中国科学技术大学
University of Science and Technology of China

Software Architecture

SSE USTC Qing Ding
dingqing@ustc.edu.cn
http://staff.ustc.edu.cn/~dingqing

MVC, MVP and MVVM: A Comparison of Architectural Patterns

Overview

中国科学技术大学
University of Science and Technology of China

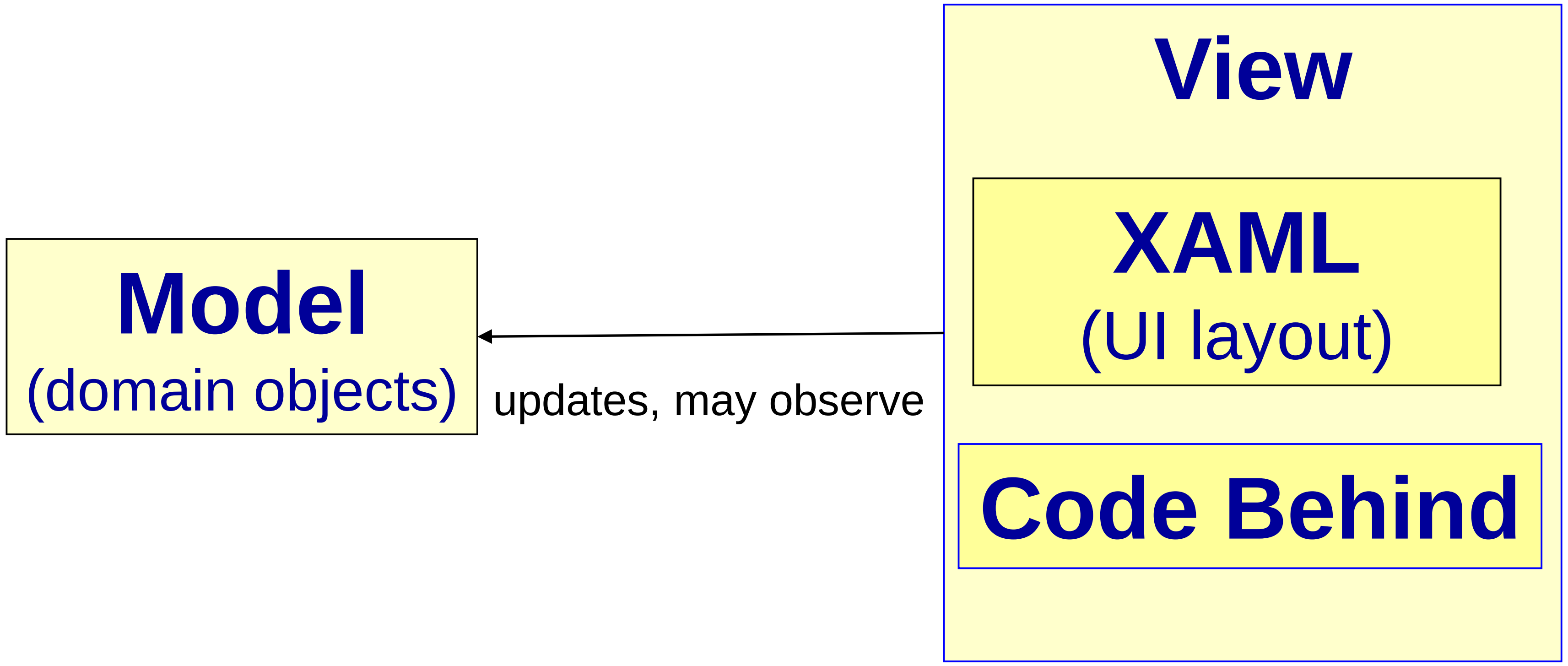
- What, why, how of Model-view-controller(C) or presenter(P) or View Model(VM) architecture patterns
- Model-View-Controller (MVC) pattern
- Model-View-Presenter (MVP) pattern
- Model-View-ViewModel (MVVM) pattern

Overview of MVVM

“Traditional” Windows Presentation Foundation(WPF) Programming

Overview of MVVM

Traditional WPF View Architecture



Overview of MVVM

中国科学技术大学
University of Science and Technology of China

XAML: window layout + named controls

```
<StackPanel>
   <TextBox x:Name="City" />
   <ComboBox x:Name="CountryList" SelectionChanged=… />
</StackPanel>
```

Code behind: event handlers and manipulating the controls

事件处理程序和操作控件

```
void CountryList_SelectionChanged(…)
{
    City.Text =
      GetCity(CountryList.SelectedItem as Country);
}
```

Overview of MVVM

中国科学技术大学
University of Science and Technology of China

Pros and Cons of the Traditional Model:

- Simplicity简单
- 可以随意操控 Power: manipulate controls as you please
- Difficult to Test 非常难以测试
- Cannot easily identify modifiable UI state不能轻易识别可修改的UI状态
- Encourages using UI as data storage使用UI作为数据存储
- Encourages mixing business logic and control manipulation业务逻辑和控制操作混合在一起

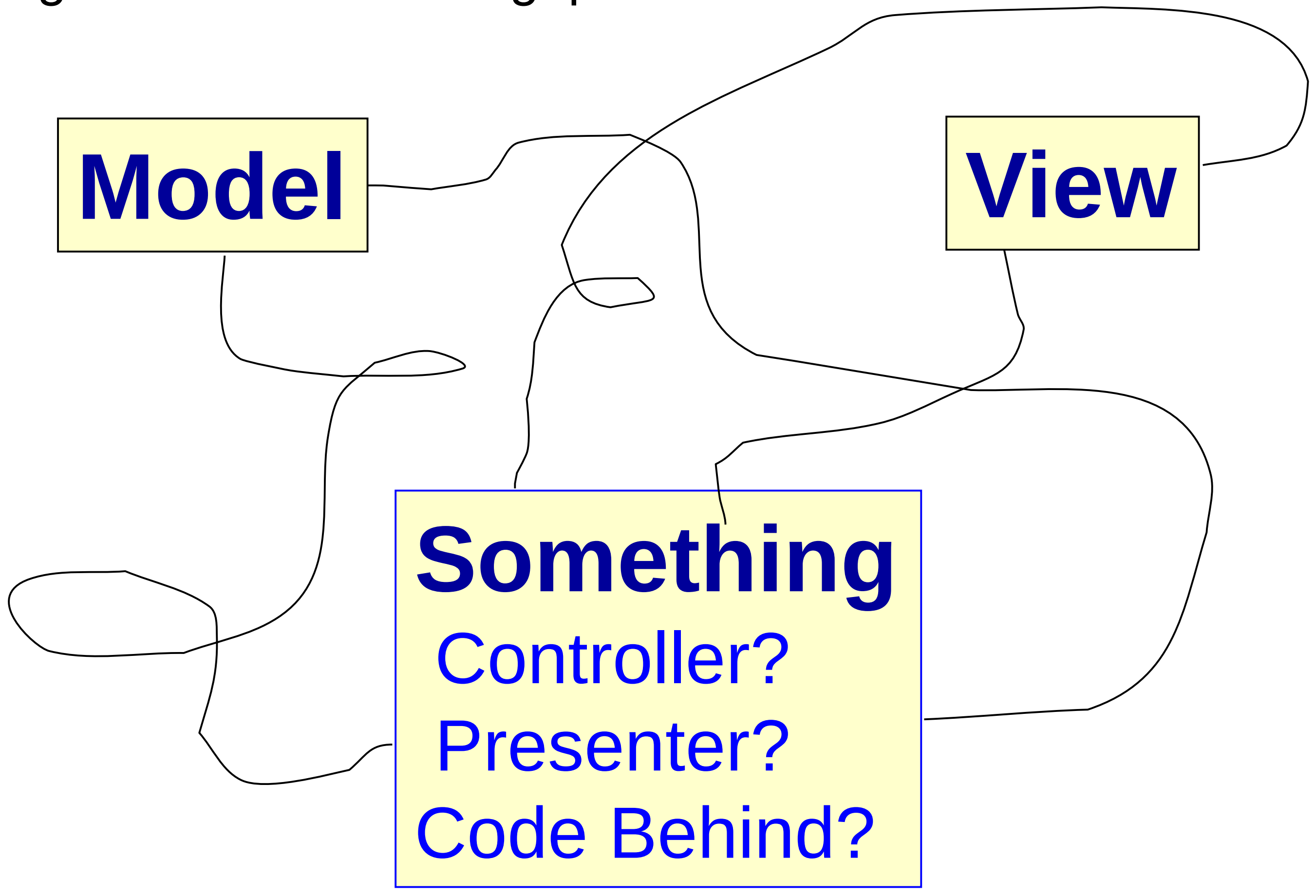
Overview of MVVM

What is MVVM

Overview of MVVM

- Model does not know anything about the UI
- View is a UI element (e.g. a Window)
- “Something” somehow fills the gap

· 模型对UI一无所知
· 视图是一个UI元素(例如窗口)
· “某物 ” 在某种程度上填补了空白



What is Model/View/(C or VM or P)

- Patterns that describe a modular approach to software development 描述软件开发的模块化方法的模式
- Modules include:
  - Model – Data
  - View – Presentation Layer
  - C or VM or P – Glue Logic

模块包括:
-模型-数据
-视图-表示层
- C或VM或P-粘合逻辑

- They are based upon a "Separation of Duties" 它们基于"职责分离 "
  - Seen in many other types o,f system frameworks

-在许多其他类型的系统框架中都可以看到

- How is this different from nTier development in the 90's - anybody remember Distributed interNet Architecture (DNA)

这与90年代的nTier发展有何不同?有人记得分布式互联网架构(DNA)吗?

