

MVVM pattern in Adobe Flex

Докладчик: Мещеряков Сергей



Что это такое и для чего это нужно?

Паттерн **Model-View-ViewModel** — это паттерн, применяющийся при проектировании архитектуры приложения. Паттерн MVVM широко применяется при создании приложений с помощью Windows Presentation Foundation и Silverlight. Первоначально был представлен сообществу Джоном Госсманом (John Gossman) архитектором WPF и Silverlight в 2005 году как модификация паттерна Presentation Model.

MVVM используется для разделения модели и её представления, что необходимо, так как позволяет изменять их отдельно друг от друга. Например, программист задает логику работы с данными, а дизайнер соответственно работает с пользовательским интерфейсом.



SOCIAL TECHNOLOGIES DEVELOPMENT
LABORATORY

Удобства использования

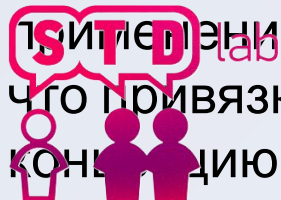
MVVM удобно использовать вместо классического [MVC](#) и ему подобных в тех случаях, когда в платформе, на которой ведется разработка, присутствует «связывание данных».

В [MVC](#) MVC/[MVP](#) изменения в пользовательском интерфейсе не влияют непосредственно на модель, а предварительно идут через Контроллер/Presenter.

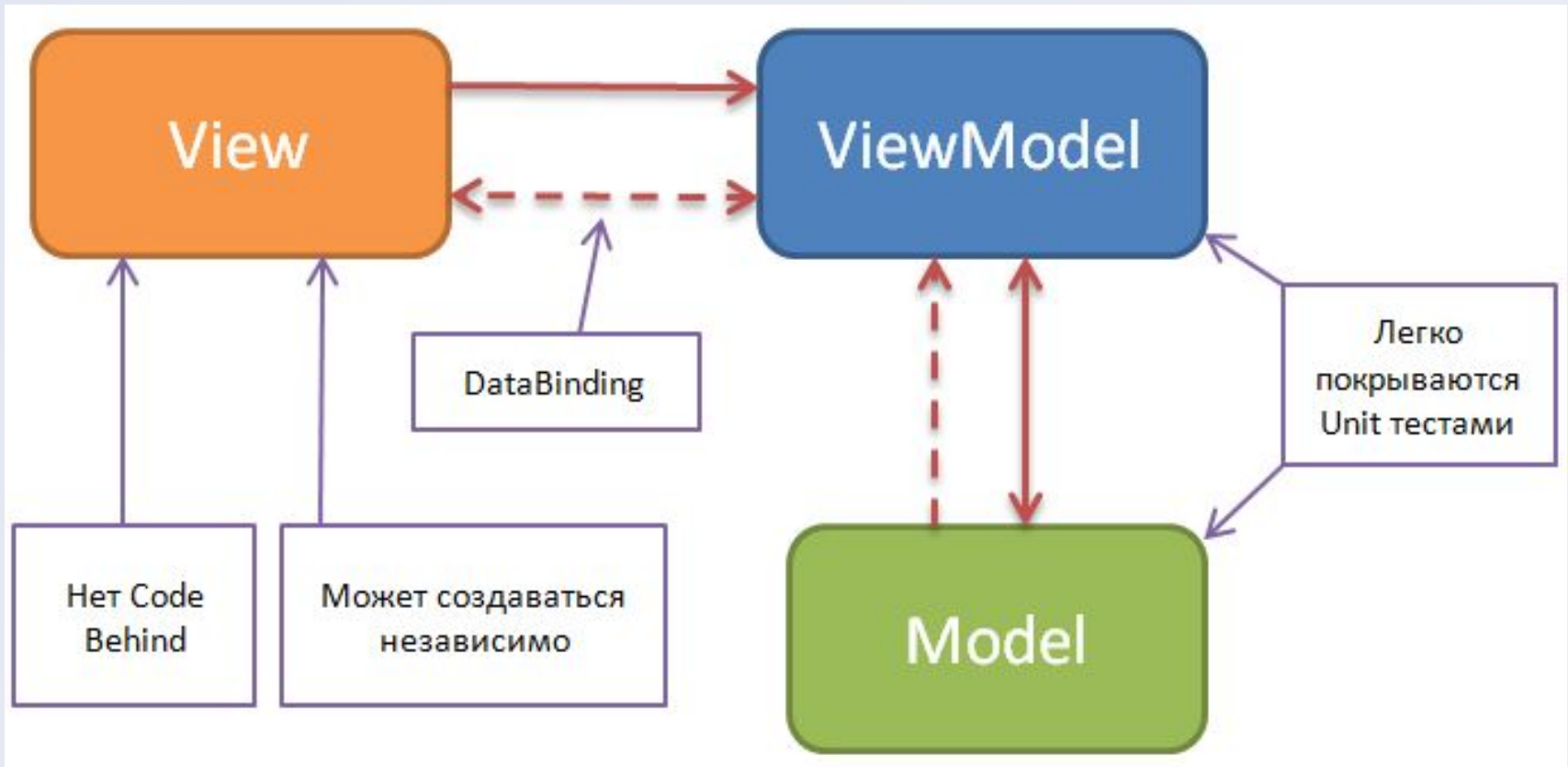
Концепция «связывания данных», позволяет связывать данные с визуальными элементами в обе стороны напрямую. Следовательно при использовании этого приема применение

