

# Introduction to PureMVC

Paddy Keane

Creative Director

World Archipelago Internet

# PureMVC - Topics

- Background - 2 mins
- Why use a framework? - 15 mins
- Development Aspiration - 2 mins
- Why PureMVC? (Overview) - 10 mins
- Implementation - 10 mins
- Case Study - 20 mins  
(flex application startup with configuration for multilingual support )
- Resources
- Q & A

Why use a Framework?

# Why use a Framework?

- Frameworks came out of a need to address recurring issues in the development process. Notably giving **agility** and a way to handle:
  - **Changing feature**
  - Problems that arise that were not considered at the beginning
  - Extending, improving, re-factoring & maintaining your project
  - reuse of existing Code/Components
  - Risk Management - allows a project to be broken down into discrete elements so risk or **Uncertainty** can be handled in smaller, manageable chunks

# Why use a Framework?

This is achieved by helping you ‘**architecture**’ your projects into **3** tiers that are the most important aspects of your application:

- your data (**Model**)
  - your UI display (**View**)
  - Application logic that orchestrates your Data & UI (**Controller**)
- 
- **Why:** Eliminates the most harmful responsibility misplacement issues that hamper scalability and **maintainability**, i.e tightly coupling data to UI and UI’s to each other.

# Why use a Framework?

This separation gives a structure that enables better code **clarity**. An organized way to show your coding **intent**, which will help both yourself and other developers understand the applications scope

To use a Framework  
successfully you first need  
to have a good  
**Development Process**

# Features, Features, Features

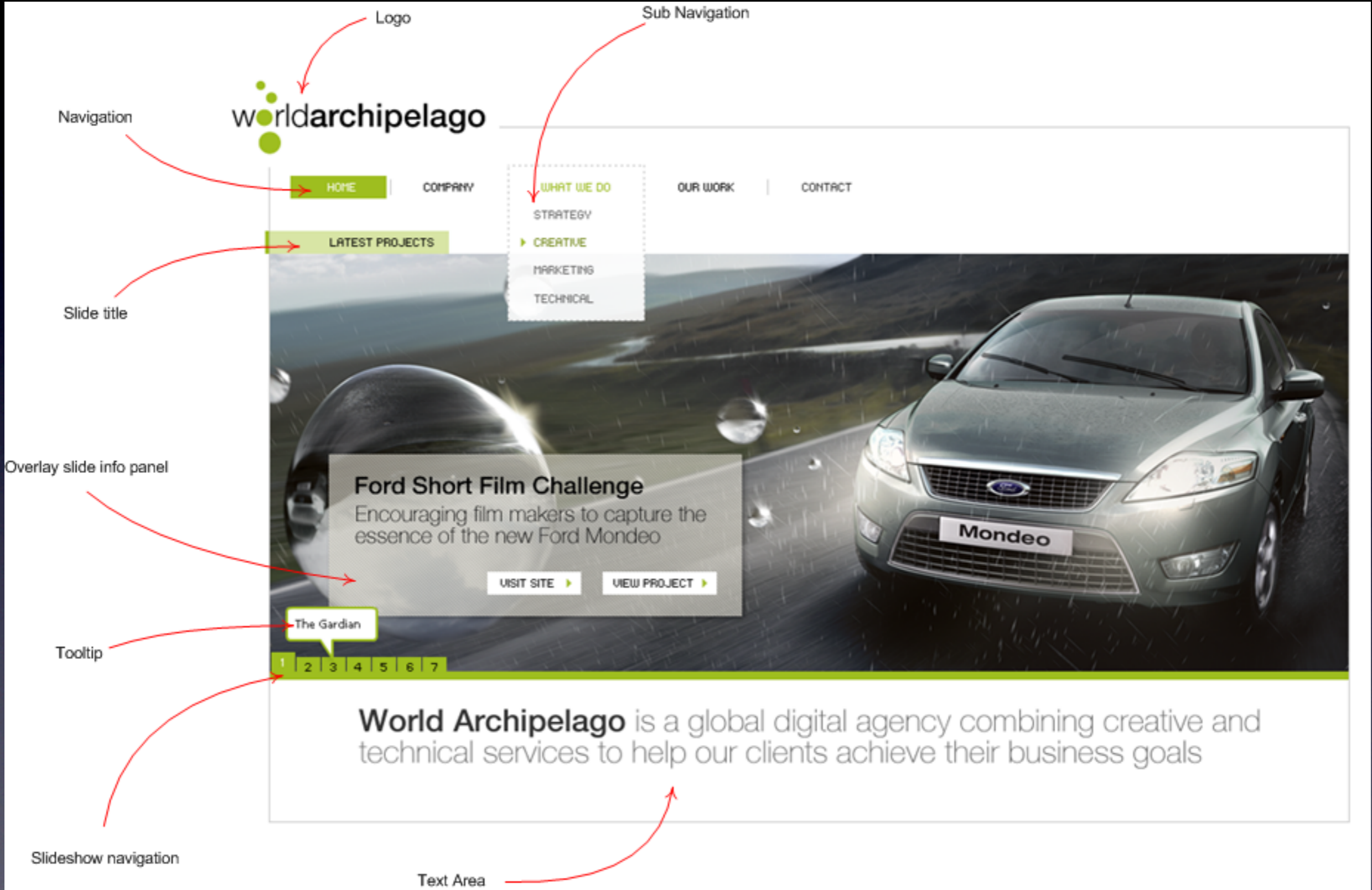
- get pen and paper and figure out a feature list with encapsulated, discrete elements
- Features can have granularity. i.e a feature can contain other features.
- This allows for prioritization.
- figure out how these features need to be implemented
  1. Front end UI features
  2. Backend 'data' features

