

Skinning no Flex 4

Arquitectura Spark

www.webfuel.pt
www.riapt.org



João Saleiro

CTO @ Webfuel

Engenheiro de Software (IUL/ISCTE)

Formador com certificação profissional

Adobe Community Professional

Adobe Certified Expert in Flex and Air

Co-fundador do RiaPT

Contacto

joao.saleiro@webfuel.pt

www.webfuel.pt
www.riapt.org

Showcase

Alguns dos nossos projectos...

www.webfuel.pt
www.riapt.org



Forex Trading Platform

Showcase

Airgile → <http://airgile.com>

www.webfuel.pt
www.riapt.org

Designers vs Developers

www.webfuel.pt
www.riapt.org

Designers + Developers

www.webfuel.pt
www.riapt.org

Arquitectura Spark

Componentes com o código do seu **comportamento** perfeitamente separado do código que define o seu **aspecto gráfico**

Arquitectura Spark

Comportamento (implementado pelo developer):

Colocado num ficheiro Actionscript (.as) sendo uma classe que deriva de

`spark.components.supportClasses.SkinnableComponent`

Aspecto gráfico (implementado pelo designer):

Definido num ficheiro MXML (.mxml) que deriva de `spark.components.supportClasses.Skin`

Arquitectura Spark

SkinnableComponent:

Vai definir as funcionalidades do componente e os elementos que precisa na Skin, mas não o seu aspecto

Skin:

Vai definir o aspecto de cada um dos elementos, mas não as suas funcionalidades

Exemplo: SkinnableComponent

```
public class MouseChase extends SkinnableComponent
{
    public function MouseChase()
    {
        addEventListener(Event.ENTER_FRAME, enterFrameHandler);
    }

    [SkinPart]
    public var chaser:UIComponent;

    [SkinPart]
    public var stopFollowingChk:ToggleButtonBase;

    protected function enterFrameHandler(event:Event):void
    {
        if (stopFollowingChk && stopFollowingChk.selected)
            chaser.move(chaser.x + (mouseX - chaser.x) / 5, chaser.y + (mouseY -
chaser.y) / 5)
        }
    }
}
```

Exemplo: Skin

```
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">

  <fx:Metadata>[HostComponent("MouseChase")]</fx:Metadata>

  <s:Group id="chaser" width="20" height="20">
    <s:Ellipse
      left="0" right="0" top="0" bottom="0">
      <s:fill><s:SolidColor color="#0000FF"/></s:fill>
    </s:Ellipse>
  </s:Group>

  <s:CheckBox id="stopFollowingChk"
    label="Follow?"
    selected="true"/>

</s:Skin>
```

Objectivos

Separar os papeis do developer do "*devigner*"

