Action-Domain-Responder

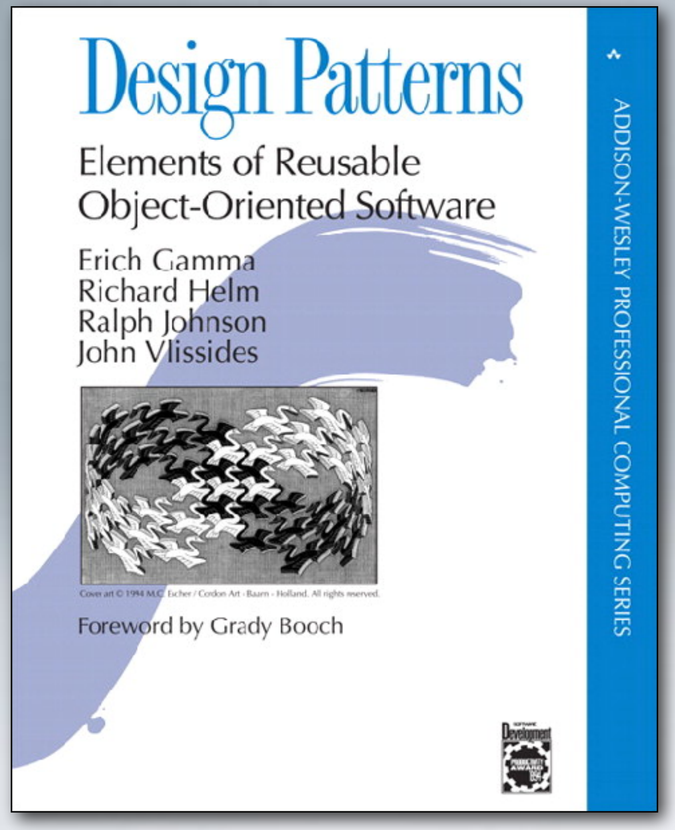
A Web-Specific Refinement of Model-View-Controller