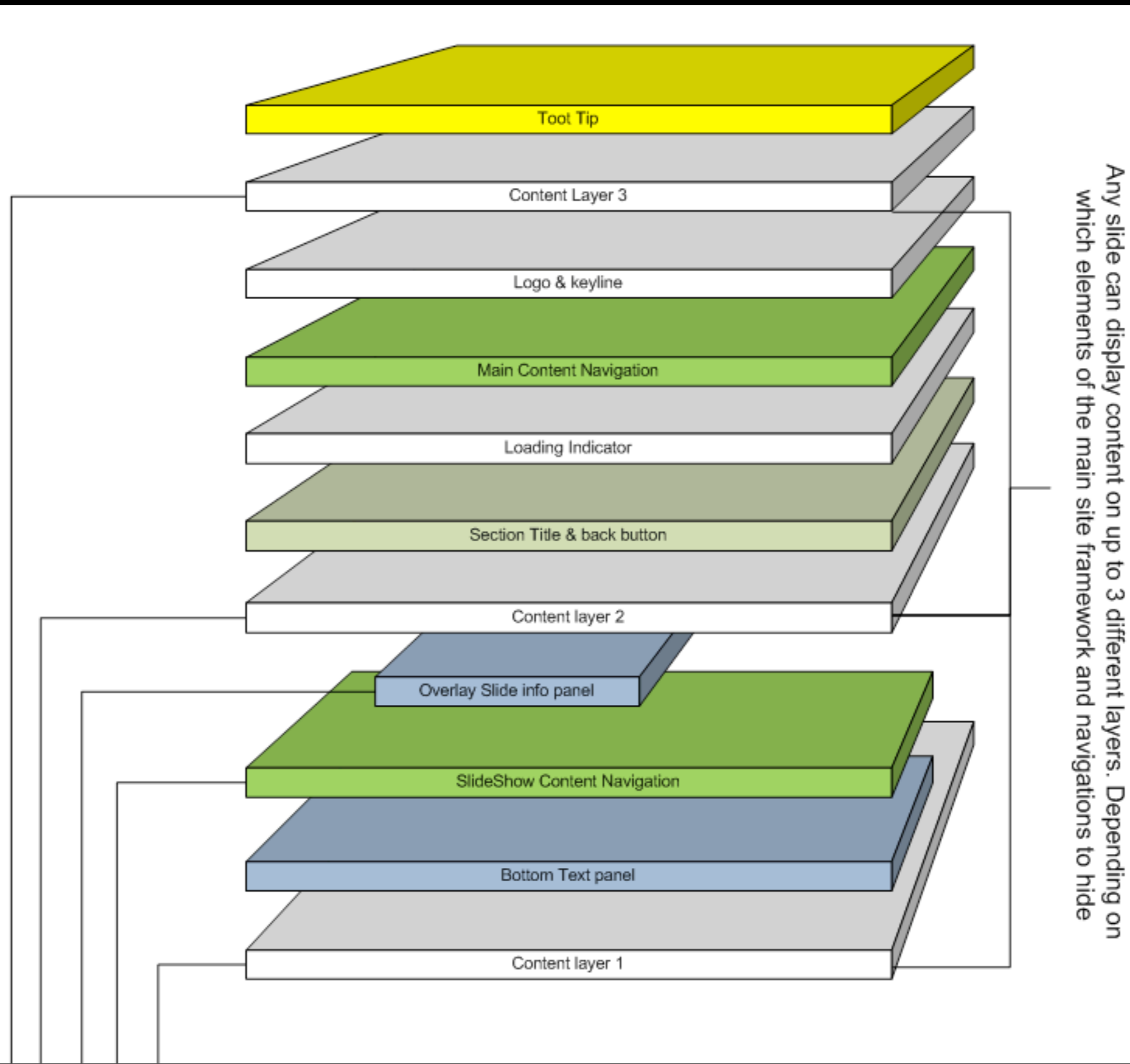
your application logic will then work to this malleable feature set

when you have a **Features List** you can then go about mocking up and doing case studies of individual features. Testing them, refining them if needed.