Reutilizar Skins

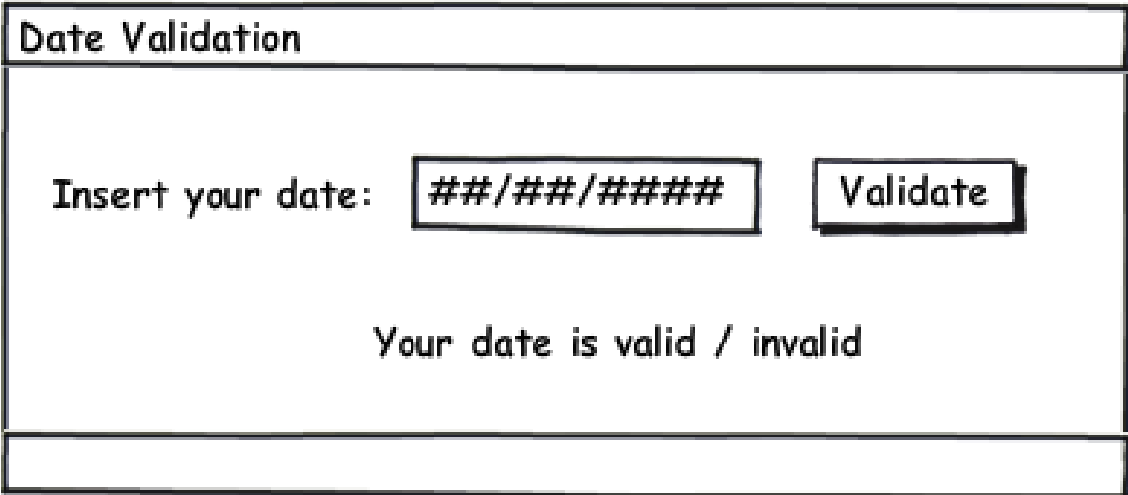
Diminuir a complexidade dos componentes

Introduzir ferramentas para criar Skins (Catalyst)

Criar um custom SkinnableComponent passo a passo

- Definir os elementos do componente (SkinParts)
- Implementar o comportamento
- Criar a Skin
- Reagir ao utilizador (eventos)
- Colocar dados na Skin
- Criar estados da skin (SkinStates)
- Criar transições (animações)

Exemplo: componente para validar datas



The mockup shows a rectangular box with a title bar at the top containing the text "Date Validation". Below the title bar, the text "Insert your date:" is followed by a text input field containing the placeholder "##/##/####". To the right of the input field is a button labeled "Validate". Below these elements, the text "Your date is valid / invalid" is centered. At the bottom of the box, there is a horizontal line representing a footer or separator.

created with Balsamiq Mockups - www.balsamiq.com

SkinParts

São os elementos visuais que compõem um SkinnableComponent

www.webfuel.pt
www.riapt.org

SkinParts

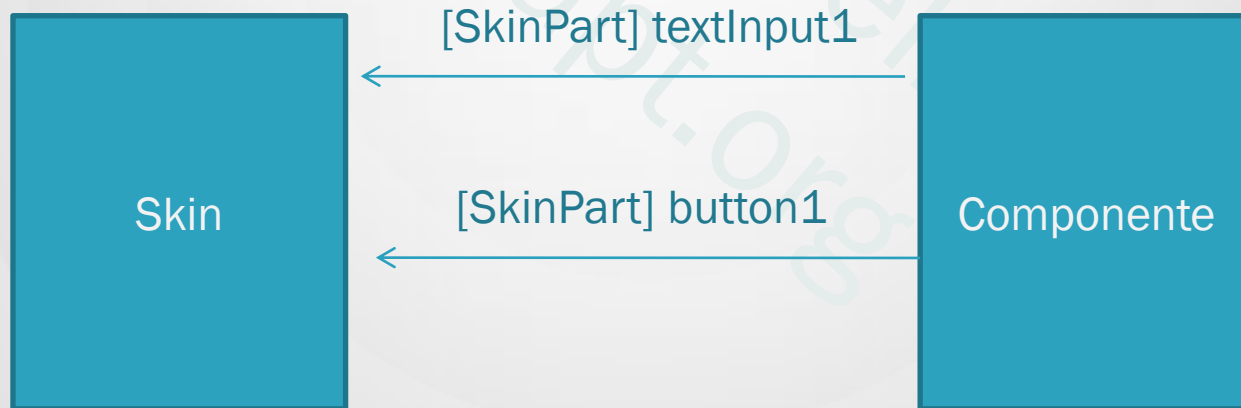
Definem-se através da metadata
[SkinPart(required="true|false")]

```
[SkinPart(required="true")]
```

```
public var chaser:UIComponent;
```