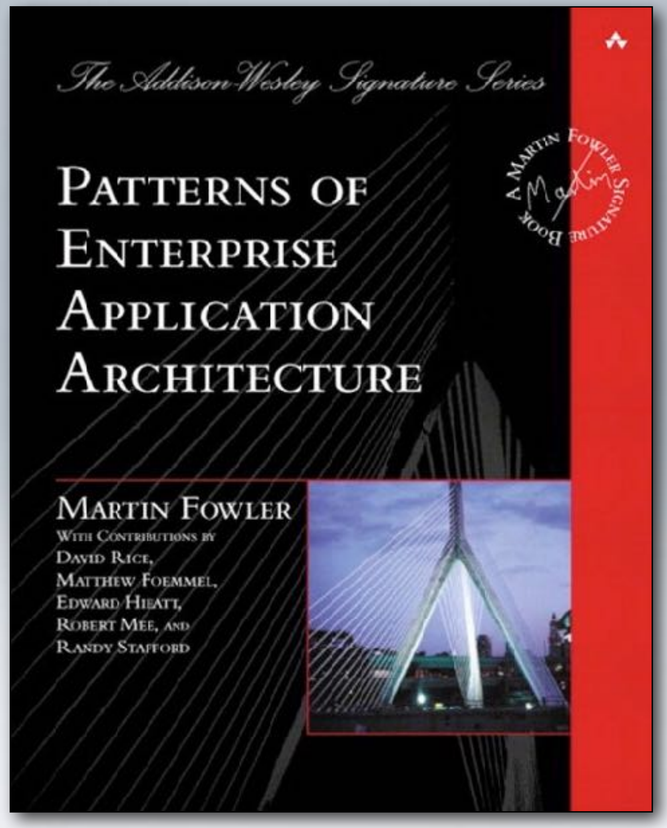
@pmjones

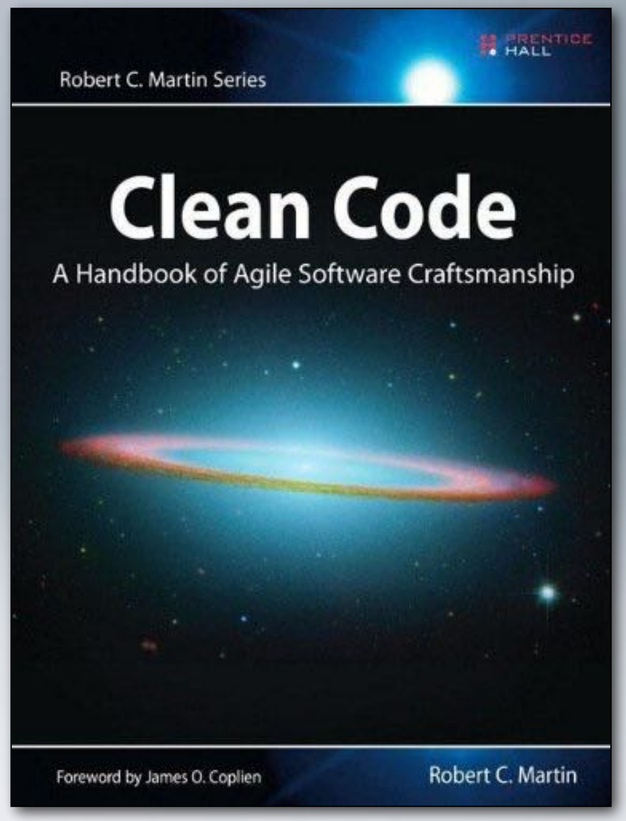
pmjones.io/adr

joind.in/15756

Read These

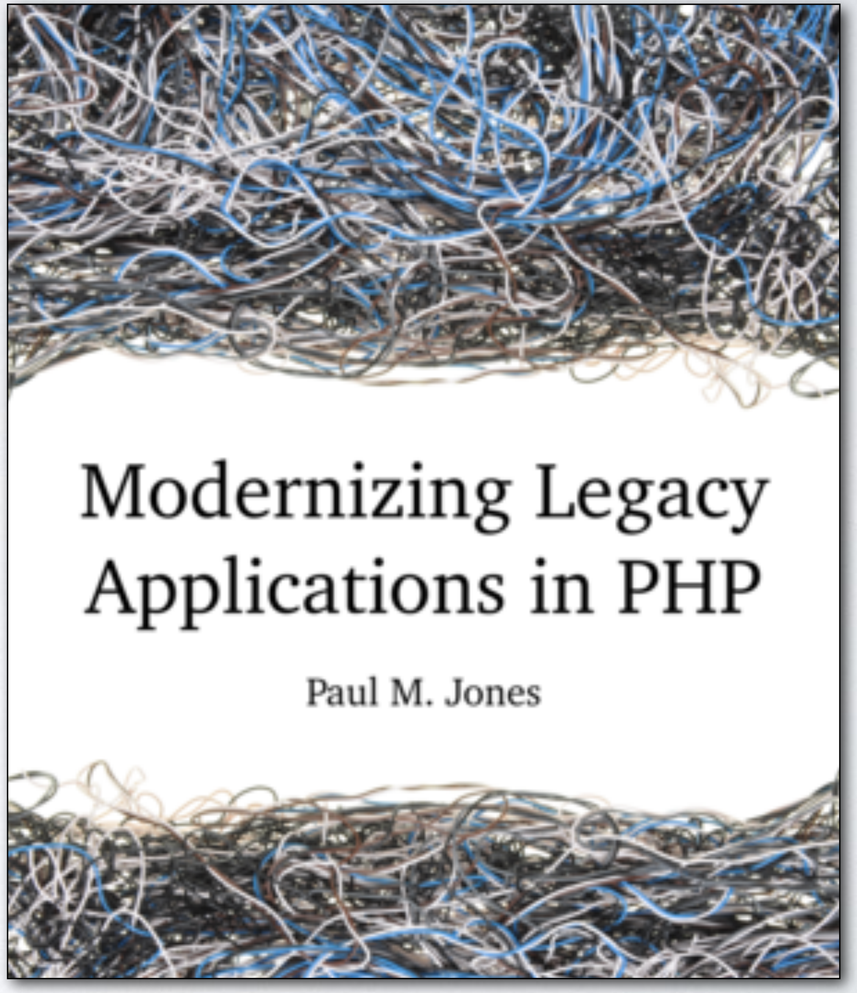

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

About Me

- 8 years USAF Intelligence
- BASIC in 1983, PHP since 1999
- Jr. Developer, VP Engineering
- Aura project, Zend FW, Relay for PSR-7
- ZCE Advisory Board
- PHP-FIG: PSR-1, PSR-2, PSR-4
- MLAPHP (http://mlaphp.com)