Why Model-View-(C or VM or P)

- The Patterns all have similar goals, however, achieve them in different ways
  这些模式都有相似的目标，但是实现它们的方式不同
- The Patterns goals are to increase:
  - Modularity
  - Flexibility
  - Testability
  - Maintainability

模式的目标是增加：
-模块化
-灵活性
-可测试性
-可维护性

Model-View-Controller (MVC)

中国科学技术大学
University of Science and Technology of China

第一次描述是在1979年为Smalltalk在施乐PARC

- First described in 1979 for Smalltalk at Xerox PARC
- Controller is centerpiece that decouples the Model and View 控制器是分离模型和视图的核心部件
- Control flow:
  - User interaction event
  - Controller handles event and converts it to a user action the Model can understand
  - Model manages the behavior and data of the application domain
  - The View interacts with the Controller and Model to generate a user interface

控制流:
-用户交互事件
-控制器处理事件并将其转换为模型可以理解的用户动作
-模型管理应用领域的行为和数据
-视图与控制器和模型交互，生成用户界面

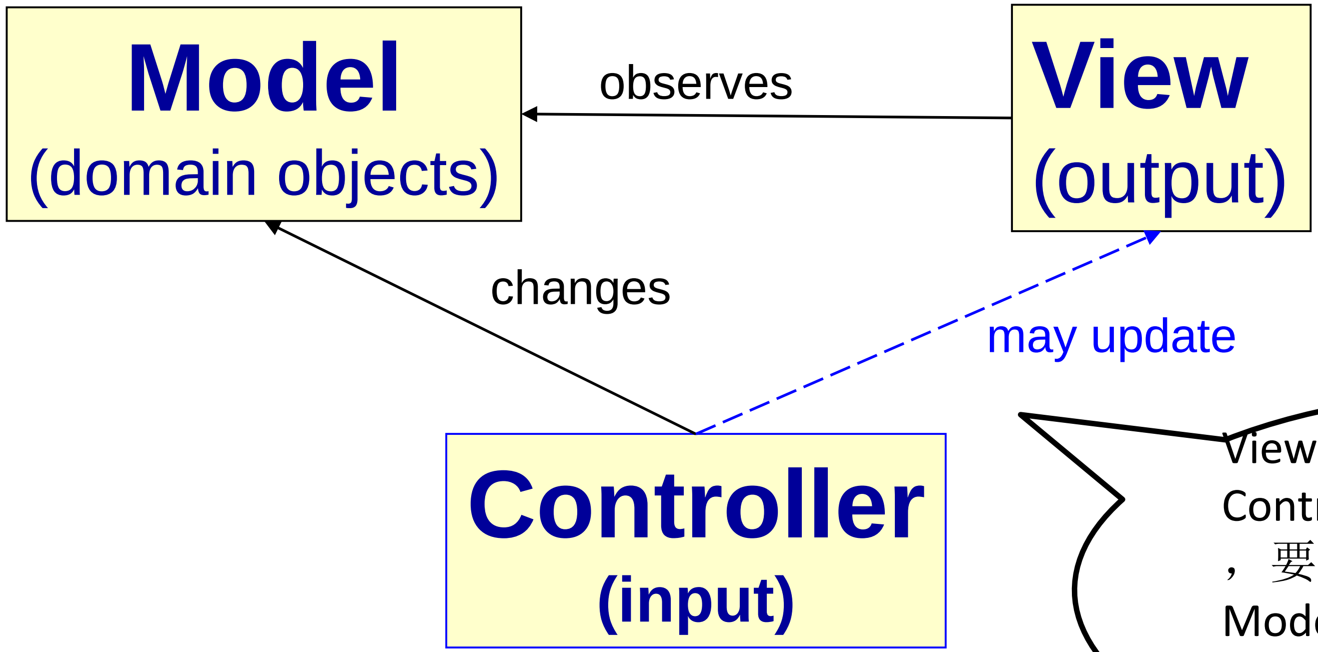
1979: Model View Controller

中国科学技术大学 University of Science and Technology of China

我已经记不清有多少次我看到一些被描述为MVC的东西，结果却与它完全不同

I've lost count of the times I've seen something described as MVC which turned out to be nothing like it.

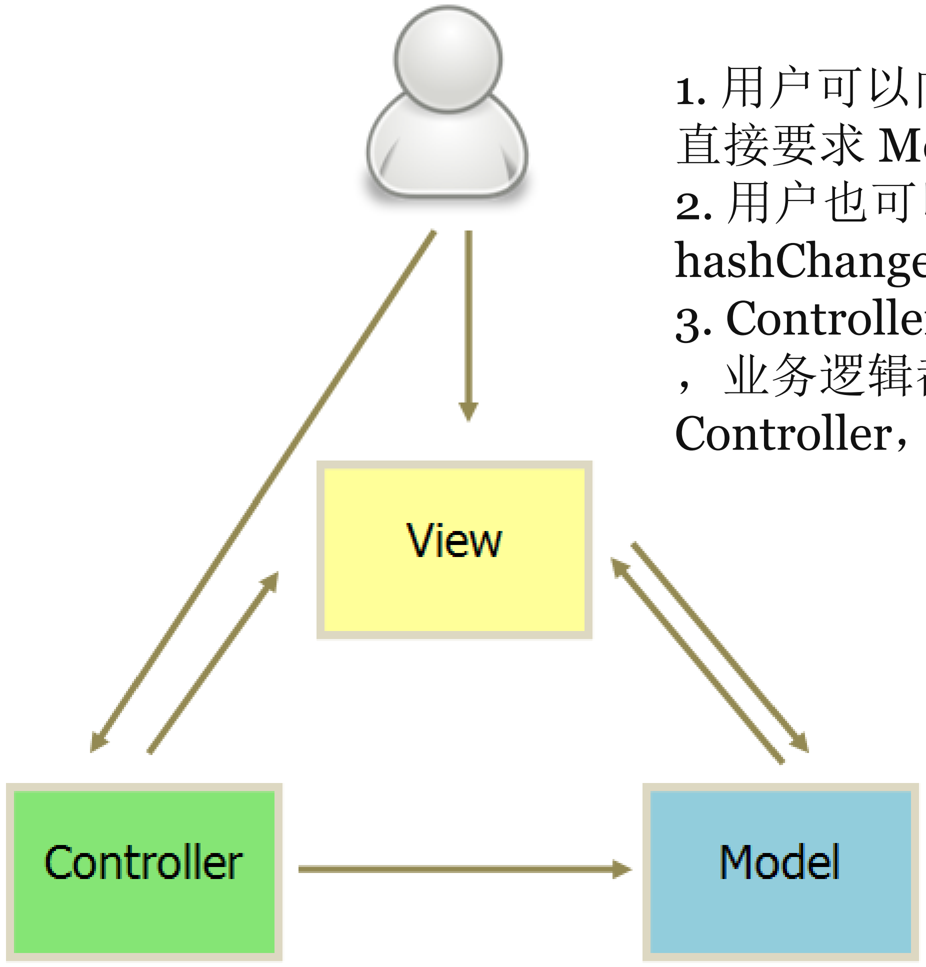
Martin Fowler



View 传送指令到 Controller
Controller 完成业务逻辑后，要求 Model 改变状态
Model 将新的数据发送到 View，用户得到反馈

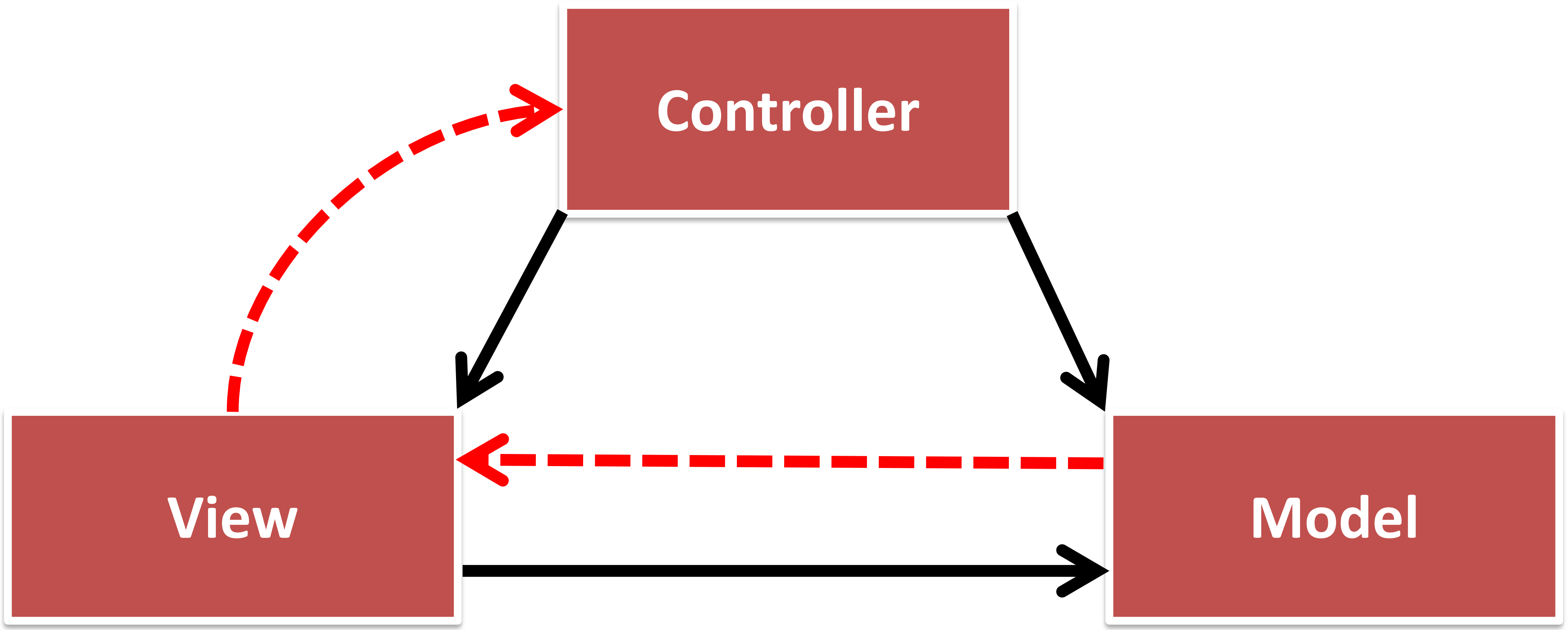
http://martinfowler.com/eaaDev/uiArchs.html