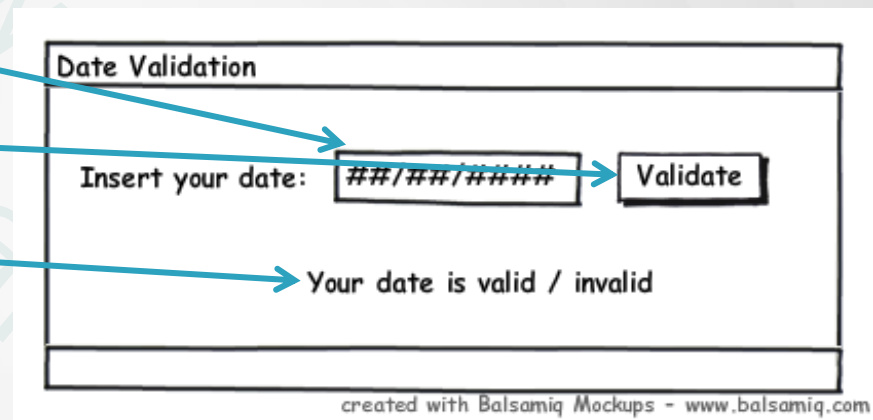
SkinParts

O Flex 4 utiliza as SkinParts para fazer a ligação dos elementos entre o componente e a skin



1. Definir SkinParts

dateTxt:TextInput
submitBtn:Button
resultLbl:Label



1. Definir SkinParts

```
public class DateValidatorComponent extends SkinnableComponent
{
    [SkinPart(required="true")]
    public var dateTxt:TextInput;

    [SkinPart]
    public var submitBtn:Button;

    [SkinPart(required="true")]
    public var resultLbl:Label;
}
```

2. Implementar o comportamento

```
public var errorMessage:String = "";  
  
public function validateDate():void  
{  
    var result:ValidationResultEvent = new  
DateValidator().validate(dateTxt.text);  
  
    if (result.type == ValidationResultEvent.INVALID)  
    {  
        errorMessage = result.message;  
        return;  
    }  
  
    errorMessage = "";  
}
```

Criar a Skin

```
<s:Button  
  id="myBtn"  
  label="Button"  
  skinClass="" />
```

s:Application>

- Create Skin...
- AccordionHeaderSkin - mx.skins.spark
- BlueButtonSkin - exemplo2
- ButtonBarFirstButtonSkin - mx.skins.spark
- ButtonBarLastButtonSkin - mx.skins.spark
- ButtonBarMiddleButtonSkin - mx.skins.spark
- ButtonSkin - spark.skins.spark
- ColorPickerSkin - mx.skins.spark
- ComboBoxButtonSkin - spark.skins.spark
- DataGridHeaderBackgroundSkin - mx.skins.spark
- DataGridHeaderSeparatorSkin - mx.skins.spark
- DateChooserNextMonthSkin - mx.skins.spark

Criar a skin

```
<s:Button
```

```
    label="Button"
```

```
    skinClass="exemplo2.GrayButtonSkin" />
```

www.webfuel.pt
www.riapt.org

3. Criar a Skin

```

<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">

  <fx:Metadata>
    [HostComponent("exemploFinal.DateValidatorComponent")]
  </fx:Metadata>

  <s:VGroup horizontalAlign="center">
    <s:HGroup verticalAlign="middle">
      <s:Label text="Insert your date:"/>
      <s:TextInput id="dateTxt" width="100"/>
      <s:Button id="submitBtn" label="Validate"/>
    </s:HGroup>
    <s:Label id="resultLbl"/>
  </s:VGroup>
</s:Skin>

```

“Desenhar” numa Skin

```
<s:Rect id="lowligh" left="1" right="1" top="1" bottom="1" radiusX="2">  
  <s:fill>  
    <s:LinearGradient rotation="270">  
      <s:GradientEntry color="0x000000" ratio="0.0" alpha="0.0627"/>  
      <s:GradientEntry color="0x000000" ratio="0.48" alpha="0.01"/>  
      <s:GradientEntry color="0x000000" ratio="0.48001" alpha="0" />  
    </s:LinearGradient>  
  </s:fill>  
</s:Rect>
```