Overview

- How we think of MVC versus its desktop origins
- How ADR refines “MVC” for the web
- How ADR relates to existing “MVC” alternatives
- Address ADR comments and questions

“You Keep Using That Word …”

Server-Side MVC

- From all concerns combined (ball of mud) …
- … to separation of concerns (data source, domain logic, presentation)
- Oriented around HTTP (stateless shared-nothing request/response)
- Google for "mvc in (php | python | rails)"
- General consensus on components, not so much on collaborations

Server-Side MVC Collaborations

- User Agent sends a Request
- Router/Dispatcher invokes Controller
- Controller manages flow, invokes Model(s) that encapsulate domain
- Controller assigns values to View template
- Controller invokes View and sets into Response (with headers)

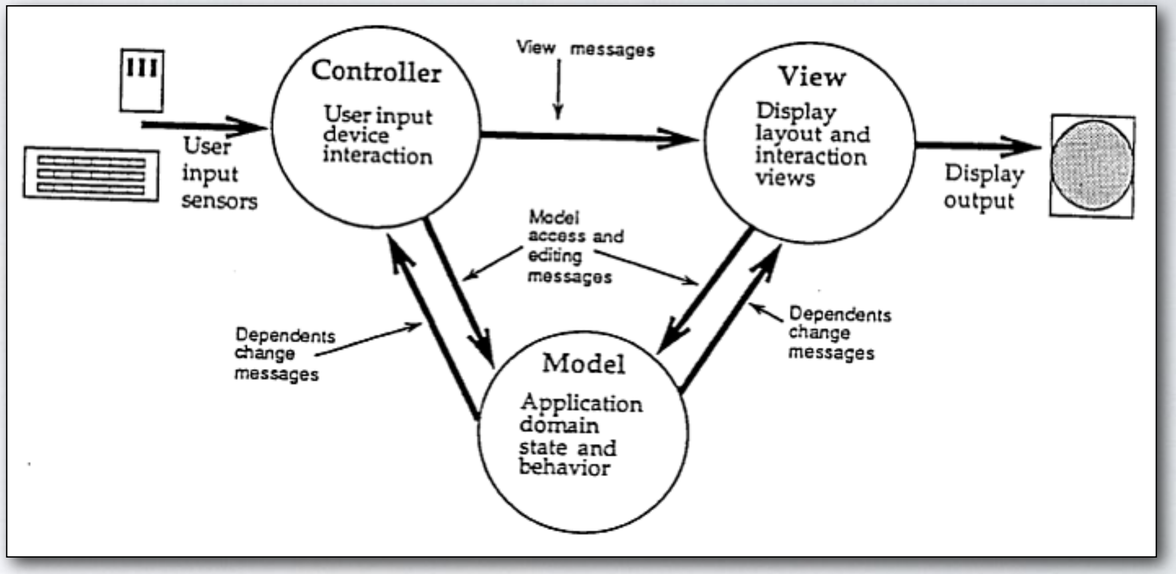
"… I Do Not Think It Means What You Think It Means."

Smalltalk-80 MVC

- Desktop graphical UI pattern
- Separation of concerns (input management, internal representation of domain, presentation to user)
- Interactive event-driven desktop (keyboard/mouse, in-memory)
- Google for "smalltalk mvc"

Smalltalk-80 MVC Collaborations

- Controller receives keyboard/mouse input events from User
- Controller notifies View and Model, respectively
- View and Model notify each other for updates
- View updates are rendered on the screen
- Hierarchical collection of interrelated MVC triads for each screen element (event system)