Backbone.js

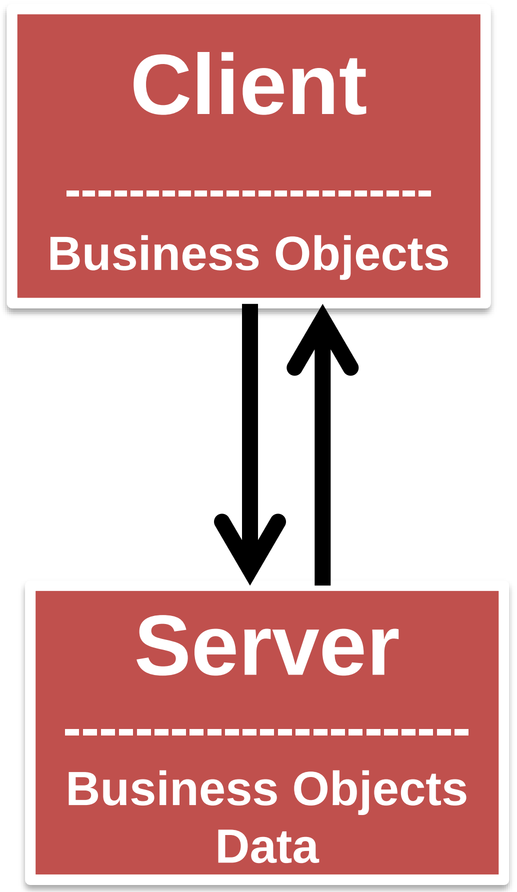


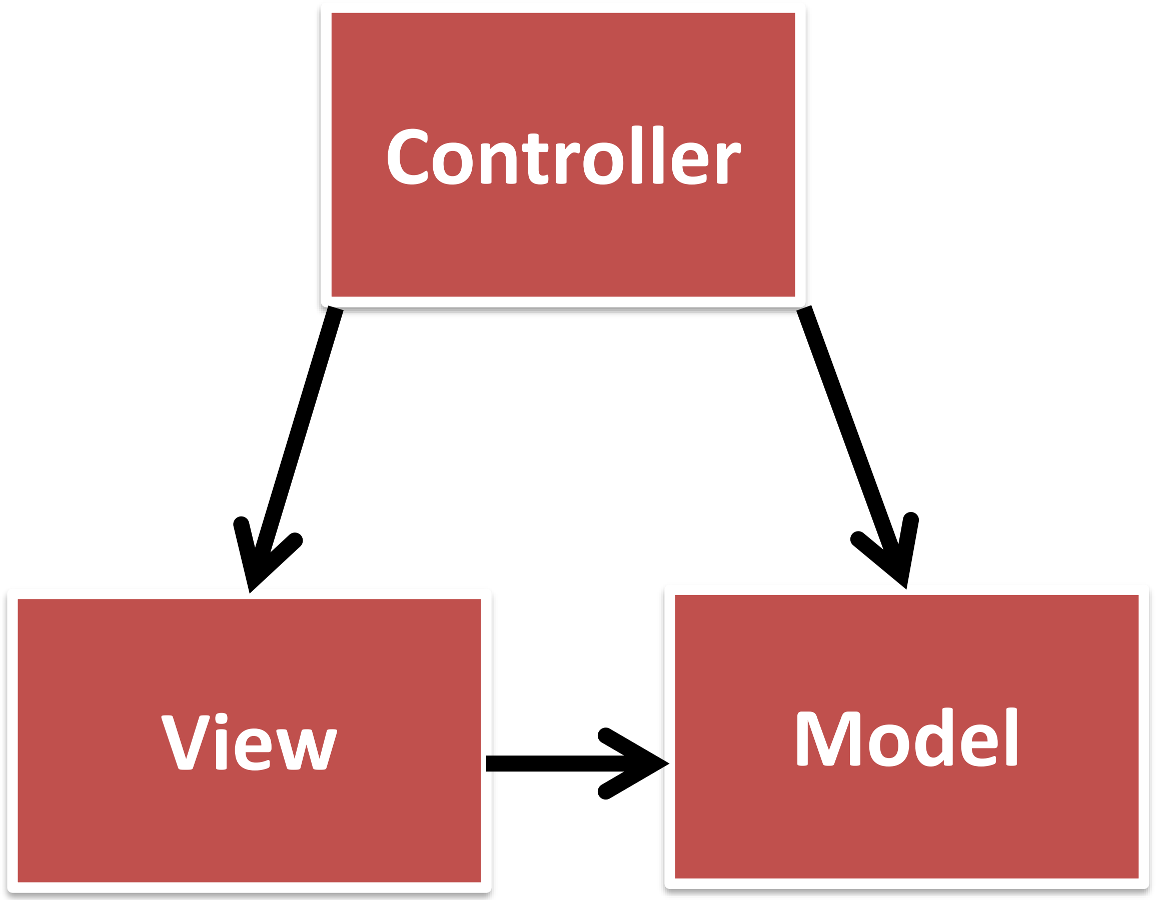
1. 用户可以向 View 发送指令（DOM 事件），再由 View 直接要求 Model 改变状态。
2. 用户也可以直接向 Controller 发送指令（改变 URL 触发 hashChange 事件），再由 Controller 发送给 View。
3. Controller 非常薄，只起到路由的作用，而 View 非常厚，业务逻辑都部署在 View。所以，Backbone 索性取消了 Controller，只保留一个 Router（路由器）。

MVC Observer Pattern



Client/Server (DNA) vs MVC

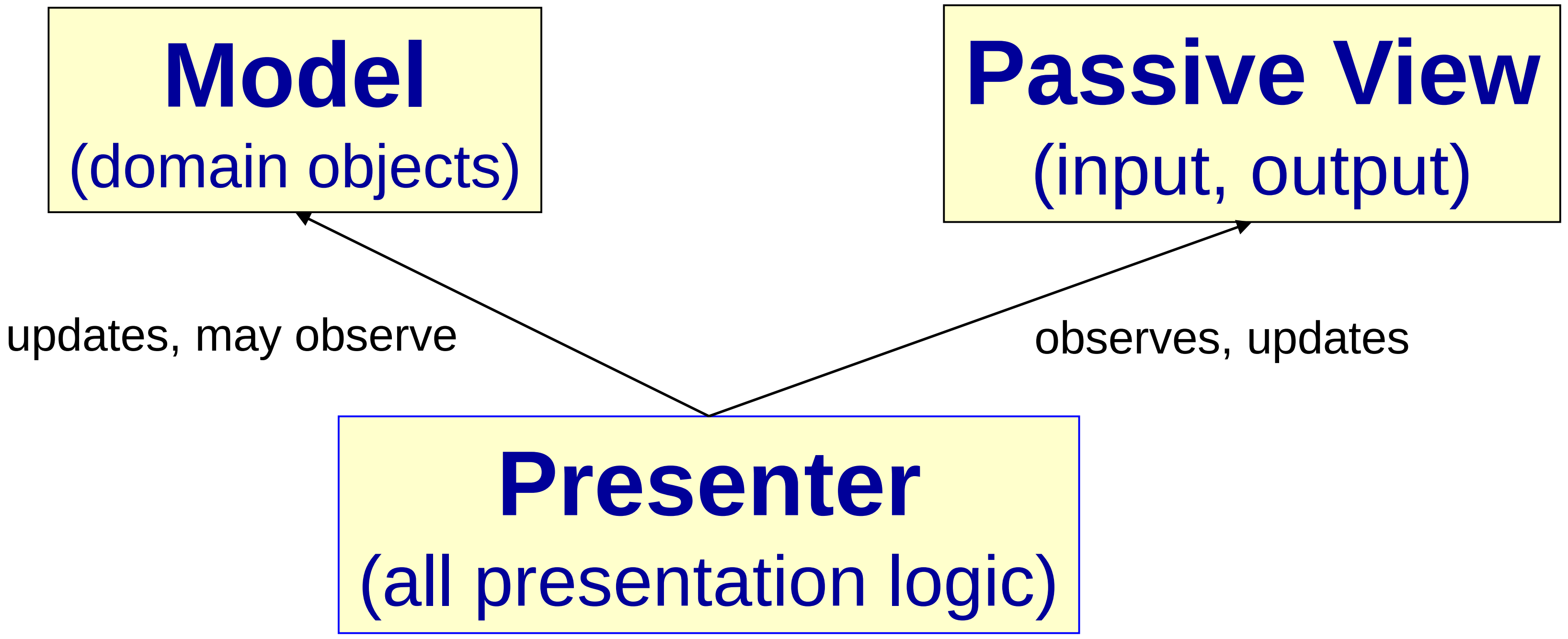




Model-View-Presenter (MVP)

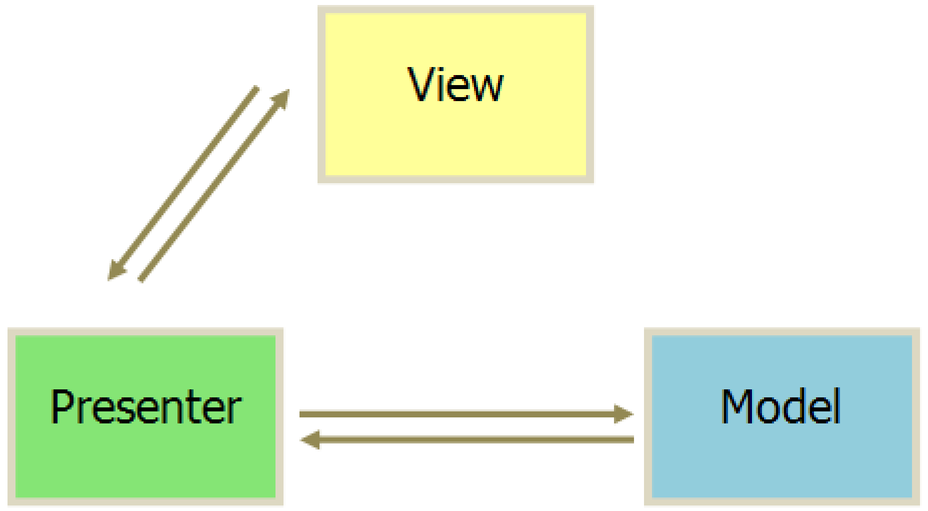
- MVP originated in early 1990s
- MVP is a derivative of MVC MVP中P的功能比MVC中C的功能要多，两者的M差距不大
- Two types of implementation
  - Passive View
  - Supervising Controller
- Presenter assumes the functionality of the MVC Controller
- View is responsible for handling UI events
- Model becomes strictly a Domain Model
- More User Interface centric

