

Using the ASP.NET MVC Framework

Dave Sussman



<http://www.flickr.com/photos/cpeachok/16438999/>

Why Another Framework?

- The Web is Stateless
 - ASP.NET isn't
- ViewState Fragility
 - Dynamic controls, Ajax
- Postback Architecture
 - False construct
- Testing
 - UI code

Key Goals

- Separation of concerns
- Enable easier testing
- Highly extensible and pluggable
- Powerful URL mapping
- Support existing page types as views
- Support existing functionality

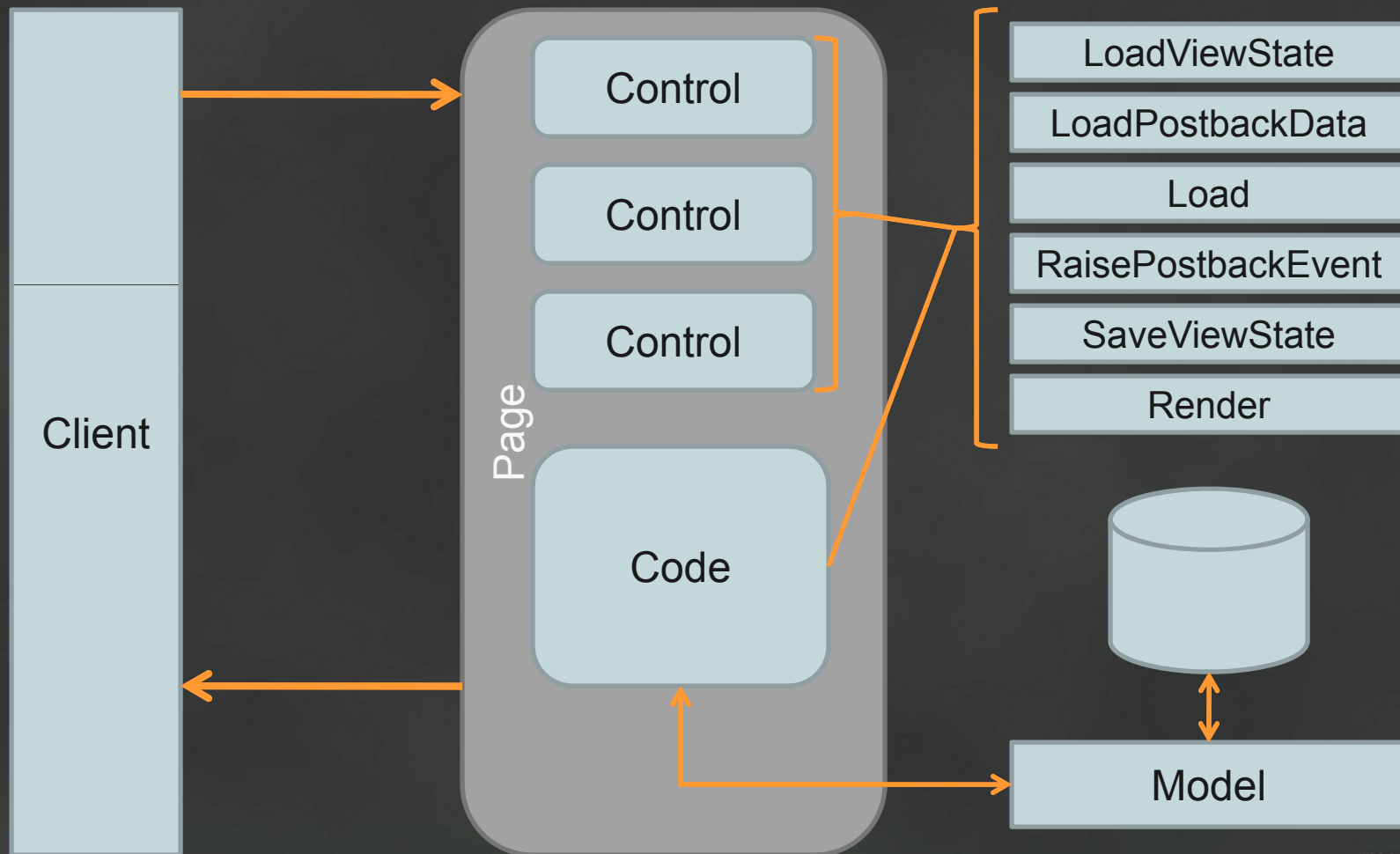
Key Differences

Classic ASP.NET	ASP.NET MVC
Postback	REST, HTML Post
Code behind	Separate controller
Single form	Multiple forms
Server controls	HTML, Helpers
Page and control lifecycle	Controller