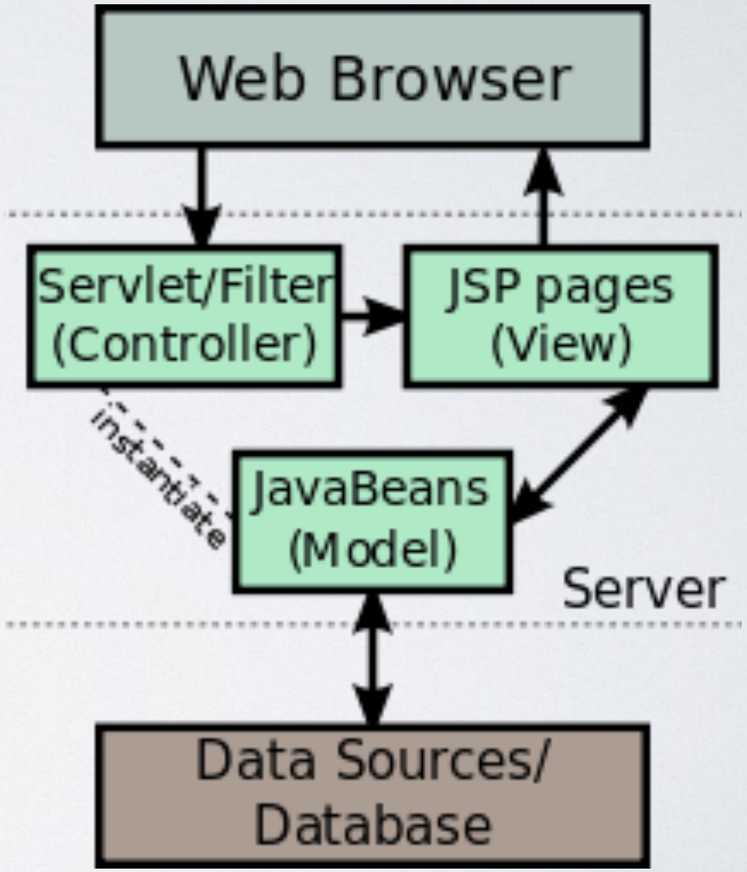
The ~~Pit~~ Web Of Despair

How To Describe Web UI Patterns?

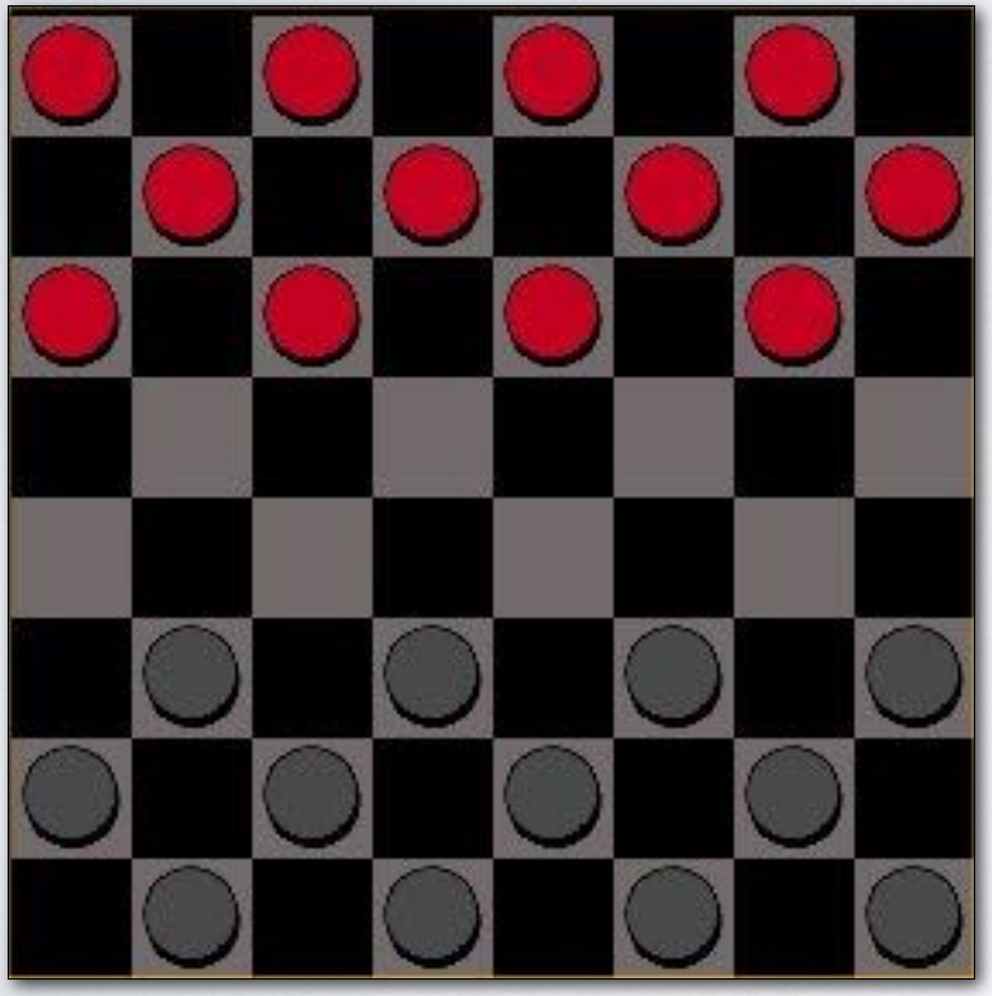
- Model 1: Sun JSP for page scripting (ball of mud)
- Model 2 : Servlet for action (Controller) and JSP for template (View)