Passive View



http://martinfowler.com/eaaDev/PassiveScreen.html

MVP

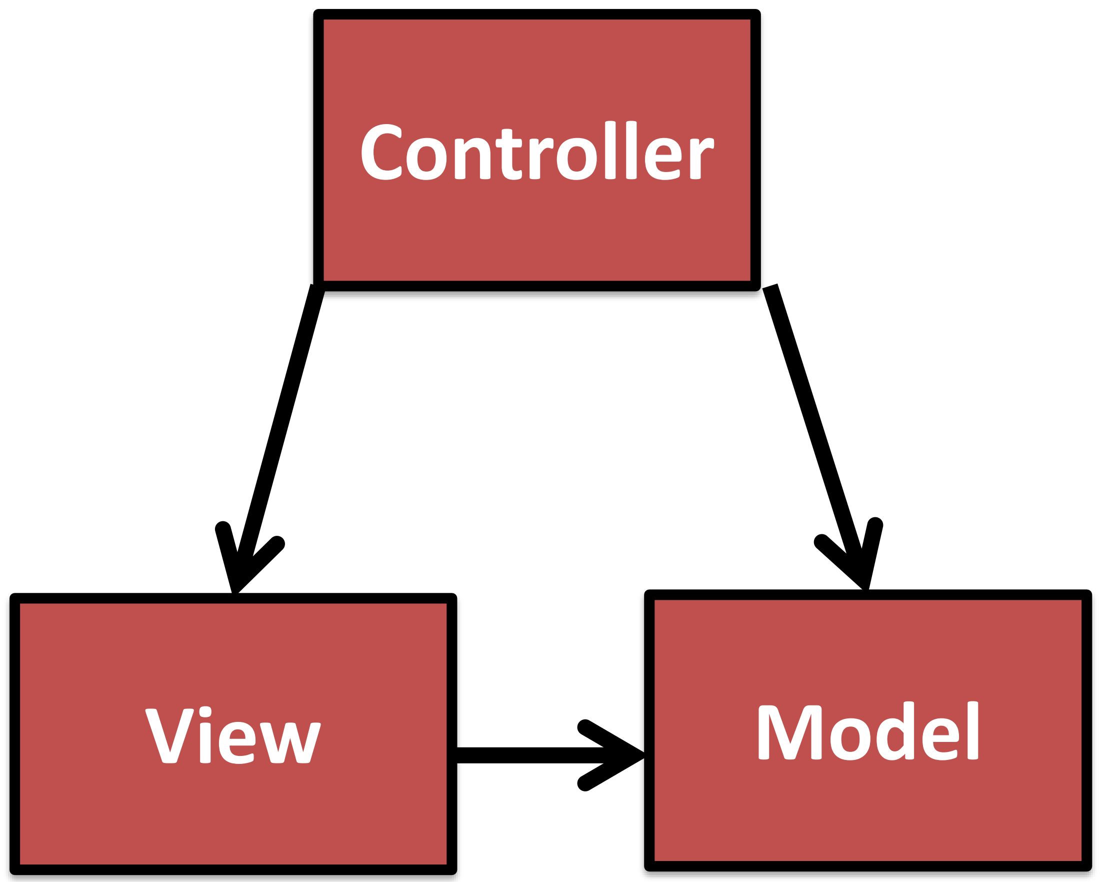


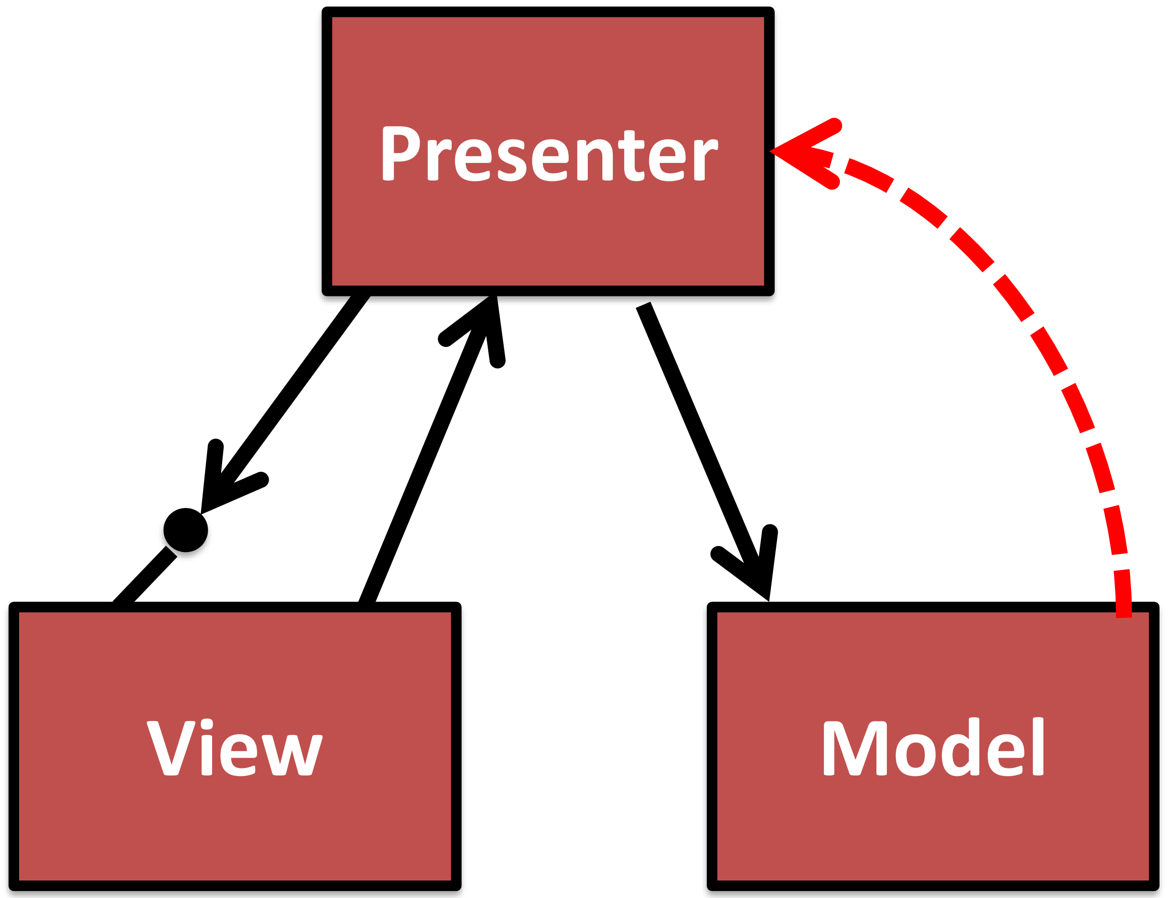
1. 各部分之间的通信，都是双向的。

2. View 与 Model 不发生联系，都通过 Presenter 传递。

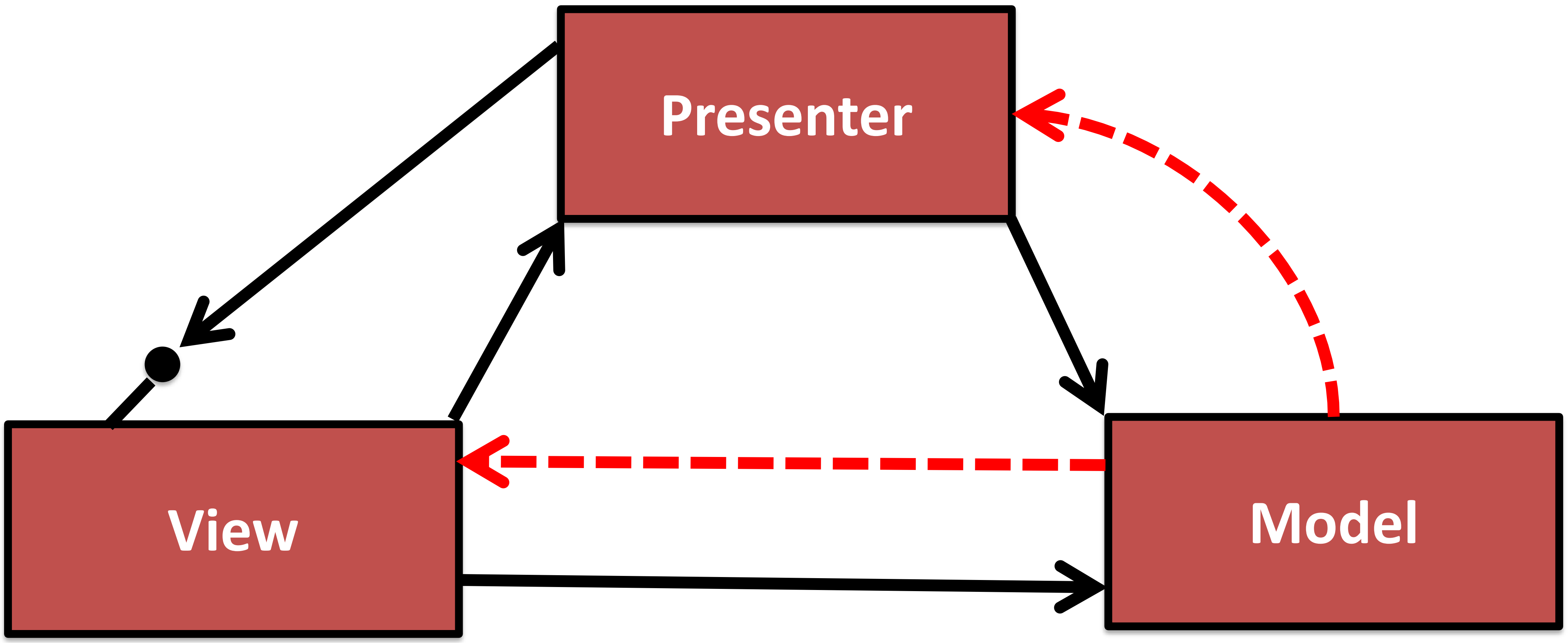
3. View 非常薄，不部署任何业务逻辑，称为"被动视图"（Passive View），即没有任何主动性，而 Presenter非常厚，所有逻辑都部署在那里。

MVC vs MVP (Passive)