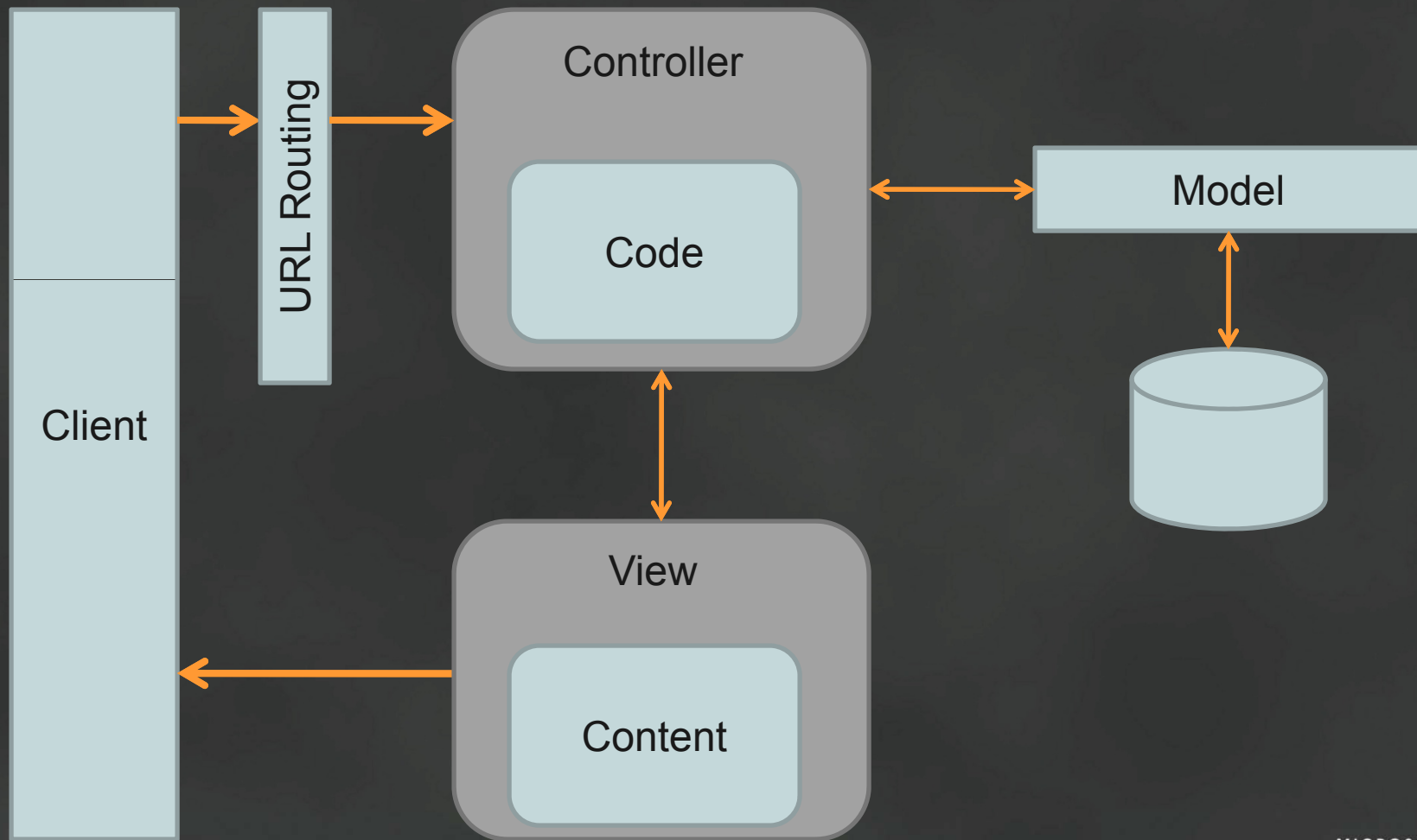
The MVC Framework

- **Model**
 - The data
- **View**
 - The view of the data
- **Controller**
 - Controls which view of the data you see