Once basic features are in place you can do some broad brush stroke application architecturing

# Case Study





Any slide can display content on up to 3 different layers. Depending on which elements of the main site framework and navigations to hide

# Development Aspiration

# Development Aspiration

- Agile development
- UCD (user centric development)
- User experience
- SOA (service orientated architecture)
- Risk management
- Elegant solutions
- Simplicity
- ....And more

# Development Aspiration

Grace or **Gracefulness**

# Development Aspiration

## {Define: Grace}

- Elegance and beauty of moment or expression
- seemliness
- A sense of propriety
- Consideration to others

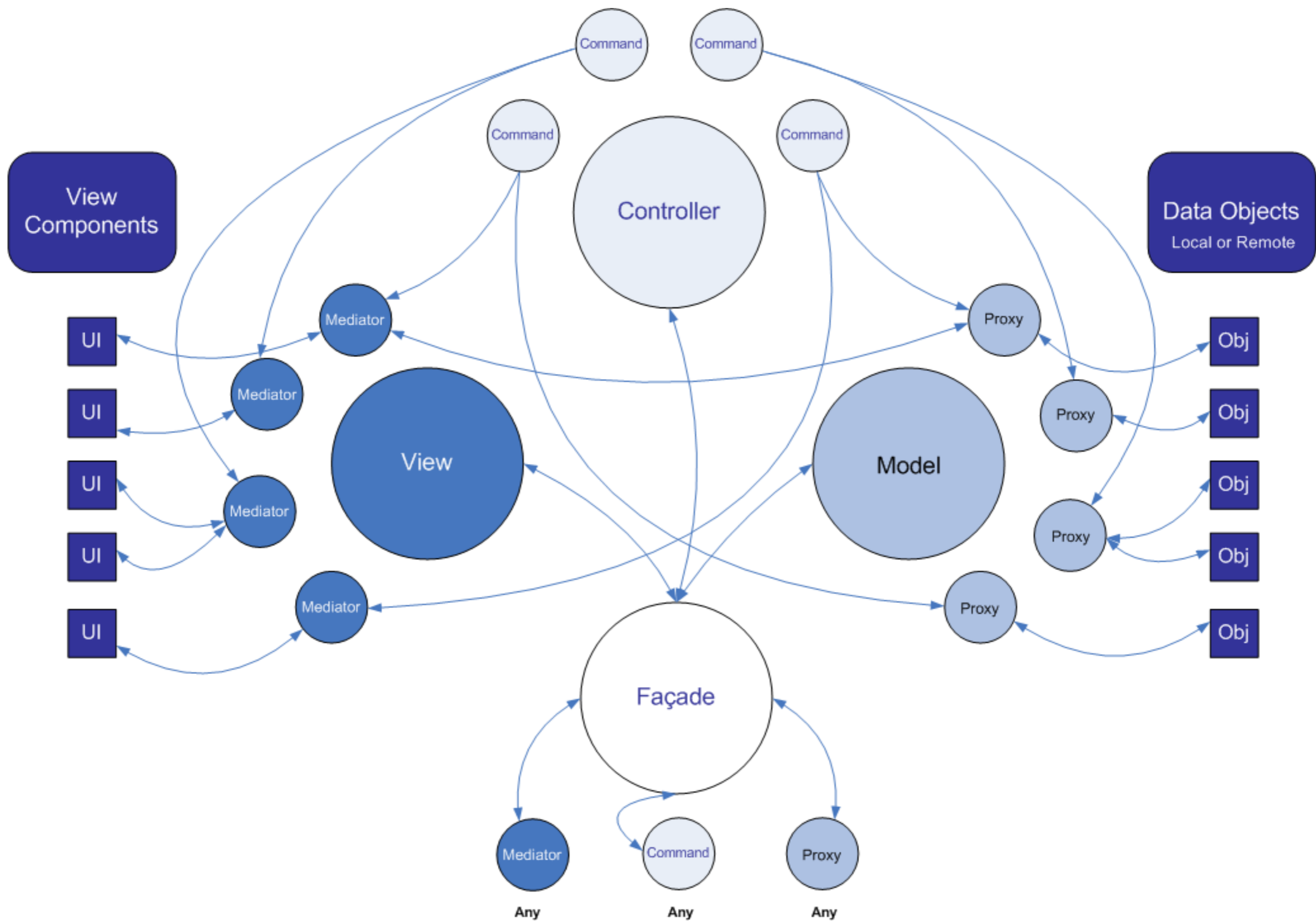
# PureMVC

# PureMVC - Framework Goals

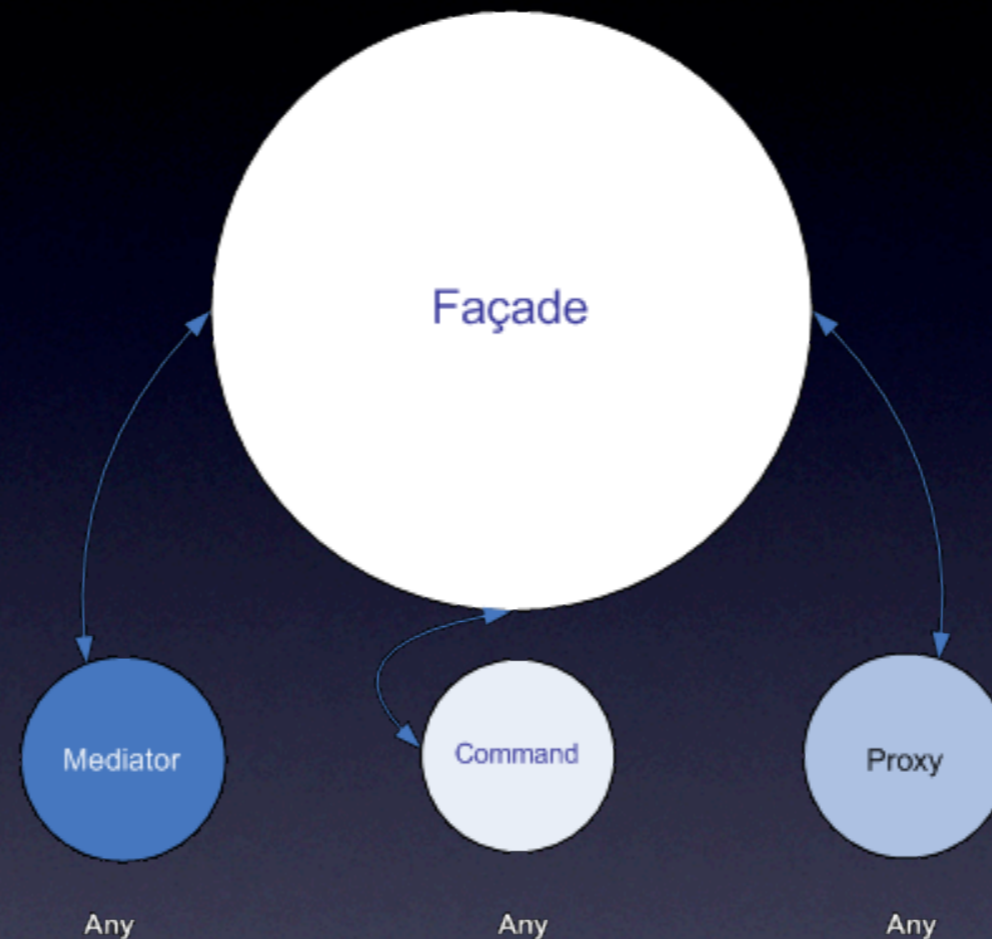
- Simple but appropriate framework scope
- Reduce confusion over application layers as well as class roles, responsibilities and collaborations
- Balance speedy implementation with scalability and maintainability
- Avoid dependencies - Language independent; AS3, AS2, C#, Java, PHP and ColdFusion. This puts PureMVC on Flex, Flash, AIR, FlashLite, .NET, Windows Mobile, Silverlight, J2ME, SE, EE, JavaFX, PHP, Coldfusion
- Hide complexity from the developer

# PureMVC - Framework Benefits

- **Loosely-coupled Architecture**
  - ▶ 'Publish/Subscribe' - observer notification style
- **Programmed to Interfaces**
  - ▶ Supports extensibility via sub-classing or interface implementation.
- **Useful Base Implementation Classes**
  - ▶ Out-of-box functionality that requires very little sub-classing, or even direct interaction with core Framework actors (Model, View, Control)
- **Build on Proven Design Patterns**
  - ▶ All patterns described in the classic 'Gang of Four' book, Design Patterns - Elements of Reusable Object Oriented Software



