find It

We are so spoiled →

11 Comments The Shade Tree Developer

Login

Sort by Best

Share Favorite



Join the discussion...



Oblomovi • 6 years ago

Your models don't work with Services right? And you can do state based testing for model. But what about business entities which represent complex data? Do your models store them? In my models there are business entities, and I'm forced to mock them, therefore I can't do simple state based testing with model :(.

^ | v • Reply • Share



Jeremy D. Miller • 7 years ago

@Geoffrey,

I'm sorry, but I haven't gotten there yet and probably can't until the end of the month. An ApplicationController is from Fowler's PEAA book. It's just the controller class for the shell. I generally give any Command or Presenter class that needs to do navigation a reference to the ApplicationController. The ApplicationController would have methods for navigating to other screens.

^ | v • Reply • Share



Geoffrey • 7 years ago

Hello,

I'm using a MVP model and alle navigation events (eg Edit / Add ) are caught by the presenter ant then the presenter is throwing an "UIEvent" to my main application. There navigation is handled.

But it feels like I could handle the navigation from my presenter. However, no references can be made to winforms lib, so I'm stuck here.

You guys are talking about an applicationcontroller. How does that work and how do you glue it together?

tx

Geoffrey

^ | v • Reply • Share

March 2007

February 2007

January 2007

December 2006

November 2006

October 2006

September 2006

August 2006

July 2006

June 2006

May 2006

April 2006

March 2006

February 2006

January 2006

December 2005

November 2005

October 2005

September 2005

August 2005

July 2005

June 2005

May 2005

April 2005

#### What others have said...

Duncan on What an Amazing (C You've Discovered

pawel paul on FubuMVC Learn: Spark

pawel paul on I'm looking for so testers for StoryTeller (OSS tool testing)

pawel paul on Serenity

pawel paul on My Programming

Are you  
still using  
WCF?



Shane Courtrille • 7 years ago  
Jeremy,

Awesome series but I did want to discuss one point you made in this post. You make the statement that "your service layer is probably just some sort of repository class".

I disagree with that comment but it could just be that I am misunderstanding what you mean. For me a service layer is a place to define "whole" operations that can be carried out on your system. It is a gateway to my domain that is responsible for dealing with things like a unit of work. A repository on the other hand is a domain object I can use to retrieve, update, add and delete objects from.

I've worked on systems that didn't have a service layer at all and you would only deal with a repository but I didn't consider it a service layer. It would seem that by mixing these two concerns you are breaking single responsibility.

Regards,

Shane

^ | v • Reply • Share >



Shep • 7 years ago  
Niyazi,

One way is to expose the properties and events of your user control(s) and have your aspx page pass them thru to the presenter layer. I believe if you read up on the Web Client Software Factory you may find your answer.

^ | v • Reply • Share >



Niyazi G • 7 years ago

I feel like having aspx as view is limiting and want to use user controls as the view instead. But not sure how to implement multiple user controls in this pattern especially one user control changing state of the other or how to redirect to another view etc. would you provide some insight for this?

^ | v • Reply • Share >



Jeremy D. Miller • 7 years ago  
@AndyT,

Dude, you're making me work here! From your questions I'd say that you're working on a more interesting application than mine, but the grass is always greener somewhere else.

"Does the app controller purely control navigation between forms?"

For the most part, yes. That post is either next or the one after.

"What about the scenario where the form is one part of a (maybe transacted) workflow, say one step in a wizard? In this situation, should there be a use-case controller or equivalent? Where would this fit into the design?"

You could do that, or you could run the workflow as a double linked list so each presenter "knows" who's next and/or before and tells ApplicationController to go somewhere. You certainly don't want the ApplicationController to become a catch all for any piece of code, so a use case "Wizard" controller sounds fine to me. I'd have to change my normal ApplicationController design a bit to accommodate that I'd imagine.

"What about use-case-specific validation rules? Should the use-case controller handle this? Or should the application create use-case specific domain objects which inherit from the real domain objects?"

I'd say that if it's possible you try to fit the validation in the domain classes first, even if is logic specific to a use case.