паттерна [MVC](#) Следовательно при использовании этого приема применение паттерна MVC становится крайне неудобным из-за того, что привязка данных к представлению напрямую не укладывается в концепцию [MVC](#) Следовательно при использовании этого приема

применение паттерна MVC становится крайне неудобным из-за того, что привязка данных к представлению напрямую не укладывается в концепцию MVC/[MVP](#).



Структура MVVM



Реализации во Flex (Model)

```
package Models
{
    public class User
    {
        private var _name:String;

        public function get Name():String
        {
            return _name;
        }

        public function set Name(value:String):void
        {
            _name = value;
        }

        private var _password:String;

        public function get Password():String
        {
            return _password;
        }

        public function set Password(value:String):void
        {
            _password = value;
        }
    }
}
```



Реализации во Flex (View)

Register User

User name: *

Password: *

Confirm Password: *

Связывание данных во Flex

```
[bindable]
public class RegisterUserViewModel : extends EventDispatcher
{
    public function RegisterUserViewModel()
    {
        _user = new User();
    }

    private var _user : User;

    [Bindable(event="UserNameChanged")]
    public function get UserName():String
    {
        return _user.Name;
    }

    public function set UserName(value:String):void
    {
        if (_user.Name == value) return;
        _user.Name = value;
        super.dispatchEvent(new Event("UserNameChanged"));
    }
}
```

```
<s:TextInput id="txtUserName" text="@{vm.UserName}"/>
```



Реализации во Flex (ViewModel свойства)

```
private var _user : User;

[Bindable(event="UserNameChanged")]
public function get Username():String
{
    return _user.Name;
}

public function set Username(value:String):void
{
    if (_user.Name == value) return;
    _user.Name = value;
    super.dispatchEvent(new Event("UserNameChanged"));
}

private var _confirmPassword : String;

[Bindable(event="ConfirmPasswordChanged")]
public function get ConfirmPassword():String
{
    return _confirmPassword;
}

public function set ConfirmPassword(value:String):void
{
    if (_confirmPassword == value) return;
    _confirmPassword = value;
    super.dispatchEvent(new Event("ConfirmPasswordChanged"));
}
```



Реализации во Flex (View MXML)

```
<fx:Declarations>
    <ViewModels:RegisterUserViewModel id="vm"/>
</fx:Declarations>
<s:Panel title="Register User" top="-1" width="400" height="301" horizontalCenter="0">
    <s:VGroup width="100%" height="167" horizontalAlign="center">
        <mx:Form width="80%">
            <mx:FormItem label="User name:" required="true">
                <s:TextInput text="@{vm.UserName}"/>
            </mx:FormItem>
            <mx:FormItem label="Password:" required="true">
                <s:TextInput text="@{vm.Password}"/>
            </mx:FormItem>
            <mx:FormItem label="Confirm Password:" required = "true">
                <s:TextInput text="@{vm.ConfirmPassword}"/>
            </mx:FormItem>
        </mx:Form>
        <s:Button label="Register" click="{vm.RegisterUserCommand()}" />
    </s:VGroup>
</s:Panel>
```



Реализации во Flex (ViewModel команды)

```
<s:Button label="Register" click="{vm.RegisterUserCommand()}" />
```

```
public function RegisterUserCommand():void
{
    if(Password != ConfirmPassword){
        Alert.show("Passwords mismatch");
    }else{
        //save user model
        ...

        Alert.show("Registration completed");
    }
}
```



Обмен сообщениями (Message)

```
public class Message
{
    //Передаваемый контент
    private var _content:Object;
    //Тип сообщения
    private var _messageType:String;
    //Отправитель
    private var _sender:Object;
    //Получатель
    private var _target:Object;
```



Обмен сообщениями (Messenger)

```
public interface IMessenger
{
    function Register(recipient:Object,
messageType:String, action:Function):void;
    function Send(message: Message):void;
    function SendTo(message: Message, targetType :
String):void;
    function UnRegister(recipient:Object,
messageType:String):void;
    function UnRegisterAll(recipient:Object):void;
}
```



Обмен сообщениями (Example)

Отправка сообщения:

```
var message:Message = new Message(MessageType.ChangeScreenCommand,  
ApplicationStates.CharacterDetails, this);  
Messenger.Default.Send(message);
```

Подписка на сообщения:

```
Messenger.Default.Register(this,  
MessageType.ChangeScreenCommand, OnScreenChangeCommand);
```

```
private function OnScreenChangeCommand(message: Message):void  
{  
    setCurrentState(message.Content as String, true);  
}
```



Спасибо за внимание

