Model-View-Presenter (MVP)

MVP is an architecture that first appeared in IBM and more visibly at Taligent during the 1990's. It's most commonly referred via the Potel paper. The idea was further popularized and described by the developers of Dolphin Smalltalk. As we'll see the two descriptions don't entirely mesh but the basic idea underneath it has become popular.

To approach MVP I find it helpful to think about a significant mismatch between two strands of UI thinking. On the one hand is the Forms and Controller architecture which was the mainstream approach to UI design, on the other is MVC and its derivatives. The Forms and Controls model provides a design that is easy to understand and makes a good separation between reusable widgets and application specific code. What it lacks, and MVC has so strongly, is Separated Presentation and indeed the context of programming using a Domain Model. I see MVP as a step towards uniting these streams, trying to take the best from each.

The first element of Potel is to treat the view as a structure of widgets, widgets that correspond to the controls of the Forms and Controls model and remove any view/controller separation. The view of MVP is a structure of these widgets. It doesn't contain any behavior that describes how the widgets react to user interaction.

The active reaction to user acts lives in a separate presenter object. The fundamental handlers for user gestures still exist in the widgets, but these handlers merely pass control to the presenter.

The presenter then decides how to react to the event. Potel discusses this interaction primarily in terms of actions on the model, which it does by a system of commands and selections. A useful thing to highlight here is the approach of packaging all the edits to the model in a command - this provides a good foundation for providing undo/redo behavior.

As the Presenter updates the model, the view is updated through the same Observer Synchronization approach that MVC uses.

The Dolphin description is similar. Again the main similarity is the presence of the presenter. In the Dolphin description there isn't the structure of the presenter acting on the model through commands and selections. There is also explicit discussion of the presenter manipulating the view directly. Potel doesn't talk about whether presenters should do this or not, but for Dolphin this ability was essential to overcoming the kind of flaw in Application Model that made it awkward for me to color the text in the variation field.

One of the variations in thinking about MVP is the degree to which the presenter controls the widgets in the view. On one hand there is the case where all view logic is left in the view and the presenter doesn't get involved in deciding how to render the model. This style is the one implied by Potel. The direction behind Bower and McGlashan was what I'm calling Supervising Controller, where the view handles a good deal of the view logic that can be described declaratively and the presenter then comes in to handle more complex cases.

You can also move all the way to having the presenter do all the manipulation of the widgets. This style, which I call Passive View isn't part of the original descriptions of MVP but got developed as people explored testability issues. I'm going to talk about that style later, but that style is one of the flavors of MVP.

Before I contrast MVP with what I've discussed before I should mention that both MVP papers here do this too - but not quite with the same interpretation I have. Potel implies that MVC controllers were overall coordinators - which isn't how I see them. Dolphin talks a lot about issues in MVC, but by MVC they mean the VisualWorks Application Model design rather than classic MVC that I've described (I don't blame them for that - trying to get information on classic MVC isn't easy now let alone then.)

So now it's time for some contrasts:

- Forms and Controls: MVP has a model and the presenter is expected to manipulate this model with Observer Synchronization then updating the view. Although direct access to the widgets is allowed, this should be in addition to using the model not the first choice.
- MVC: MVP uses a Supervising Controller to manipulate the model. Widgets hand off user gestures to the Supervising Controller. Widgets aren't separated into views and controllers. You can think of presenters as being like controllers but without the initial handling of the user gesture. However it's also important to note that presenters are typically at the form level, rather than the widget level - this is perhaps an even bigger difference.
- Application Model: Views hand off events to the presenter as they do to the application model. However the view may update itself directly from the domain model, the presenter doesn't act as a Presentation Model. Furthermore the presenter is welcome to directly access widgets for behaviors that don't fit into the Observer Synchronization.

There are obvious similarities between MVP presenters and MVC controllers, and presenters are a loose form of MVC controller. As a result a lot of designs will follow the MVP style but use 'controller' as a synonym for presenter. There's a reasonable argument for using controller generally when we are talking about handling user input.

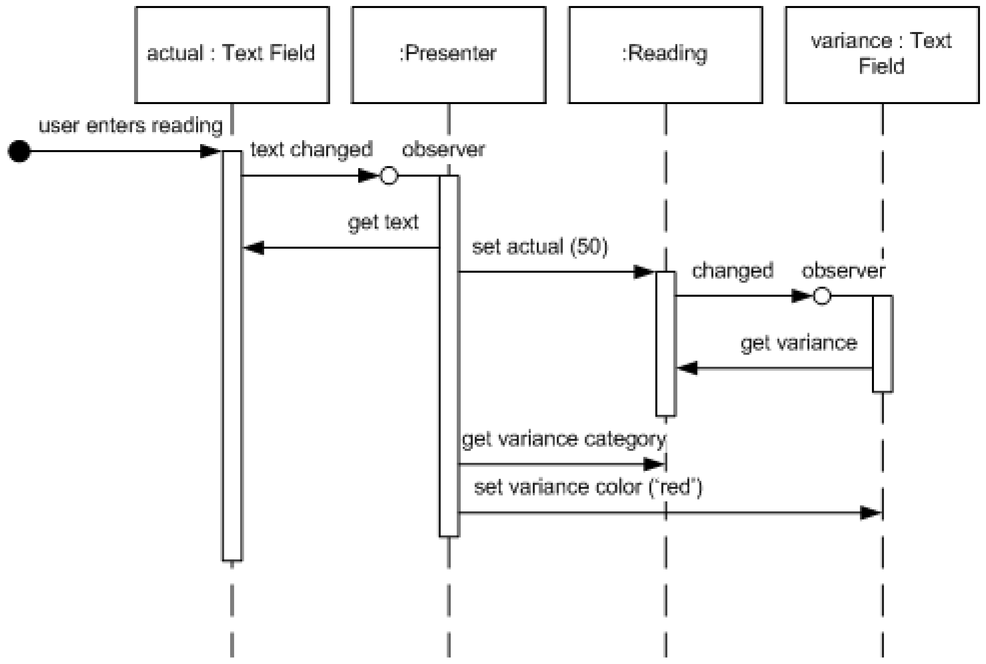

Figure 12: Sequence diagram of the actual reading update in MVP.

Let's look at an MVP (Supervising Controller) version of the ice-cream monitor (Figure 12). It starts much the same as the Forms and Controls version - the actual text field raises an event when its text is changed, the presenter listens to this event and gets the new value of the field. At this point the presenter updates the reading domain object, which the variance field observes and updates its text with. The last part is the setting of the color for the variance field, which is done by the presenter. It gets the category from the reading and then updates the color of the variance field.

Here are the MVP soundbites:

- User gestures are handed off by the widgets to a Supervising Controller.
- The presenter coordinates changes in a domain model.
- Different variants of MVP handle view updates differently. These vary from usingObserver Synchronization to having the presenter doing all the updates with a lot of ground in-between.