ASP.NET Page Lifecycle



ASP.NET MVC Lifecycle



Key Differences - Postback

- REST based

```
<form action="/Products/Detail/4" method="get">
```

- Supports multiple forms
- Specific actions, specific forms

```
/Products/Categories  
/Products/Categories/Add  
  
/Products/Detail/4  
/Products/Edit/4
```

Key Differences - Routing

- Routing Engine
 - Maps URLs to Controllers

```
routes.Add(  
    new Route("{controller}/{action}/{id}",  
    new MvcRouteHandler())  
    {  
        Defaults = new RouteValueDictionary(  
            new {action="Index", id=(string)null}),  
    });
```

Key Differences - Controller

- **Contains actions**
 - Fetch data from Model and pass to View

```
public class ProductsController: Controller
{
    public void Detail(int id)
    { }

    public void Edit(int id)
    { }
}
```

Routing

/Products/Detail/4

```
routes.Add(  
    new Route("{controller}/{action}/{id}",  
    new MvcRouteHandler())  
    {  
        Defaults = new RouteValueDictionary(  
            new {action="Index", id=(string)null}),  
        });
```

```
public class ProductsController: Controller  
{  
    public void Detail(int id)  
    { }  
  
    public void Edit(int id)  
    { }  
}
```

Routing

```
/Products/Detail/4
```

```
routes.Add(  
    new Route("{controller}/{action}/{id}",  
    new MvcRouteHandler())  
    {  
        Defaults = new RouteValueDictionary(  
            new {action="Index", id=(string)null}),  
    });
```

```
public class ProductsController: Controller  
{  
    public void Detail(int id)  
    { }  
  
    public void Edit(int id)  
    { }  
}
```

Routing

```
/Products/Detail/4
```

```
routes.Add(  
    new Route("{controller}/{action}/{id}",  
    new MvcRouteHandler()  
    {  
        Defaults = new RouteValueDictionary(  
            new {action="Index", id=(string)null}),  
        });
```

```
public class ProductsController: Controller  
{  
    public void Detail(int id)  
    { }  
  
    public void Edit(int id)  
    { }  
}
```

Key Differences - Interface

- View doesn't need Server Controls
 - Because there is no page lifecycle
- Current view is HTML and render code

```
<ul>
  <% foreach (var p in ViewData.Products) { %>
  <li>
    <%= Html.ActionLink<ProductController>
      (c => c.Edit(p.ProductID), "Edit") %>
  <li>
  <% } %>
</ul>
```

Key Differences - Interface

```
<asp:ListView id="ProductList" runat="server">
  <LayoutTemplate>
    <ul><asp:PlaceHolder id="items" runat="server" /></ul>
  </LayoutTemplate>
  <ItemTemplate>
    <li>
      <%= Html.ActionLink("Edit", "Edit",
                          new {id = product.ProductID}) %>
    </li>
  </ItemTemplate>
</asp:ListView>
```

```
CategoryList.DataSource = viewData;
CategoryList.DataBind();
```

UI Helpers

- Easy creation of content
- Form actions
 - Post to named controller/action
- Links to named controllers/actions
- Rendering user controls

UI Helpers

```
<%= Html.Select("Product.CategoryID",  
                ViewData.Categories,  
                ViewData.Product.CategoryID) %>
```

```
<%= Html.TextBox("Product.ProductName",  
                ViewData.ProductName) %>
```

```
<%= Html.SubmitButton("submit", "save") %>
```

```
<%= Html.RenderUserControl("~/foo.ascx") %>
```

```
<%= Html.RenderUserControl("~/foo.ascx", new { Prop=2 }) %>
```

Integration

- MVC Pages are just web pages
- Navigate and post to standard ASPX pages
 - and vica versa

MVC and Ajax

- ASP.NET Ajax relies upon page model
 - Doesn't currently integrate with MVC
- Pure client side Ajax
 - Works fine

Summary

- MVC is not the solution
 - It is a solution
- Separation of concerns
- Increased testability

Resources

- <http://www.asp.net/downloads/3.5-extensions/>
- <http://weblogs.asp.net/scottgu>
- <http://blog.wekeroad.com/category/mvc/>