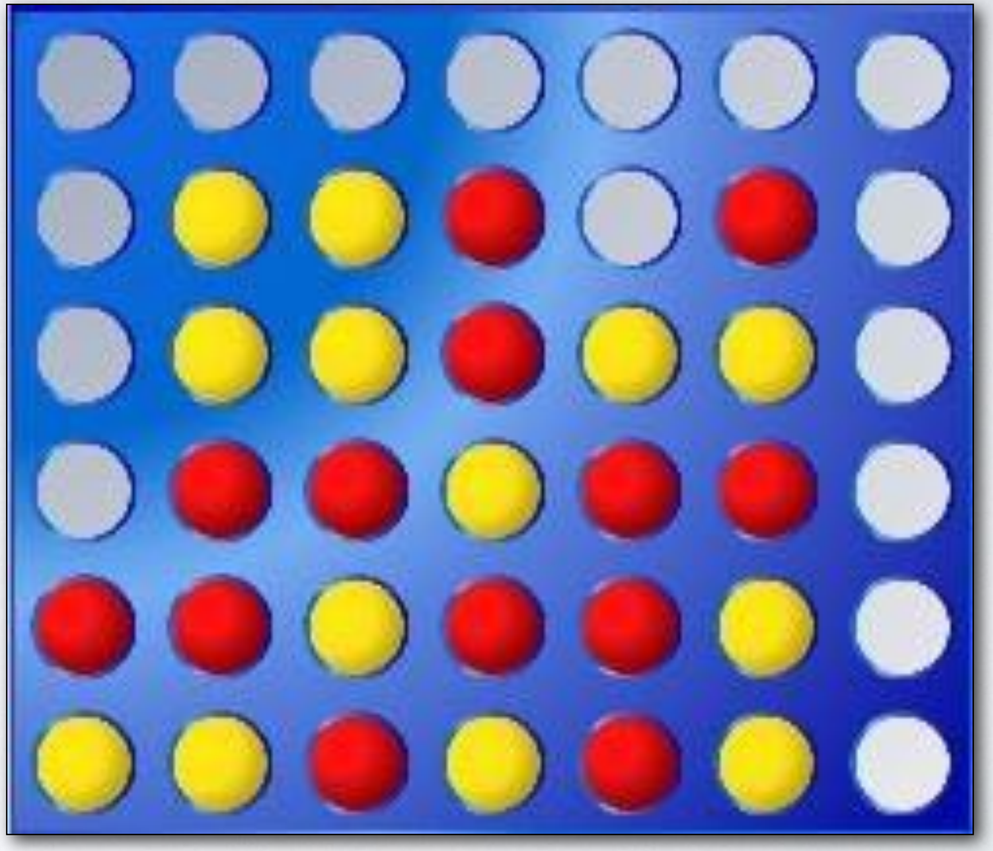
The Names Are The Same, But The Game Has Changed

- Used the name MVC for Model 2
- Subverted the collaborations
- From event-driven to request/response (pages)
- No more messaging interactions between triads
- One collected set of interactions delivered



Are These The Same?





“Dear God, What Is That Thing?”