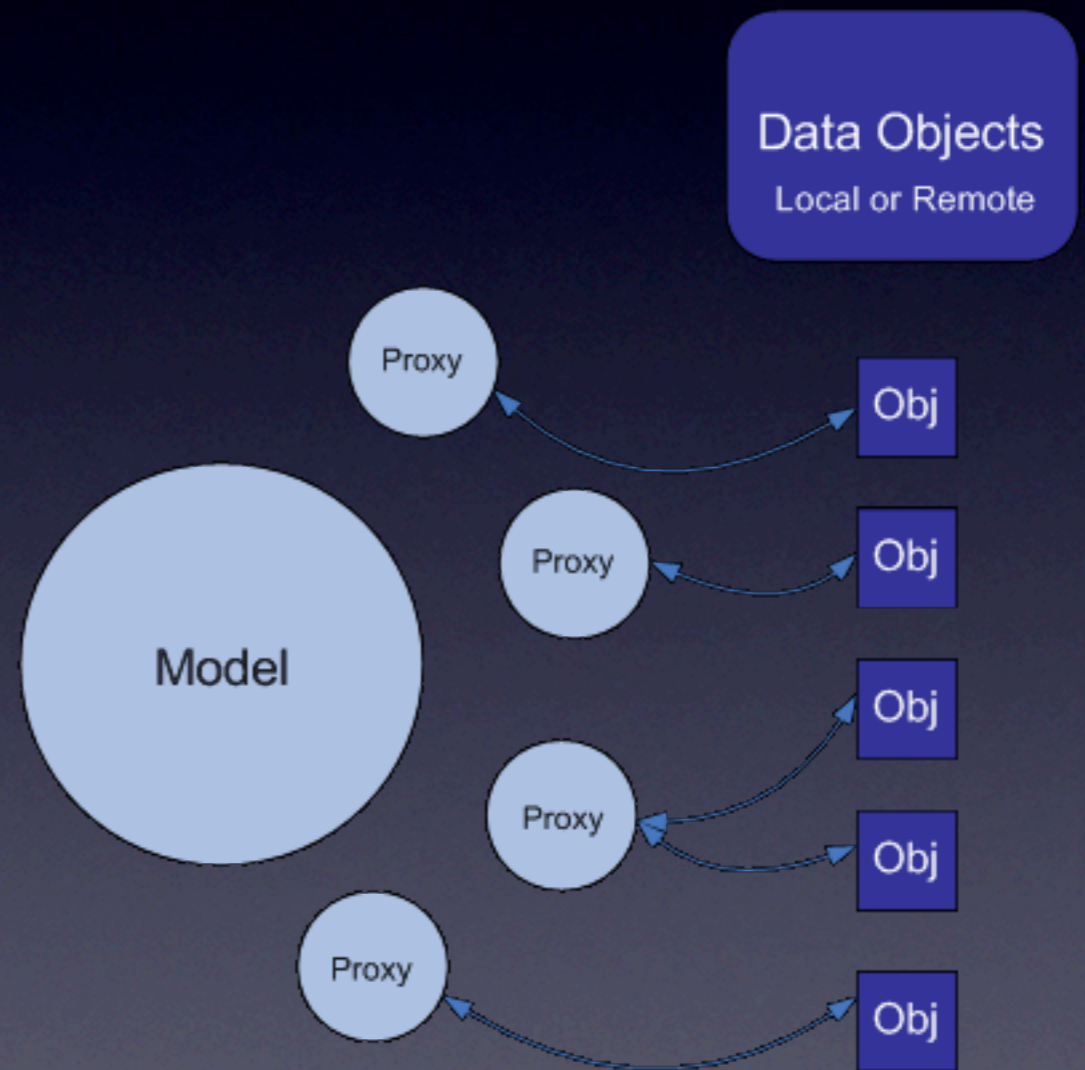
# Façade



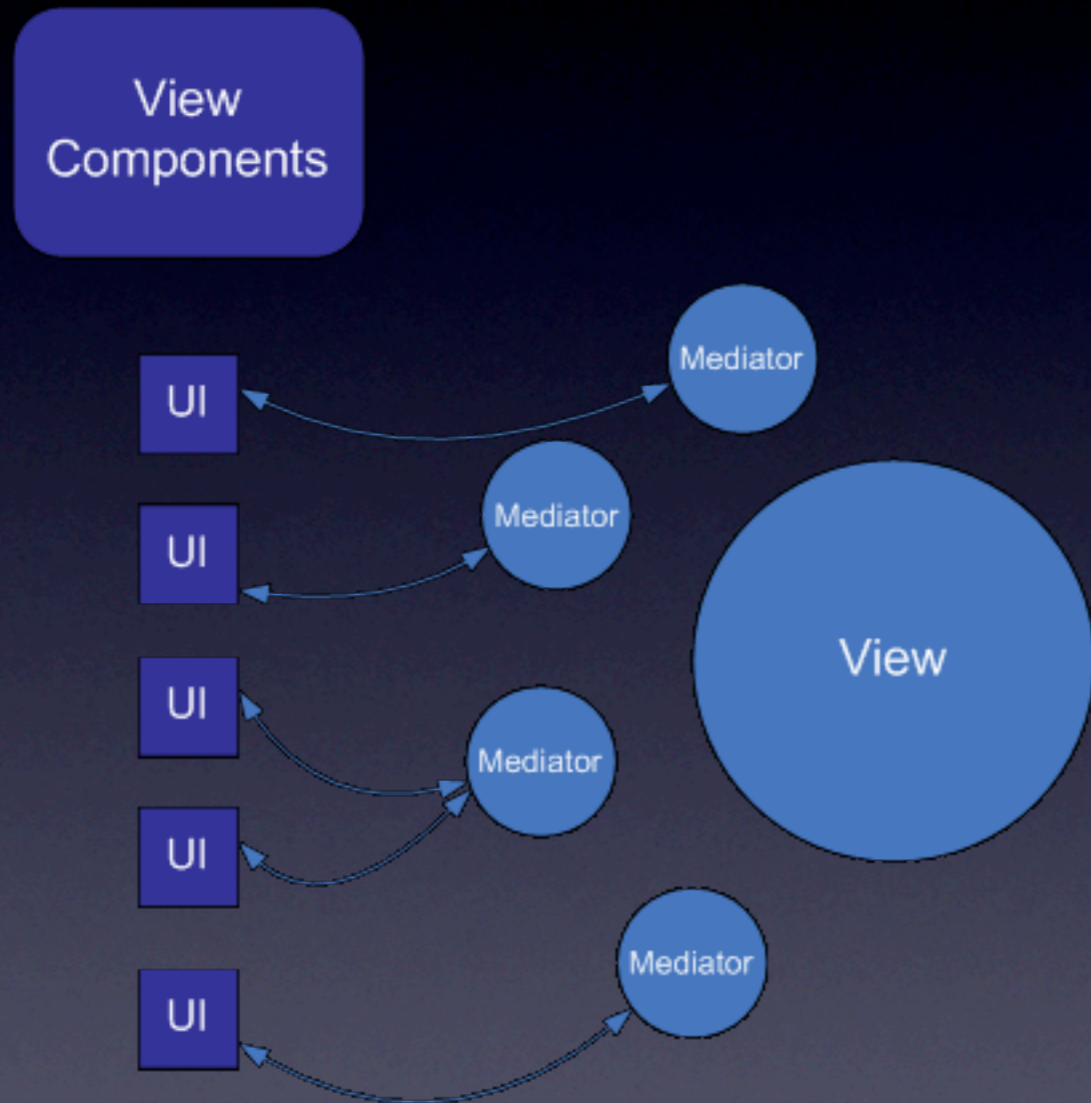
- Façade provides a single interface for accessing the framework
- Proxys, Mediators & Commands have access to it
- You generally extend it for your specific application interests
- Hides the Model, View & Controller classes so you don't have to work with them directly

# Model (Proxies & Data objects)

- **Data Objects** can be simple data holders, such as common DTO's (Data Transfer Objects)
- **Proxies** control access to Data Objects, they query, format and server data
- ... or they can use the **Delegate** pattern to access remote data (not to be confused with remoting)
- This allows Data Objects to know absolutely nothing about PureMVC