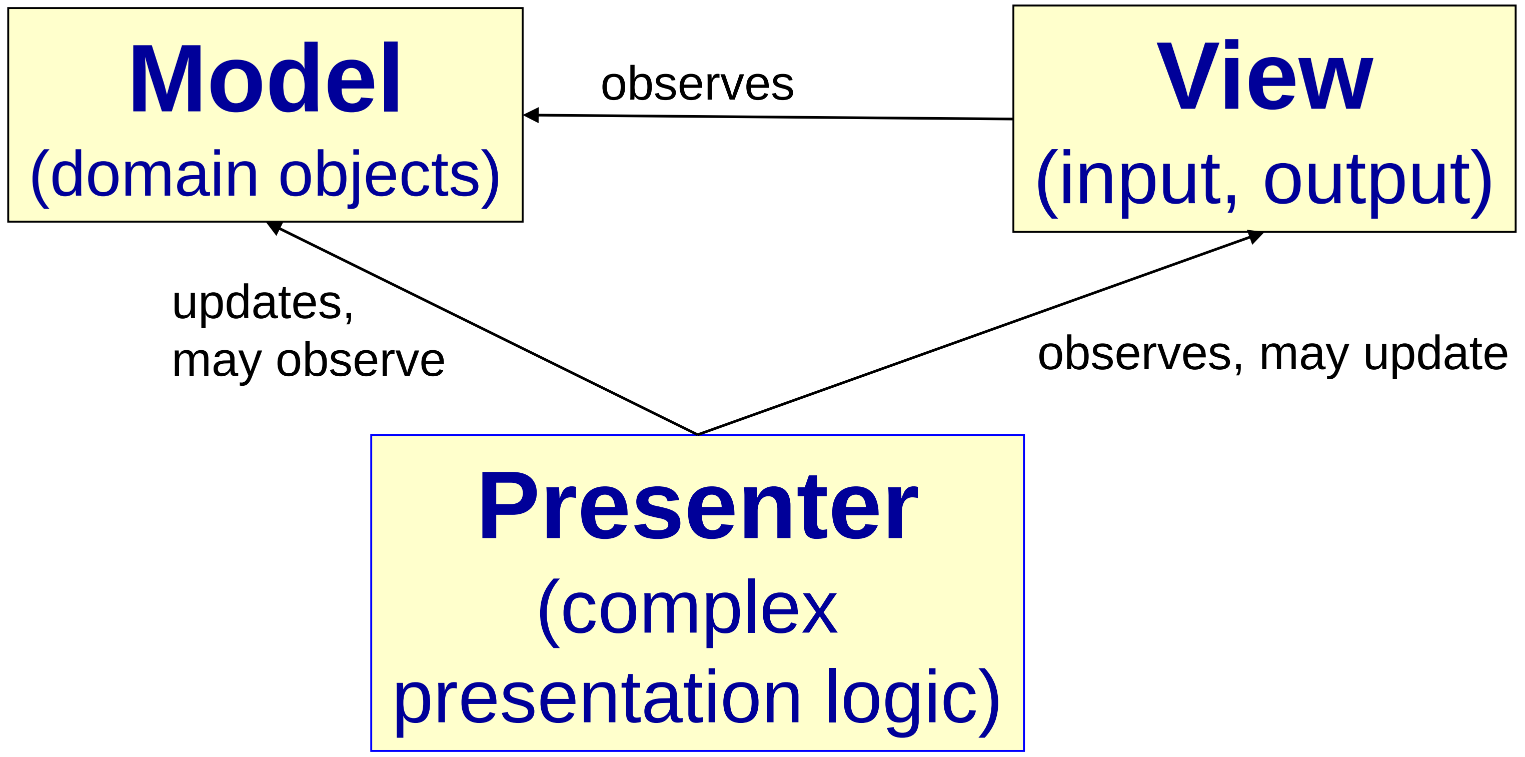




MVP – Supervising Controller Pattern



2004: Model View Presenter (supervising controller)

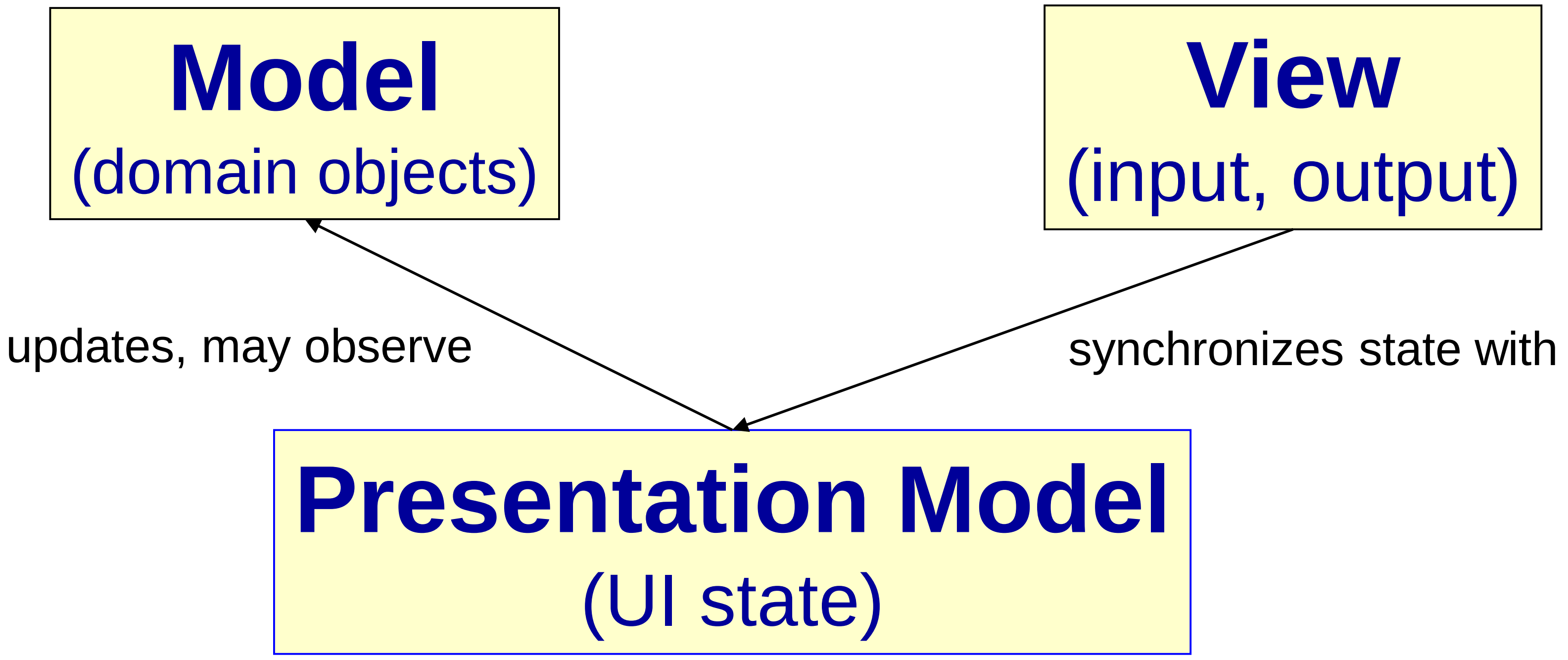


“ 让视图处理尽可能多的内容，只有涉及到更复杂的逻辑时Presenter才会介入。 ”

“Let the view handle as much as possible and only step in when there's more complex logic involved.”
http://martinfowler.com/eaaDev/SupervisingPresenter.html

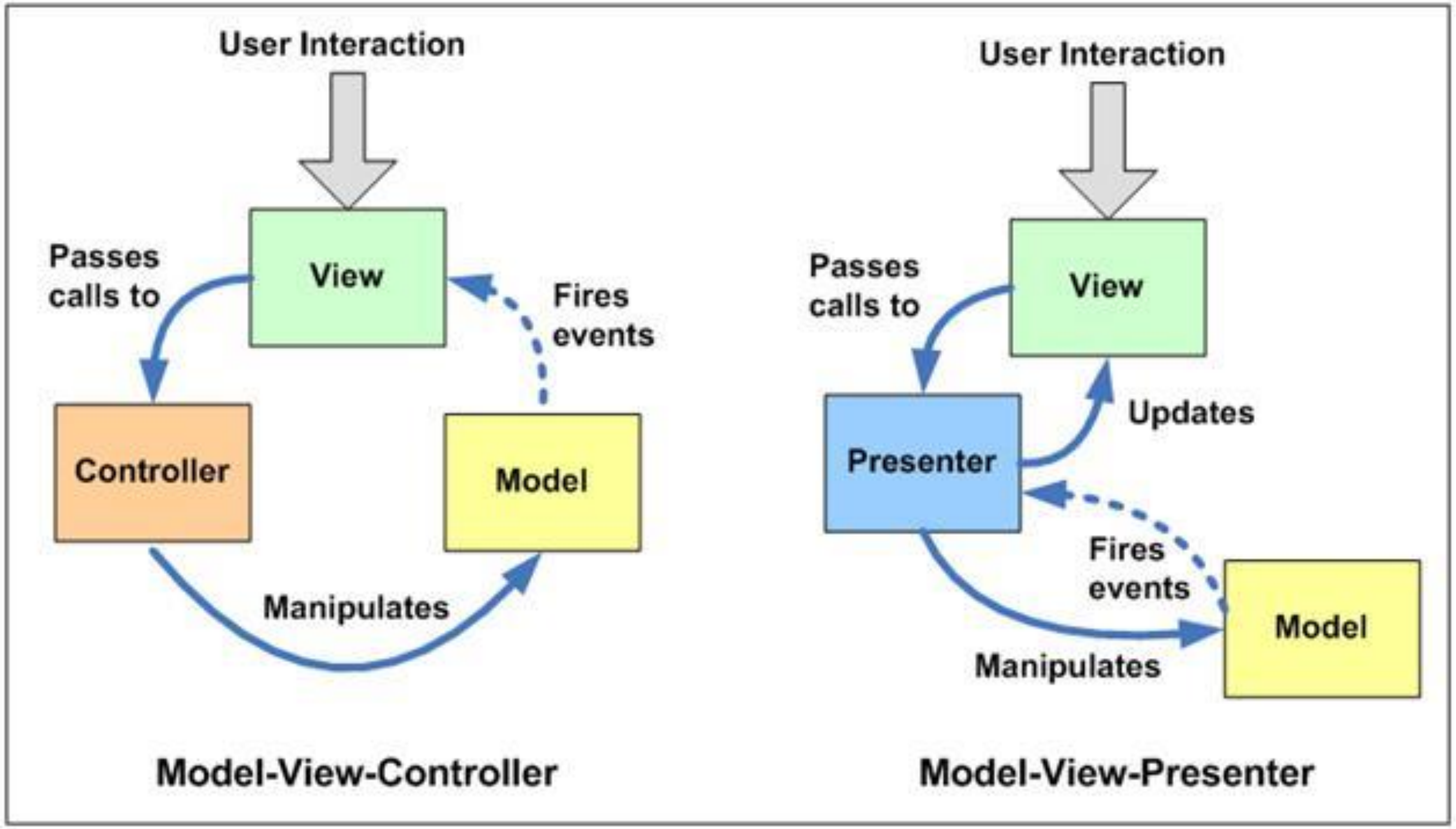
Presentation Model

中国科学技术大学 University of Science and Technology of China



Presentation model contains all variable UI state in a UI-agnostic manner
The most annoying part of Presentation Model is the synchronization. Ideally some kind of framework could handle this... like .NET's data binding.
http://martinfowler.com/eaaDev/PresentationModel.html