“Desenhar” numa Skin

```
<s:BitmapImage id = "background"
source = "@Embed(source='GrayButtonUp.png', scaleGridLeft='2',
scaleGridTop='2', scaleGridRight='7', scaleGridBottom='32')"
source.over = "@Embed(source='GrayButtonOver.png', scaleGridLeft='2',
scaleGridTop='2', scaleGridRight='7', scaleGridBottom='32')"
source.down = "@Embed(source='GrayButtonDown.png', scaleGridLeft='2',
scaleGridTop='2', scaleGridRight='7', scaleGridBottom='32')"
left = "0"
right = "0"
height = "100%"/>
```

Reagir ao utilizador (eventos)

Opção 1: a skin trata dos eventos, e chama os métodos no SkinnableComponent - *hostComponent.method(arguments)*

Opção 2: O SkinnableComponent associa por actionscript eventlisteners aos elementos da Skin –
element.addEventListener(...)

4. Reagir ao utilizador (opção 1)

```
<s:Button id="submitBtn"  
          label="Validate"  
          click="hostComponent.validateDate()" />
```

Reagir ao utilizador (opção 2)

Em teoria bastaria fazer no SkinnableComponent:
`submitBtn.addEventListener(MouseEvent.CLICK,
submitBtn_clickHandler);`

Mas “onde” colocar esse código? No construtor? A Skin é anexada ao componente *depois* deste ter sido criado, logo o construtor não é o local ideal...

Reagir ao utilizador (opção 2)

Solução: fazer override ao método `partAdded`. Este método é chamado sempre que um elemento da Skin é anexado ao componente:

```
protected override function partAdded(partName:String,  
instance:Object):void  
{  
    if (instance == submitBtn)  
    {  
        submitBtn.addEventListener(MouseEvent.CLICK,  
submitBtn_clickHandler);  
    }  
}
```

4. Reagir ao utilizador (opção 2)

// Não esquecer de implementar também o partRemoved

```
protected override function partAdded(partName:String,  
instance:Object):void
```

```
{  
    if (instance == submitBtn)  
    {  
        submitBtn.addEventListener(MouseEvent.CLICK,  
submitBtn_clickHandler);  
    }  
}
```

```
protected function submitBtn_clickHandler(event:MouseEvent):void
```

```
{  
    validateDate();  
}
```

Colocar os dados na Skin

Opção 1: a skin vai buscar os dados usando *bindings*

Opção 2: O SkinnableComponent associa por Actionscript os dados ao elemento na Skin

5. Colocar os dados na Skin (opção 1)

No SkinnableComponent:

```
[Bindable]
```

```
public var errorMessage:String = "";
```

Na Skin:

```
<s:Label id="resultLbl"  
text="{hostComponent.errorMessage}"/>
```

5. Colocar os dados na Skin (opção 2)

No SkinnableComponent:

```
protected override function partAdded(partName:String,  
instance:Object) :void
```

```
{  
    if (instance == resultLbl)  
    {  
        resultLbl.text = errorMessage;  
    }  
}
```

```
public function set errorMessage(value:String) :void
```

```
{  
    _errorMessage = value;  
    if(resultLbl)  
    {  
        resultLbl.text = value;  
    }  
}
```

Uma “pequena” melhoria

O nosso designer teve uma ideia: e se a validação ocorresse à medida que o utilizador escreve?

O submitBtn é um SkinPart não required, certo? Então que tal se retirarmos o submitBtn da Skin, e fizermos:

```
<s:HGroup verticalAlign="middle">
    <s:Label text="Insert your date:"/>
    <s:TextInput id="dateTxt"
        width="100"
        change="hostComponent.validateDate()" />
</s:HGroup>
```

Criar estados da Skin (SkinStates)

Uma Skin pode conter vários estados (diferentes representações visuais).