- The pattern name “Model-View-Controller” has several meanings
- Hard to determine exactly what collaborations to expect
- Smalltalk-80 MVC relates more closely to client-side
- Model 2 MVC relates more closely to server-side
- Even more semantic diffusion since 2000:
- https://www.google.com/search?q=mvc+diagram

You're Trying To Kidnap What I've Rightfully Stolen

Toward A Web-Specific UI Pattern

- Stop using in-memory desktop GUI patterns as server patterns
- Entirely new name to break the association with "MVC"
- Remember we are in a client/server (request/response) environment
- Use existing server-side "MVC" as a basis
- Refine the components and collaborations toward better practices

Refining the "Model" to "Domain"

- The "Domain" has essentially identical responsibilities
- Reminiscent of "Domain Logic": TransactionScript, DomainModel, TableModule, ServiceLayer
- Reminiscent of "Domain Driven Design": Repository, App Service
- (ActiveRecord is categorized as a "Data Source" pattern)

Refining the "View" to "Responder"

- Usually think of a View system as templates (screen elements)
- Client receives HTTP response of **both** body **and** headers
- This means the View in server-based MVC is **not** the template
- The View in server-based MVC is the **Response**

Intermingled Presentation Logic

- Template Views generally build HTTP **body** values
- Remaining Controller logic manipulates HTTP **header** values
- Presentation logic is mixed between Views and Controllers
- Need a layer that is completely in charge of building the Response

“Responder” For Presentation

- Responder layer handles setting headers, status, etc
- Additionally uses templates for setting body content
- Invoke a Responder for presentation of Response

Using Responders In Controllers

- Remove Response presentation from all Controller action methods
- index(), create(), read(), update(), delete()
- Each action method has its own set of status codes and templates

One Responder Per Controller?

- One Responder per Controller to cover all possible action methods?
- No: inject one Responder per action method
- But: injecting Responders that might not be needed

Refining the “Controller” To “Action”

- Instead of a Controller with index(), create(), read(), etc. …
- … one class per Action: IndexAction, CreateAction, ReadAction, etc.
- Inject the individual Responder into the individual Action

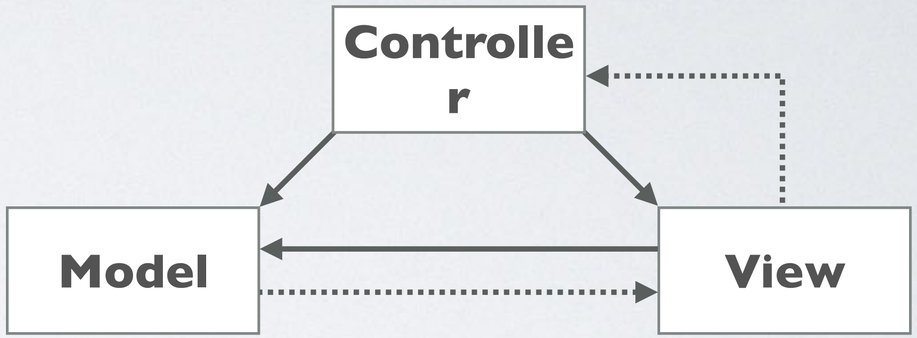
“Let me explain.
No, there is too much.
Let me sum up.”

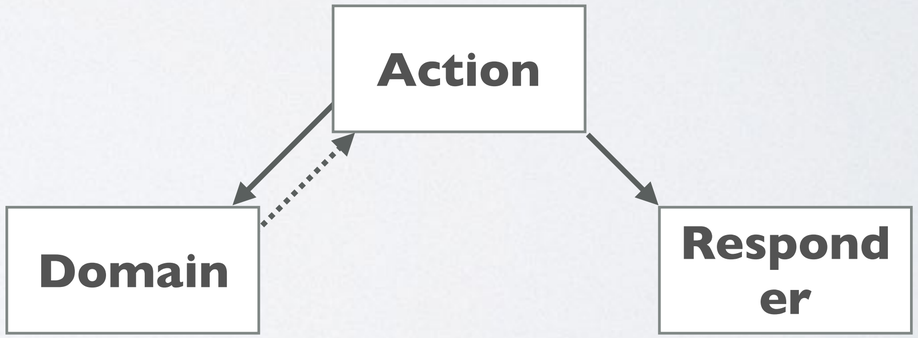
Components

- **Domain** is the logic to manipulate the domain, session, application, and environment data, modifying state and persistence as needed.
- **Responder** is the logic to build an HTTP response or response description. It deals with body content, templates and views, headers and cookies, status codes, and so on.
- **Action** is the logic that connects the *Domain* and *Responder*. It uses the request input to interact with the *Domain*, and passes the *Domain* output to the *Responder*.