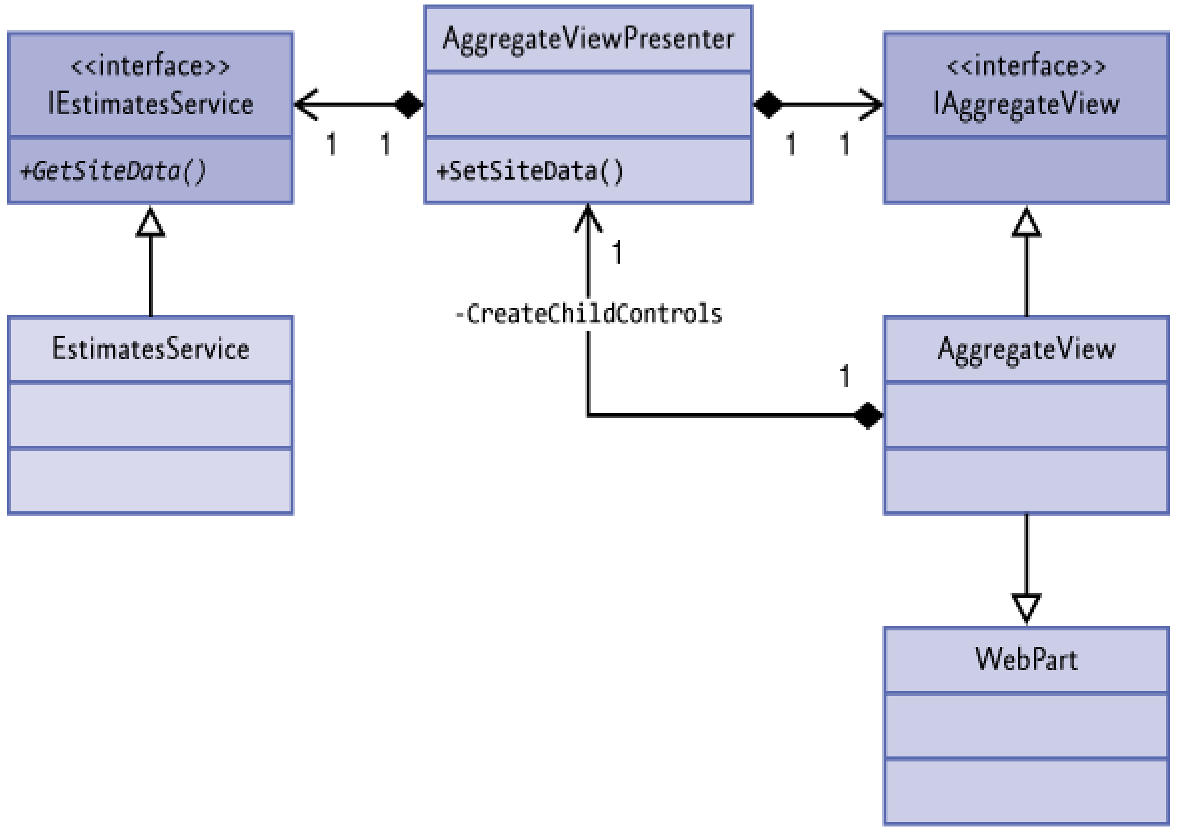
Compared to…


User Interaction
Passes calls to
View
Fires events
Controller
Model
Manipulates
Model-View-Controller
User Interaction
Passes calls to
View
Updates
Presenter
Fires events
Model
Manipulates
Model-View-Presenter

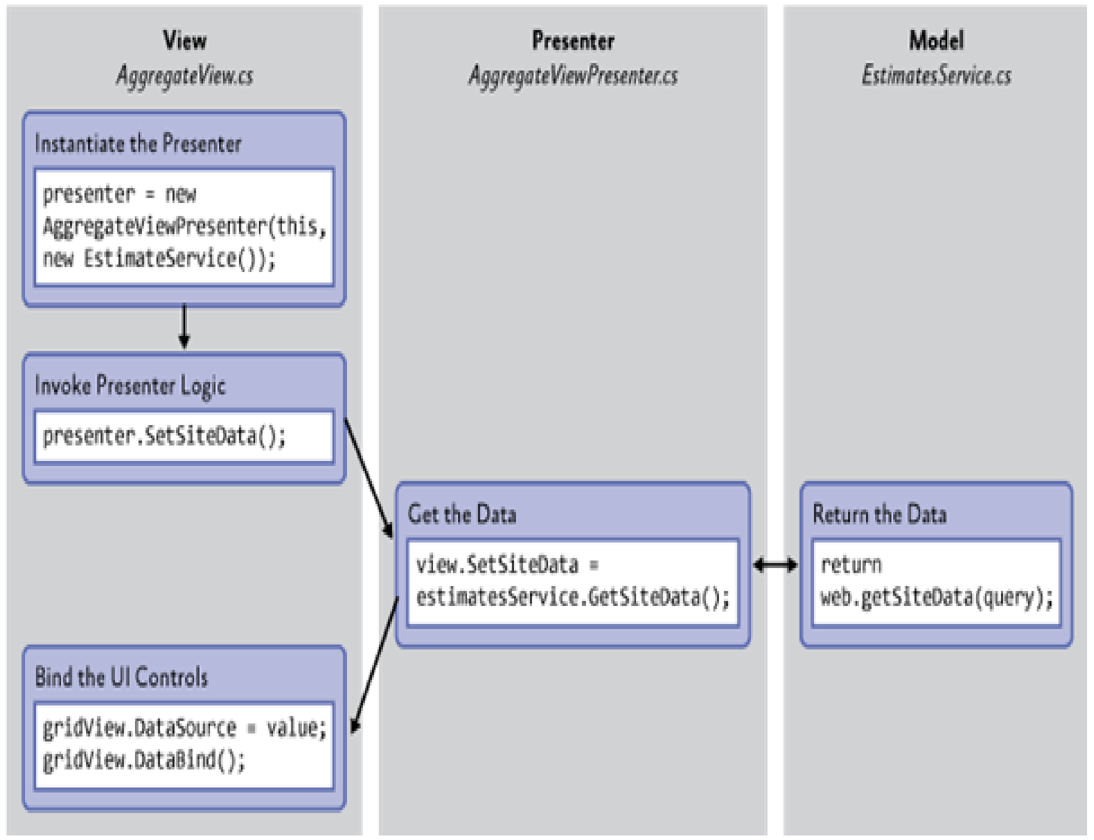
SharePoint 2010 Developer Guidance

- MVP Class Diagram and Flow of Execution for the SandBox Execution Model

Class diagram for the Sandbox RI



Flow of execution in the Sandbox RI



```
presenter = new
AggregateViewPresenter(this,
new EstimateService());
```

```
presenter.SetSiteData();
```

```
gridView.DataSource = value;
gridView.DataBind();
```

```
view.SetSiteData =
estimatesService.GetSiteData();
```