Por exemplo, um Button possui os estados "up", "down", "over", "disabled" (entre outros)

O nosso componente poderá conter os estados "normal", "valid", e "invalid"

Criar estados da Skin (SkinStates)

Passo 1: definir os estados através da metadata [SkinState]

Passo 2: implementar o método
`getCurrentSkinState():String;`

Passo 3: indicar ao Flex os momentos de mudança de estado (invalidando o estado)

Passo 4: adicionar os estados à Skin

Passo 5: definir o aspecto gráfico para cada estado

6. Criar estados da Skin

Passo 1: definir os estados através da metadata [SkinState]

```
[ SkinState ("normal") ]
```

```
[ SkinState ("valid") ]
```

```
[ SkinState ("invalid") ]
```

```
public class DateValidatorComponent extends  
SkinnableComponent
```

6. Definir estados da Skin

Passo 2: implementar o método `getCurrentSkinState()`

```
protected var state:String = "normal";  
override protected function getCurrentSkinState():String  
{  
    return state;  
}
```

6. Definir estados da Skin

Passo 3: indicar ao Flex os momentos de mudança de estado (invalidando o estado)

```
public function validateDate():void
{
    var result:ValidationResultEvent = new DateValidator().validate(dateTxt.text);

    if (result.type == ValidationResultEvent.INVALID)
    {
        errorMessage = result.message;
        state = "invalid";
    }
    else
    {
        errorMessage = "";
        state = "valid";
    }
    invalidateSkinState();
}
```

6. Definir estados da Skin

Passo 4: adicionar os estados à Skin

```
<s:states>  
  <s:State name="normal" />  
  <s:State name="valid" />  
  <s:State name="invalid" />  
</s:states>
```

6. Definir estados da Skin

Passo 5: definir o aspecto gráfico para cada estado

```
<s:HGroup verticalAlign="middle">  
    <s:Label text="Insert your date:"/>  
    <s:TextInput id="dateTxt"  
        width="100" change="hostComponent.validateDate()" />  
    <s:Ellipse width="10" height="10" visible.normal="false">  
        <s:fill>  
            <s:SolidColor color.valid="#00FF00"  
                color.invalid="#FF0000" />  
        </s:fill>  
    </s:Ellipse>  
</s:HGroup>
```

Uma (minúscula!) intervenção do designer...

Ora o nosso responsável pelos interfaces viu o componente e disse "Yuck!!"

E pegou na fantástica e elaborada Skin que eu tinha criado, e....

... Antes do designer

Insert your date: ●



... Depois do designer

Criar transições (animações)

Definem-se através de uma Array de <s:Transition> na Skin
Cada transição pode levar um ou mais efeitos diferentes

```
<s:transitions>  
  <s:Transition fromState="normal"  
                toState="valid" effect="..." />  
  <s:Transition fromState="invalid"  
                toState="valid" effect="..." />  
</s:transitions>
```

7. Criar transições (animações)

```
<s:transitions>
  <s:Transition fromState="*"
                toState="*"
                autoReverse="true">
    <s:effect>
      <s:Parallel duration="850">
        <s:AnimateColor targets="{ [color] }"/>
        <s:Fade targets="{ [dateLbl, dateTxt] }"/>
      </s:Parallel>
    </s:effect>
  </s:Transition>
</s:transitions>
```

7. Criar transições (animações)

```

<s:transitions>
  <s:Transition fromState="*"
               toState="*"
               autoReverse="true">
    <s:effect>
      <s:Parallel duration="850">
        <s:AnimateColor targets="{ [circlecolor] }"/>
        <s:Fade targets="{ [dateLbl, dateTxt] }"/>
      </s:Parallel>
    </s:effect>
  </s:Transition>
</s:transitions>

```

* Significa qualquer estado

Vamos animar a cor do círculo

E fazer um fade out das caixas de texto

Flash Catalyst

Ferramenta para criar Skins a partir de mockups em Photoshop ou Illustrator

Mais info:

<http://www.adobe.com/products/flashcatalyst/>

Questões?

joao.saleiro@webfuel.pt

www.webfuel.pt
www.riapt.org