GarfieldTech

Technical thoughts, tutorials, and musings

Home

MVC vs. PAC

Submitted by Larry on 31 December 2006 - 4:42pm

One of the most common mistakes I see people make when talking about web architecture is with regards to MVC. Generally it comes down to a statement such as this:

> *"It's a web app, so we have to use MVC. That way we separate the logic and presentation, which means keeping PHP out of our display layer. All the important projects do it that way."*

Of course, such a statement is false. It demonstrates a lack of understanding about MVC, about web applications, about "important projects", and about software architecture in general. Let's try to clarify, with a little help from Wikipedia.

Modular design

Modular design is a generally recognized Good Thing(tm) in software engineering. As in science in general, breaking a problem down to smaller, bite-sized pieces makes it easier to solve. It also allows different people to solve different parts of the problem and still have it all work correctly in the end. Each component is then self-contained and, as long as the interface between different components remains constant, can be extended or even gutted and rewritten as needed without causing a chaotic mess.

I should note that modular design is not the same thing as plugin-based design, which is what many many open source projects use. Drupal "modules" are actually plugins. The Linux kernel, Apache, Eclipse, and many other high-profile projects with "loadable modules" are plugin-based architectures. Plugin-based architectures are not incompatible with modular design and in fact complement it well, but I digress...

Most software architectural patterns are based around the idea that modular design is a good thing, rather by definition. There are lots of architectural patterns for systems depending on what it is they're supposed to do. For interactive systems, the usual breakdown of components is "display component", "data storage component", and "business logic".

MVC

The most commonly-known interactive system architecture is Model-View-Controller, or MVC. Most good desktop applications use MVC or a variant of it, sometimes with the Controller partially merged into the View. In MVC, as the pretty picture at the other end of

that links shows, the Model holds data, the View is the part the user sees, and the Controller is an intermediary for business logic. Seems reasonable, right? Now take a closer look.

In MVC, the View component has direct access to the Model. The Controller itself doesn't enter the picture unless there is actual change to the data. Simply reading and displaying data is done entirely by the View component itself. As a result, a system can have many View components active at once, all reading and displaying data in a variety of different ways, even on different systems or in different languages or different modes (GUI vs. textual vs. web). On the flipside, however, that means the View component has to have some rather complex logic in it. It needs to know how to pull data out of the Model, which means it needs to know what the data structure is (or at least a rich API in front of the Model). It needs to be able to handle user interaction itself, with its own event loop.

Well, that rules out most would-be web-MVC setups. One of the most common cries of such systems is to "get database stuff away from HTML". Instead, everything is handled through a smart controller that uses a template layer to render and display output. That's not a bad design necessarily, but it is not MVC. If the display component does not have direct, random-access, pull-access to the data store, then it is not MVC.

PAC

A less publicized but still widely used architecture is Presentation-Abstraction-Control, or PAC. The two main differences between MVC and PAC are that in PAC the Presentation component is "dumb" while all the intelligence resides in the Controller and PAC is layered. Again, see the pretty picture.

You'll notice that the Presentation and Abstraction components never speak to each other. The Controller takes input, not the display component. The Controller has all the business logic and routing information. The Presentation component is essentially just a filter that takes raw data that the Controller pushes through it and renders it to HTML (or WML, or XML, or text, or an icon in a graphical monitoring system, or whatever). It's just a templating system.

The classic example of a PAC architecture is an air traffic control system. One PAC Agent takes input from a radar system about the location of an incoming 747, and uses the Presentation component to paint a picture of that blip on a canvas (screen). Another Agent independently takes input about a DC-10 that is taking off, and paints that blip to the canvas as well. Still another takes in weather data and paints clouds, while another tracks the incoming enemy bomber and paints a red blip instead. (Er, wait...)

The Web and Drupal

A web application doesn't map nicely to either MVC or PAC. There's no consistent event loop for the View component of MVC. For PAC the input is coming from the same source as the display, that is, in via the Presentation layer (kinda). Most web applications I've seen (and written) tend to use a sort of hybrid approach, which then gets (wrongly) called MVC. I blame Sun for that confusion, mostly, but there's enough blame to go around, too.

I'm going to pick on Drupal here because it's what I'm most familiar with, but this isn't a Drupal-specific observation. Drupal is very much a PAC architecture. In fact, it's a rather good PAC architecture. The menu system acts as the Controller. It accepts input via a single source (HTTP GET and POST), routes requests to the appropriate helper functions, pulls data out of the Abstraction (nodes and, in Drupal 5, forms), and then pushes it through a filter to get a Presentation of it (the theme system). It even has multiple, parallel PAC agents in the form of blocks that push data out to a common convas (page.tpl.php).

That's all well and good, but we need to remember, then, that Drupal is a PAC, not MVC framework. If Drupal were an MVC framework, then theme_* functions would have node_load() calls scattered throughout them. That's one reason why, for instance, Adrian's FAPI 3 plans scare me. He talks in his presentation about "a modified MVC but with the Model and View somewhat merged". Well, to start with Drupal isn't MVC in the first place. Secondly, the Model and View are the **last** pieces you want to merge. Merging a thin Controller into a thick View makes some sense, but merging a rich Model and rich View together makes a rich mess. That loses the separation of data and display that was the whole point in the first place.

Currently the closest Drupal comes to MVC is the Panels module. Panels allows the site admin to set up a custom layout with custom regions and then pull blocks, Views, or nodes into it in those regions. Currently that's the only real pull-based logic that Drupal supports for display given the active discouragement of database access (even via accessors like node_load()) in theme_functions and template files. Even that, however, is still limited to specific panel pages. There's no supported way to randomly pull a block into a node page, for instance. I know Earl "merlin" Miles is deeply engrossed in Panels 2.0, The Next Generation, but I don't believe it currently involves turning Panels inside out. He is, of course, welcome to correct me if he does. ;-)

There's also the Viewfield module for turning a View into a field of a node, but as cool a concept as that is it's still PAC thinking, or at best a very weird hybrid muddle.

Conclusion

There's more to say on this subject, but it's getting late so I will find a convenient stopping point for now. When thinking about your web applications, though, try not to fall back on the default "separation is good so we're using MVC which means separation" logic. Only the first part of that statement, "separation is good", is true. There are lots of ways to build a modular system, all of them appropriate in different places. MVC is not the only, nor even the most widely-used of them. If you aren't calling your data store directly from your display layer, in fact, you're not using it at all.

I'll be back next year with more thoughts on how to address the MVC vs. PAC question, and the impact they have on team development processes. Until then, Happy New Year and Happy Coding!

Web development
PHP