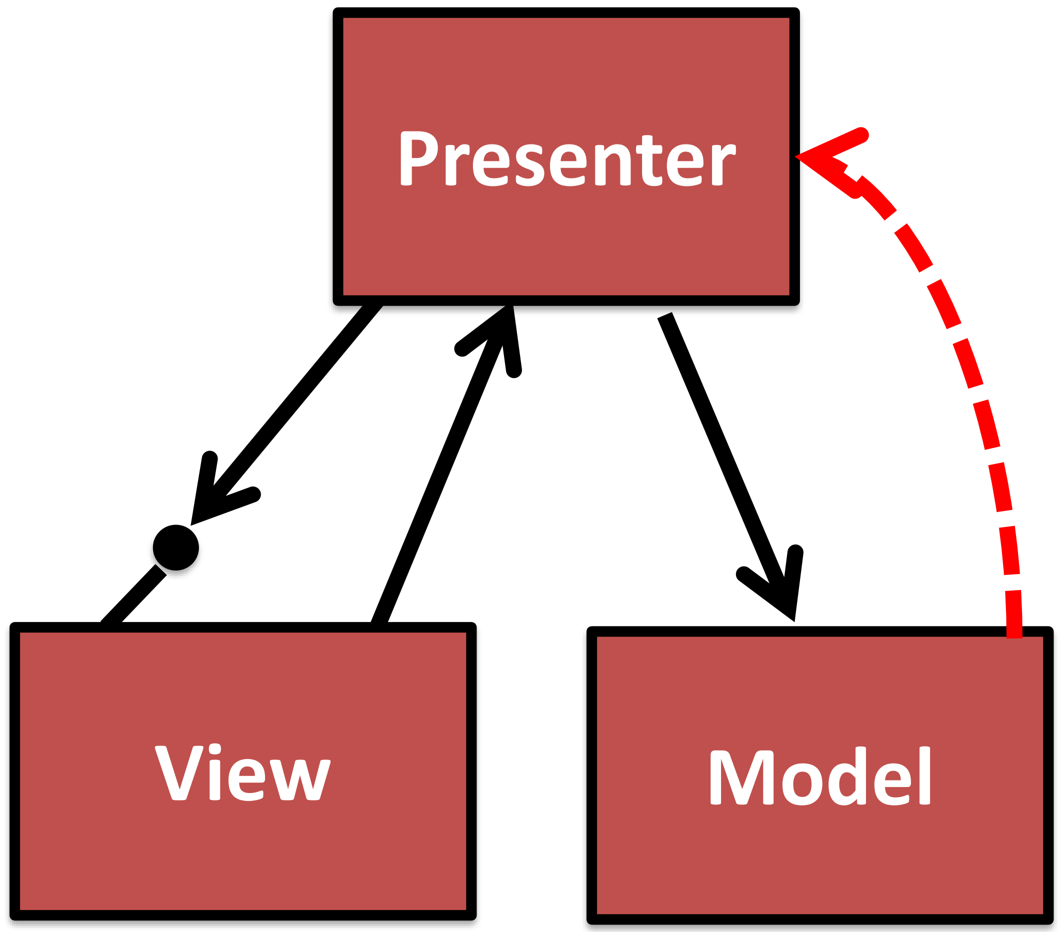
```
return
web.getSiteData(query);
```

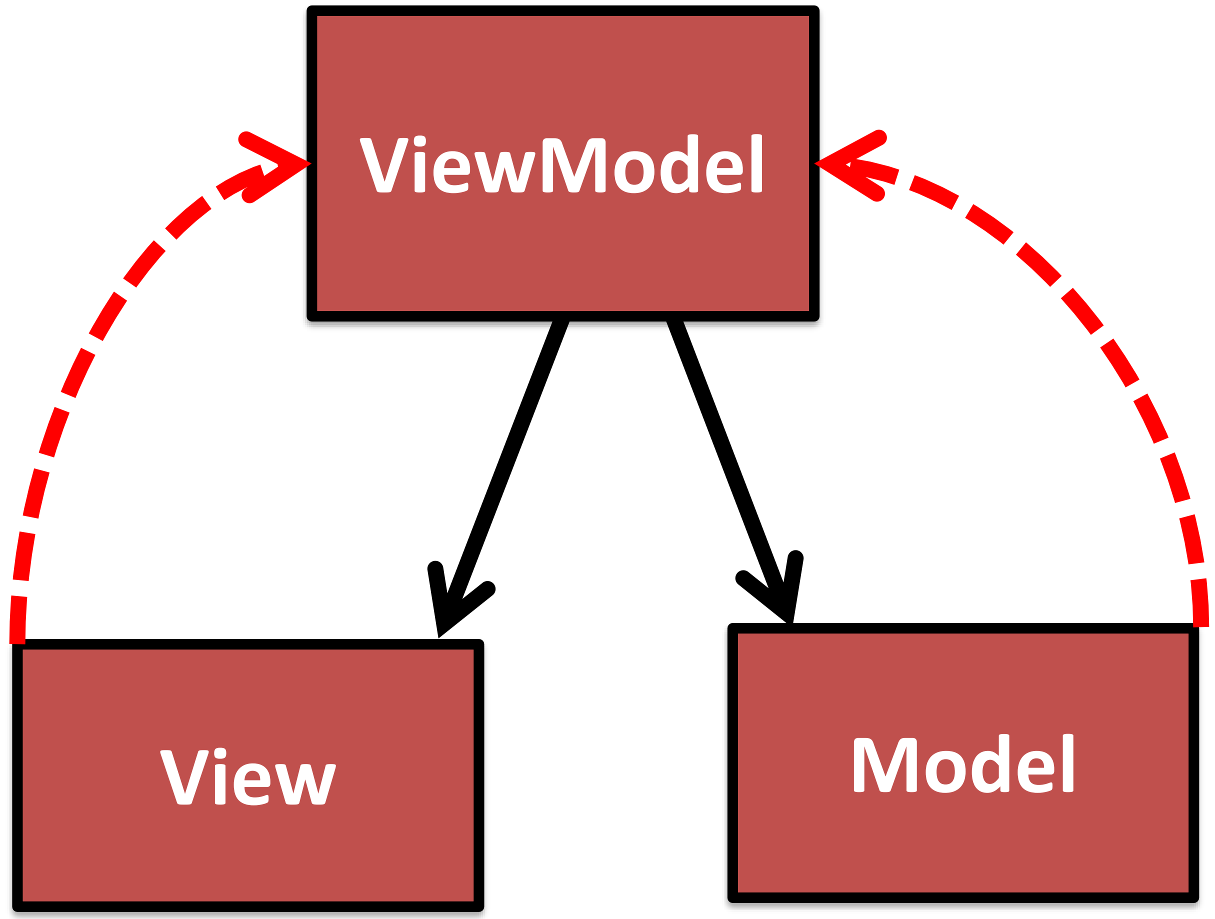
Model-View-ViewModel (MVVM)

- Largely based on MVC
- Specialization of the MVP pattern known as the Presentation Model
- Built specifically for the WPF and Silverlight environments
- Model and View works just like MVC
- ViewModel is a “Model of the View”将业务逻辑数据与表现数据分开
  - It extends the Model with Behaviors the View could use
  - Data Binding between View and Model
  - Passes commands between the View and Model