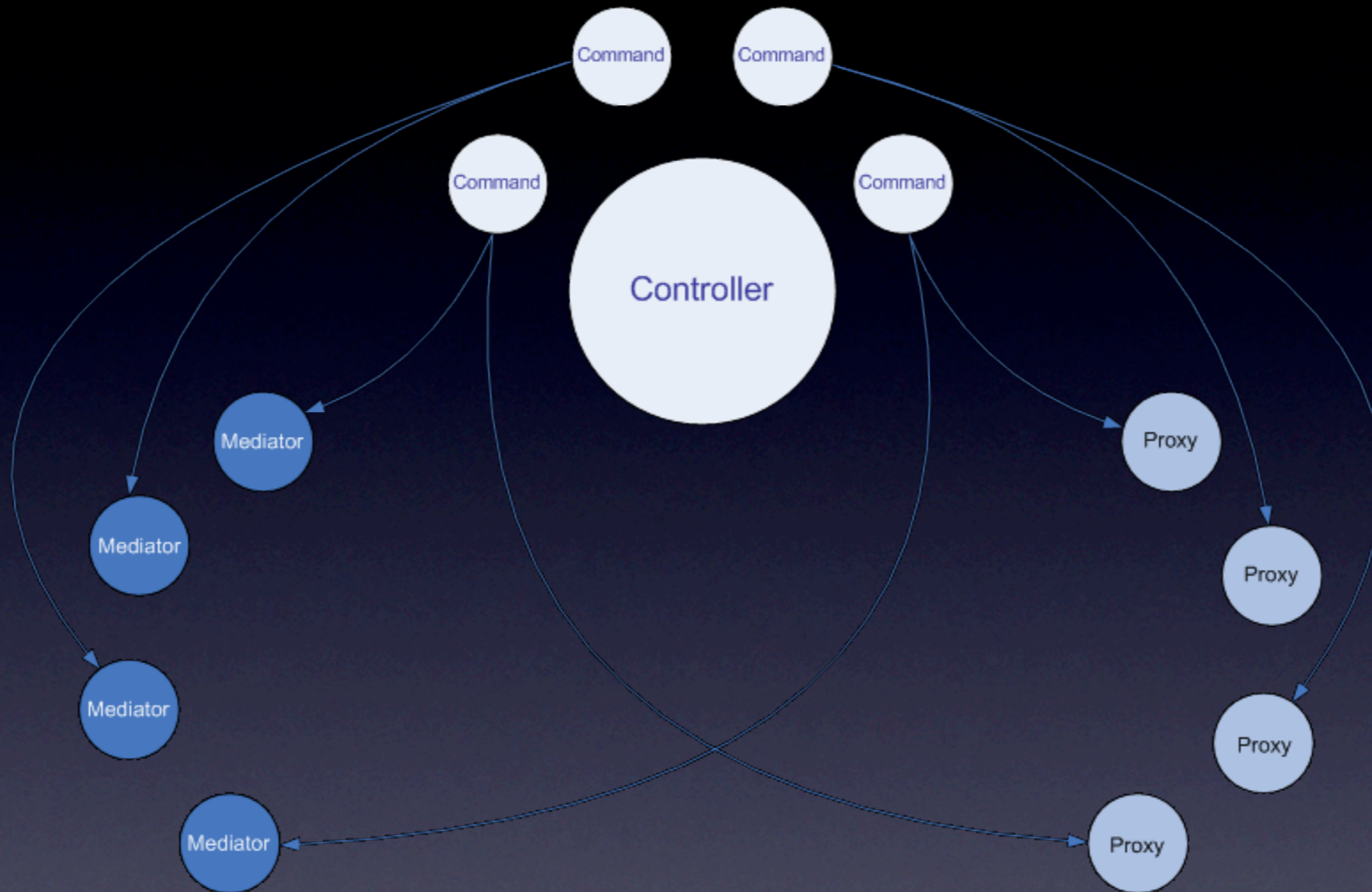


# View (Mediators & UI components)



- **UI Components** are standard components like DataGrids, component groups held in a Form or bespoke, custom components.
- **Mediators** steward UI components, listening for their events, inspecting & setting their properties.
- Mediators interact with the system on behalf of the UI Components
- This allows UI Components to know absolutely nothing about PureMVC

# Controller (Commands)



- **Commands** House the *Business Logic* of the application, Coordinating/ Orchestrating complex or system-wide activities. i.e startup and shutdown
- A Command may retrieve and interact with *Proxies*, communicate with *Mediators* or execute other *Commands*.

# Notifications

**Notifications** work in conjunction with, not in place of Events.



**Proxies** - May broadcast, but do **not** listen for *Notifications*



**Mediators** - Listen for and may broadcast *Notifications*



**Commands** - Are triggered by and may broadcast *Notifications*

# Implementation

# Implementation

I.

- ▶ **Architecture Data** *(figure out your script)*
  - UML diagramming can come in handy hear
  - Create your data Objects based on features required. i.e data sets that are discrete and easy to maintain and query
  - make some dummy data, XML
  - Figure out as many configuration parameters needed (colours, label names, URIs of remote resources, services...

# Implementation

2.

## ▶ Wireframe UI Components

- Figure out what will be a discrete encapsulated component set.
- Have first stab at API based on feature list.

Public properties that can be queried (getters/setters)

Known Inputs - functions to inject data so UI can be decorated.

Known Outputs - events names it will broadcast

- make case studies, test and develop these components to the minimum feature set required.

# Implementation

## 3. begin PureMVC

### ▶ Extend Façade, creating a Concrete Façade

- Typically called ApplicationFacade, but can be called anything
- Define constants for Notifications names, Since all actors communicate via your concrete façade
- Initialize Controller with Notification to Command mappings

# Implementation

4.