Collaborations

- Action feeds input from HTTP request to a Domain layer
- Action feeds output from Domain layer to a Responder
- Responder builds the HTTP response headers and body





Code Examples

About The Examples

- Overly-simplified to highlight components and collaborations
- Action is minimalist, almost trivial (micro-framework-ish)
- Domain becomes much more robust
- Responder has to determine presentation based on Domain values

“I’m Not The Real Dread Pirate Roberts.”

Is ADR Just Another Pattern In A Mask?

- EBI
  (Entity-Boundary-Interactor)
- DCI
  (Data-Context-Interaction)
- MVP
  (Model-View-Presenter)
- MVVM
  (Model-View-ViewModel)
- PAC
  (Presentation-Abstraction-Control)
- RMR
  (Resource-Method-Representation)

Entity-Boundary-Interactor

At best, ADR maps only roughly to EBI:

- the ADR Action and Responder elements may represent a web-specific EBI Boundary

- the ADR Domain element may represent an EBI Interactor element, encapsulating or otherwise hiding the EBI Entity elements from the ADR Action.

Alternatively, in ports-and-adapters or hexagonal architecture terms, it may be reasonable to think of the Action as a "port" through which an EBI Boundary is invoked as part of the ADR Domain. Finally, the Responder could be seen as an "adapter" back through which the application data is returned.

Data-Context-Interaction

DCI is described as a complement to MVC, not a replacement for MVC. I think it is fair to call it a complement to ADR as well.

Model-View-Presenter (Supervising Controller, Passive View)

- Model and the Domain map closely, as they do in MVC.

- Passive View does not map well to either Action or Responder; it might better be regarded as the response that gets returned to the client.

- Supervising Controller might map to Responder, in that it "manipulate[s] the view to handle more complex view logic". However, Responder is not responsible for interacting with the Domain, and it does not receive the client input, so does not seem to be a good fit for Supervising Controller.

- Alternatively, Supervising Controller might map to Action, but the Action is not responsible for manipulating the view (i.e. the response).

Model-View-ViewModel

Maps only incompletely to ADR.

- *The Model in MVVM maps closely to the Model in MVC and the Domain in ADR.*

- *Similarly, the View in MVVM maps closely to the View in MVC and the Responder in ADR.*

- *However, the ViewModel does not map well to a Controller in MVC or an Action in ADR.*

Presentation-Abstraction-Control

PAC is used as a hierarchical structure of agents, each consisting of a triad of presentation, abstraction and control parts.

The agents (or triads) communicate with each other only through the control part of each triad.

It completely insulates the presentation (view in MVC) and the abstraction (model in MVC).

This provides the option to separately multithread the model and view which can give the user experience of very short program start times, as the user interface (presentation) can be shown before the abstraction has fully initialized.

Resource-Method-Representation

Resource <--> Domain
Method <--> Action
Representation <--> Responder

“A Resource can be thought of as an object with private variables and public methods that correspond to HTTP methods. From an MVC point of view, a resource can be thought of as a model with a bit of controller thrown in." -- Mixing of concerns.

"The Representation is like a view in MVC, we give it a resource object and tell it to serialize the data into it's output format." -- No allowance for other HTTP responses.

ADR might be considered an expanded or superset variation of RMR, one where a Resource and an action one can perform on it are cleanly separated into a Domain and an Action, and where the Representation (i.e., the building of the response) is handled by a Responder.

“I Would Not Say Such Things If I Were You!”

Criticisms

- Isn't that a lot of classes?
- Where does "feature" go in ADR?
- Can I use it with "language" ?
- Can I use it on the client?
- Omission of Request
- Omission of Front Controller
- The examples are too limiting
- You're forgetting "pattern"

“You’d Make A Wonderful Dread Pirate Roberts.”

Conclusion

- MVC originated for in-memory desktop user interfaces
- Did not translate to the web so well
- Explored ADR as a way to refine MVC specifically for the web
- Code, commentary, and criticism around ADR

Thanks!

- pmjones.io/adr (ADR Paper)
- github.com/radarphp/Radar.Project (ADR "framework")
- paul-m-jones.com and @pmjones
- joind.in/15756