If you'll thumb through the 1st or 2nd chapter in Fowler's PEAA book, he talks a bit about the different ways you can organize domain logic. When he mentions use case controller there's a danger of duplicating logic because of how coarse-grained use case controllers can be. I'd try to watch domain logic leaking into the use case controllers to prevent possible duplication.

^ | v • Reply • Share >



AndyT • 7 years ago

Great blog thread, with some great comments from your readers, too.

You said "In a typical Model View Presenter architecture the screens are composed of 4 basic types of classes ("M", "V", "P", and the Service Layer)." In previous sections, you mentioned the application controller. How does this fit into your design? Does the app controller purely control navigation between forms?

What about the scenario where the form is one part of a (maybe transacted) workflow, say one step in a wizard? In this situation, should there be a use-case controller or equivalent? Where would this fit into the design?

I agree that the domain objects should be responsible for validation. However, is this limited to generic validation rules that apply in all scenarios? What about use-case-specific validation rules? Should the use-case controller handle this? Or should the application create use-case specific domain objects which inherit from the real domain objects?

^ | v • Reply • Share >



SteveJ • 7 years ago

Some of the responsibilities going into a model would seem to be the same sort of things I would put in an object called a cache. The cache updates itself with data (when given from a service layer or wherever). It then serves that data out to clients when asked for it. So if a client/presenter wants a filtered set of data, the cache does the filtering (without changing the underlying data). If a calculation needs run, the cache is the guy who does it, since it's his data being crunched. In cases where the cache seems to take on more operations than just data retrieval (calculations, etc) I wrap the cache with an object I call the model, but it's really just running operations on the cached data (and it is usually too touchy with the cache, which seems wrong to me (doCalculation(cache.getA(), cache.getB()));

This is a little different from validation rules to me...I would use a model for those, but if the operation in question is basically a utility function (sort, sum, filter) then I leave it on my cache class.

Is this approach only practical because of my client-server architecture? Or because my cache is operating on domain objects (DTOs)?

The other difference I'm seeing is that my cache isn't really allowed to change it's underlying data to service a view. I might use a model to do that, but I wonder who's job it is in this pattern to send changes back to the server...

^ | v • Reply • Share >



Jeremy D. Miller • 7 years ago

Jeremy,

Where does data access go? Nowhere near the user interface code. The persistence, be it direct ADO.Net access or web services, is somewhere behind the Service Layer. I strongly prefer a Data Mapper approach that makes the Model classes completely Persistence Ignorant.

I guess you could technically use an Active Record approach that would put the data access stuff into the Model objects, but that would lock you hard into a 2 tier model. I could be wrong, but I'd expect most WinForms applications communicate with the backend through web services or remoting, so that option doesn't make a lot of sense to me. Plus the fact that I hate that approach in general.

Oh, and the no ORM thing? Life is entirely too short to spend human being time on writing ADO.Net code by hand in the majority of systems. That's like sending people out to do the harvest with scythes.

^ | v • Reply • Share >



Jeremy Collins • 7 years ago

I'm loving these series of articles. One thing I always struggle with is "where does the data access code go?"

Basically, which module / layer knows about System.Data.Sql?

Does the "service" layer retrieve and persist collections of domain objects? Or does the model layer handle all of its own persistence?

[Home](#)[About](#)[CodeBetter Cl](#)[Community](#)[Editors](#)[Search Results](#)[Friends of CodeBetter.Com](#)[Red-Gate Tools For SQL and .NET](#)[Telerik .NET Tools](#)[JetBrains - ReSharper](#)[Beyond Compare](#)[NDepend](#)[Ruby In Steel](#)[SlickEdit](#)[SmartInspect .NET Logging](#)[NGEDIT: ViEmu and Codekana](#)[DevExpress](#)[NHibernate Profiler](#)[Unfuddle](#)[Balsamiq Mockups](#)[Scrumy](#)[Umbraco](#)[NServiceBus](#)[RavenDb](#)[Web Sequence Diagrams](#)[Ducksboard](#)

CodeBetter.Com © '14

Stuff you need to Code Better!

Proudly powered by [WordPress](#).