## ▶ Write a few Commands

- Handle Business logic, ensuring that use-cases of the application are executed properly
- Commands may create and register Proxies and Mediators
- May retrieve and act upon Proxies
- May send Notifications to be responded to by Mediators
- May trigger or execute other Commands
- Will be executed by Controller when mapped Notification is broadcast

# Implementation

5. create your Proxies

# Implementation

6. create your Mediators

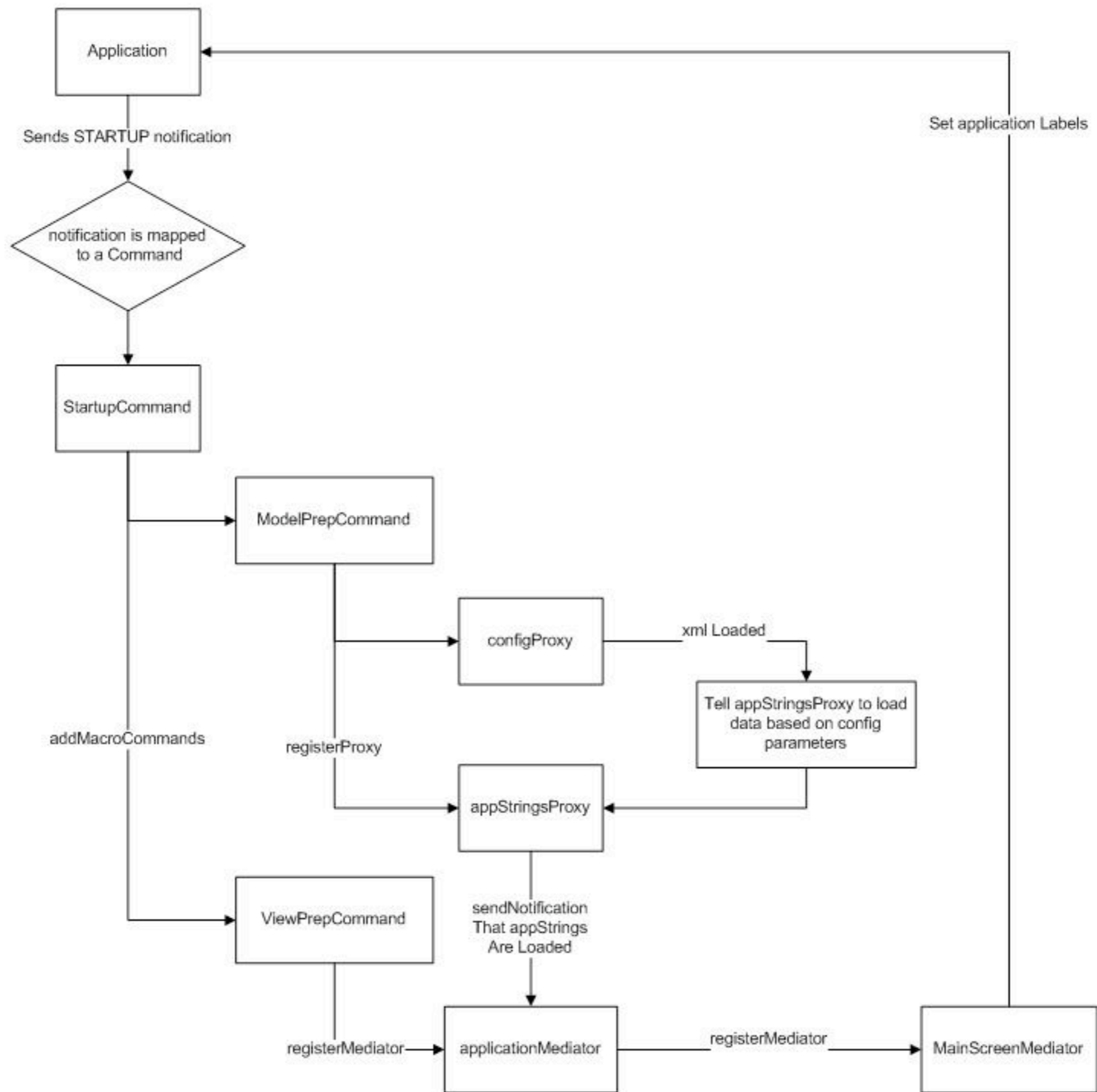
# Implementation

7. Appraisal - then back to point 4. Make more commands, hone UI components and their Mediators, improve Proxies, Data Objects. Add new functionality....

# Implementation

8. Sprinkle with sugar and server

# Case Study



# Resources

<http://pureMVC.org> - great examples, docs & forum

<http://algorithmist.wordpress.com/2008/01/22/puremvc-links>

Read the Best practices Doc, then read it again

Thank you Cliff  
for PureMVC