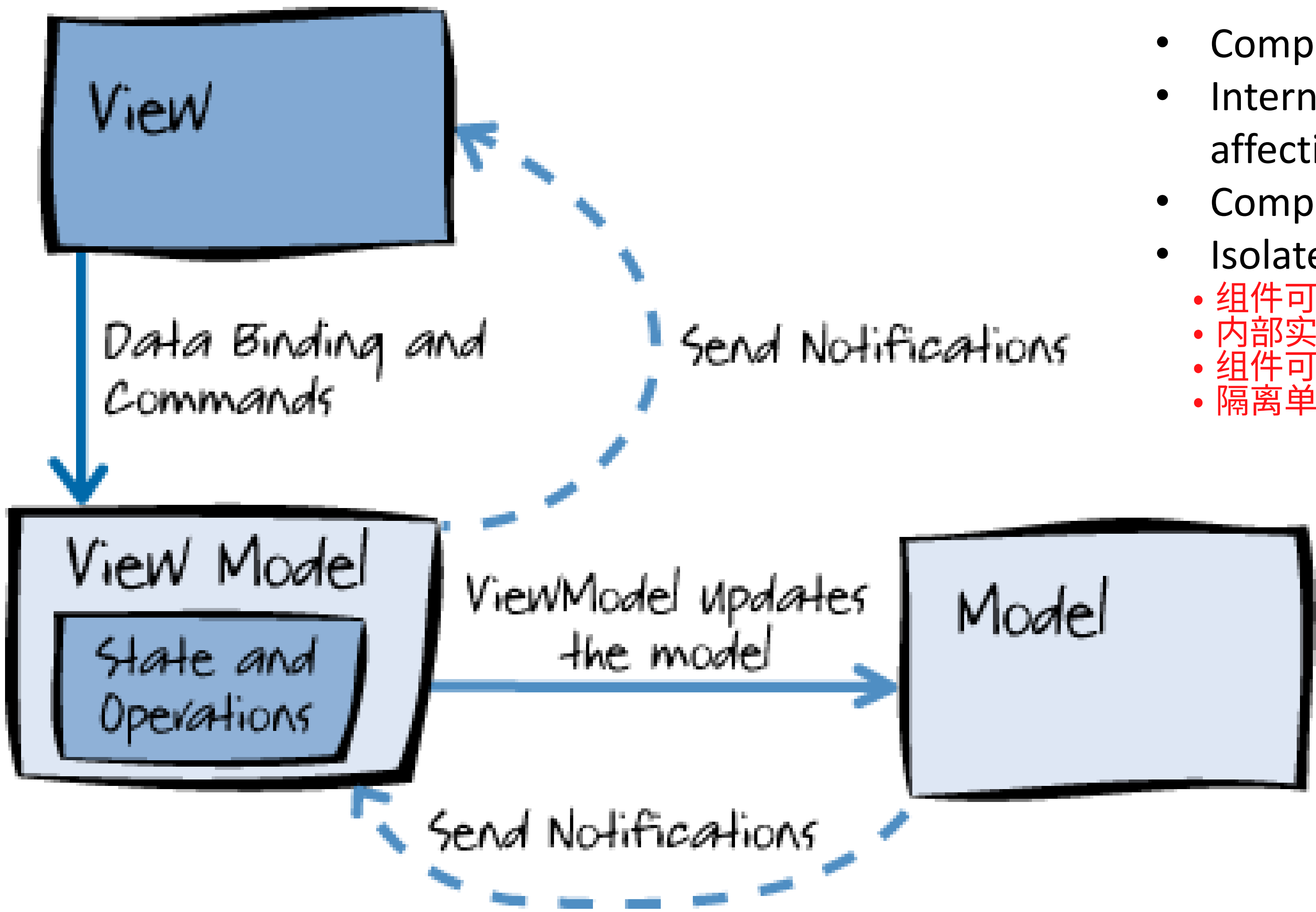
MVP (Passive) vs MVVM

中国科学技术大学 University of Science and Technology of China





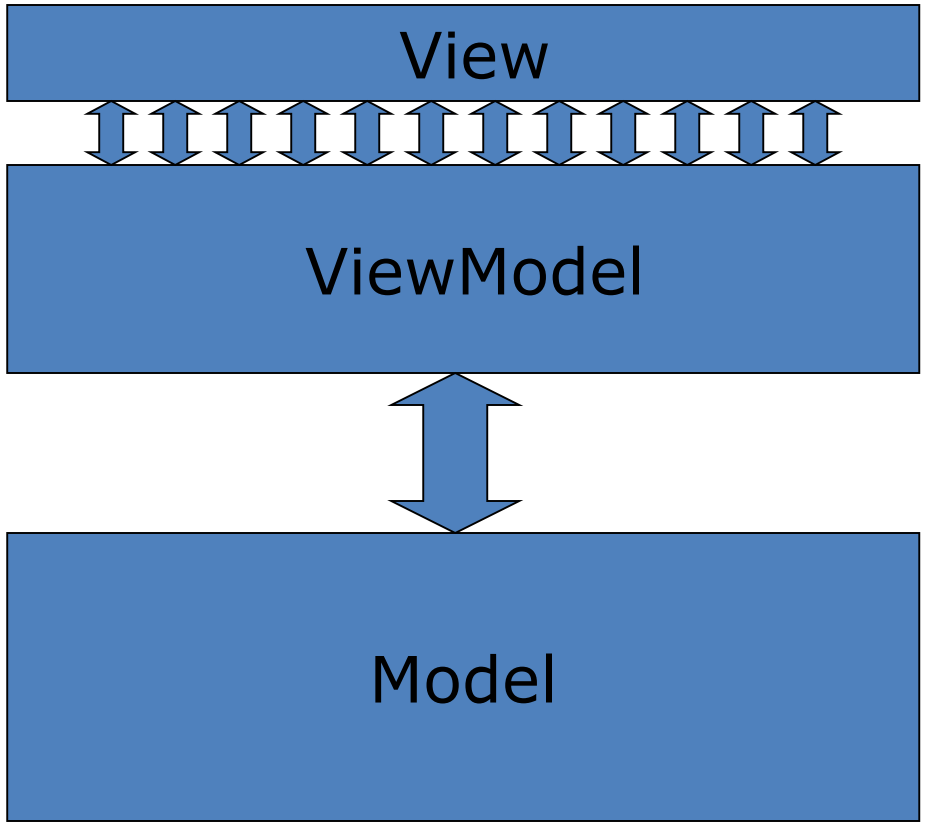
The MVVM Pattern



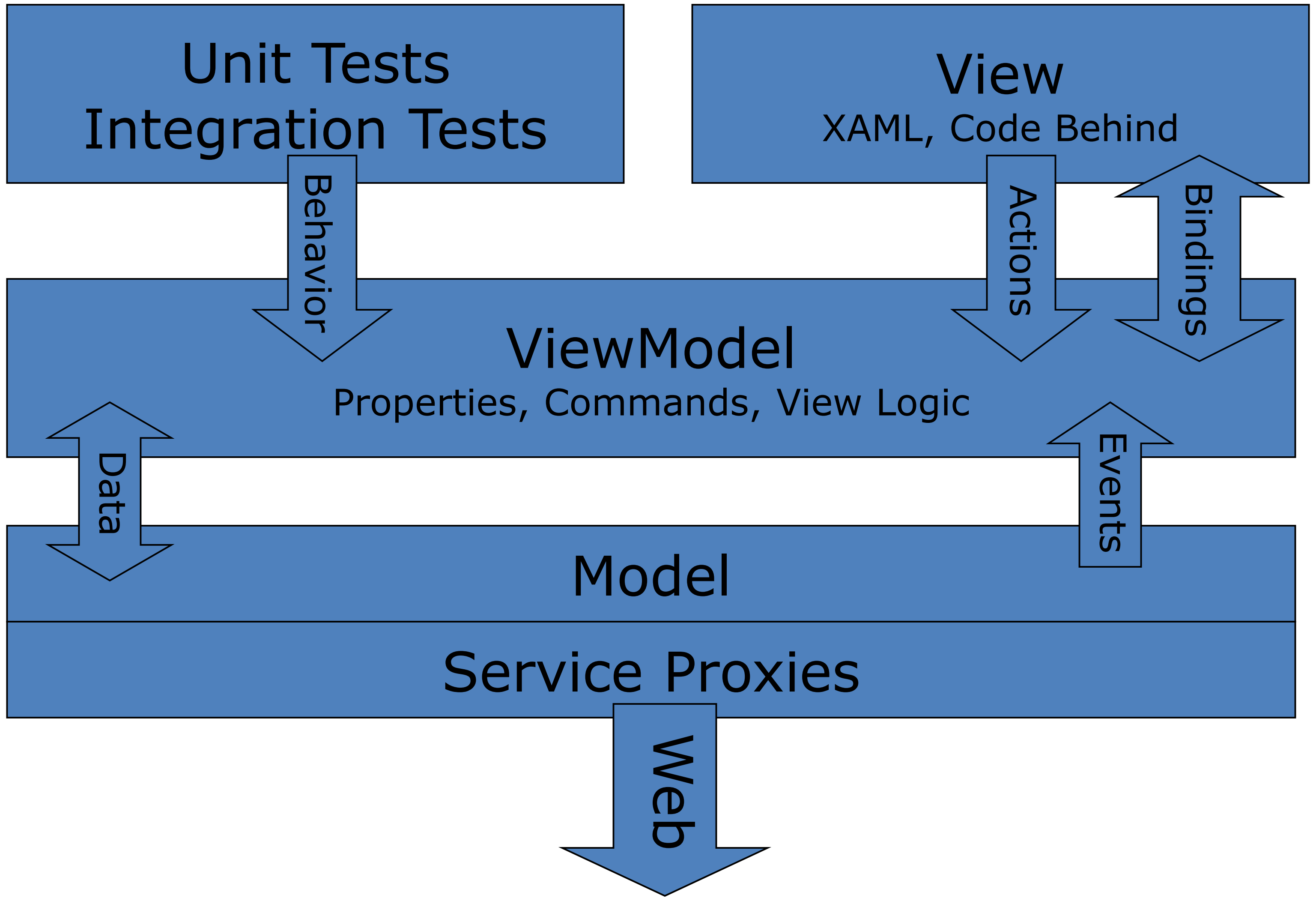
- Components can be swapped out
- Internal implementation can be changed without affecting other components.
- Components can be worked on independently
- Isolated unit testing
  - 组件可以换出来
  - 内部实现可以在不影响其他组件的情况下改变
  - 组件可以独立工作
  - 隔离单元测试

采用双向绑定（data-binding）：View的变动，自动反映在 ViewModel，反之亦然。Angular 和 Ember 都采用这种模式。

How Do We Achieve that?



A More Complete Diagram



MVVM – The View

- Defines structure, layout and appearance.
- Only knows about the View Model.
- Ideally it is pure XAML with little to no code behind.
- Can have it’s own View Model or inherit from its parent.
- At runtime, UI responses to View Model properties raising change notification events.
- A view on Windows Phone is typically a page in the application.

MVVM – The Model

- This is your domain model.
- Includes business and validation logic.
- Examples: Repositories, business objects, data transfer objects (DTOs), value objects etc.
- Only knows about itself – not the views, controllers etc.

MVVM – The View Model

- Intermediary between View and the Model.
- Only knows about the Model.
- Handles view logic.
- Interacts with Model by invoking methods in the model classes.
- Provides the data from model in a form that the view can use.
- Provides command implementations that the view uses.
  - Example: Clicking a button on the UI triggers a command in the View Model.
- Defines state.

Mini Patterns

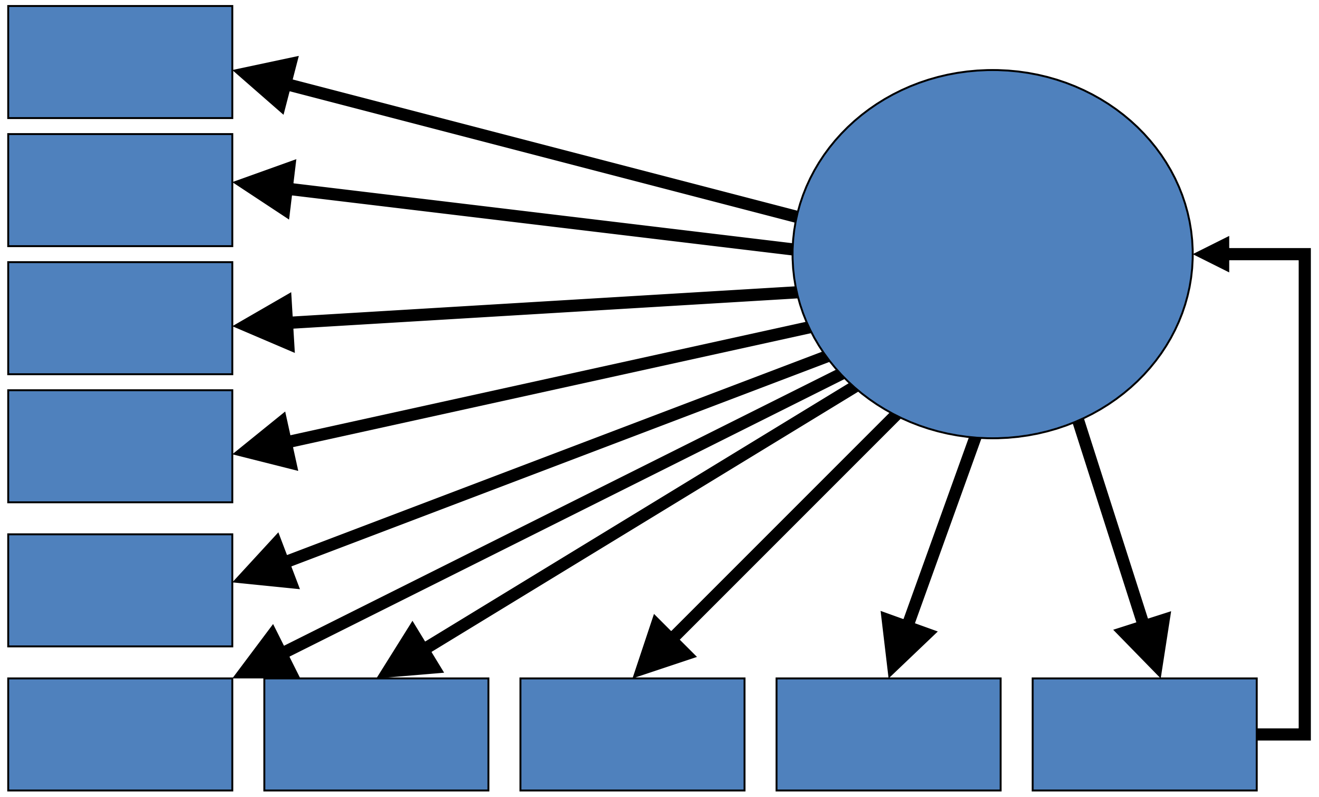
NotifyPropertyChanged

```
public interface INotifyPropertyChanged
{
   event PropertyChangedEventHandler
         PropertyChanged;
}
```

Command Pattern

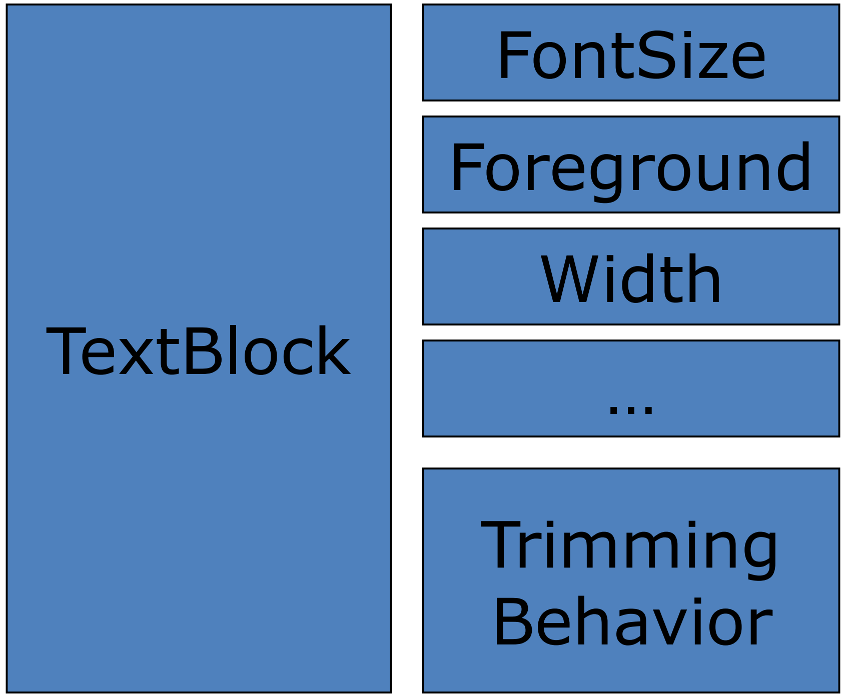
```
public interface ICommand
{
  bool CanExecute(object param);
  void Execute(object param);
  event EventHandler CanExecuteChanged;
}

public class DelegateCommand : ICommand
{
  public DelegateCommand(
    Action<object> command,
    Predicate<object> canExecute);
}
```

Event Aggregator Pattern



Attached Behaviors

```
<TextBlock Foreground="Red" Width="200"
    Behaviors:Update.WhenTextChanged="True"
/>
```



MVVM Guidelines

中国科学技术大学
University of Science and Technology of China

1. Reduce or eliminate your code-behind

2. Bind all of your UI inputs/outputs to your ViewModel

3. Implement INotifyPropertyChanged on your ViewModel

4. Put your view behavior into the ViewModel

5. Do not put any view state into the model

6. Only bind to a model object if there is no view-specific info

7. When testing, treat ViewModel as the Real UI

8. Avoid events. Use commands instead

Third Party Support

- Prism
- MVVM Light
- Caliburn
- Silverlight FX

MVVM – The Benefits

- Enables developer-designer workflow.
- Developers and designers can work independently and concurrently on their components.
- Developers can create unit tests without the need for the View.
- You can redesign the UI without touching the code.
- View Model acts as an adapter and allows changes to the model more easily.

MVVM vs MVC vs MVP - Controllers

- Controller determines which view is displayed.
- Events in the View trigger actions in the controller.
- Multiple Views for each controller.
- Example: ASP.NET MVC Website

MVVM vs MVC vs MVP - Presenters

- 2-way communication with the view.
- View calls functions on an instance of the presenter.
- Presenter talks to an interface implemented by the view.
- Single presenter for each view.
- Example: Windows Forms.

- NOTE: Not going into detail as I have never used MVP.

MVVM vs MVC vs MVP - ViewModels

- Again – 2-way communication with view.
- ViewModel represents the View.
- Matches up more closely with the View rather than the Model.
- View binds directly with View Model.
- Changes in the View are reflected automatically in the View Model and visa-versa.
- Single ViewModel for each View.
- Examples: WPF, XAML, Knockoutjs