Flex4 皮肤机制在拓扑图形系统中的应用①

钱 蔚

(国网电力科学研究院, 南京 210000)

摘 要: Flex4 的重要改进之一是提供了新的组件和皮肤架构. 皮肤架构建立在逻辑和组件视觉元素的清晰分离的基础上, 为设计人员和开发人员之间创造一个更顺畅的工作流程. 主要研究了皮肤机制在拓扑图形系统中的应用. 首先介绍了皮肤机制的原理, 然后介绍了使用方法, 最后结合通用拓扑图形系统的实现详细介绍了皮肤机制在实际应用中的优点.

关键词: Flash Player; Flex; 皮肤; FXG; 拓扑图

Application of Flex4 Skin Mechanism to Topology Graphic System

QIAN Wei

(State Grid Electric Power Research Institute, Nanjing 210000, China)

Abstract: One of the important improvements of Flex 4 is to provide a new component and skin architecture. Skin architecture is based on a clear separation of logic and components of visual elements, and creates a smoother work flow between designers and developers. This paper mainly studies the application of skin mechanism in topology graphic system. Firstly, this paper introduces the principle of skin mechanism, then describes the usage method, finally introduces the advantage of skin mechanism according to the realization of a general topology graphic system.

Key words: flash player; flex; skin; FXG; topological graph

1 引言

所谓皮肤, 是指软件的界面, 即软件的可视外观. 为了满足用户更好的使用体验和个性需要, 我们常用的输入法、播放器、浏览器甚至是操作系统中都加入了皮肤或主题的功能. 软件的皮肤和功能是分离的, 开发者专注于功能的开发, 可视化设计者为软件定制出不同风格的皮肤, 用户根据自己的爱好来自由选择皮肤而不会影响到软件的功能, 这对于以往那些千篇一律的软件外观来说是一个很大的进步.

Flex 4 的主要主题之一是思维设计(Design in mind), 皮肤则是这个主题的重要组成部分. Flex 4 中可以更容易地完全改变应用程序的外观和体验. 新的皮肤架构建立在 Flex 4 的一些架构改动以及逻辑和组件视觉元素清晰分离的基础上[1]. 正因为如此, 在 Flex 4 的组件中没有直接包含任何有关他们的视觉外观的信息. 所有这些信息包含在皮肤文件中, 皮肤文件遵循 FXG 和新的 states 语法, 使用 MXML 编写, 这样它们就更容易地被阅读和编写, 同时也更易于工具访问.

2 皮肤机制

皮肤机制的最大特点是显示和逻辑的分离, Flex4 新增了 SkinnableComponent 组件, 该组件继承子 UIComponent 并扩展了 Skin 相关的功能; 同样 Skin 类也继承自 UIComponent. 皮肤机制的实现依赖主组件类和皮肤文件(外观类).

主组件类是 SkinnableComponent 的子类, 该类封装了组件的核心行为. 这包括定义组件调度的事件、组件表示的数据、接通作为主组件组成部分的任何子组件以及管理和跟踪内部组件状态.

与主组件类对应的是一个继承自 Skin 的外观类, 它负责管理与组件的可视外观相关的一切内容, 包括

① 收稿时间:2013-12-04;收到修改稿时间:2013-12-24

图形、布局、表示数据、更改不同状态中的外观以及从一个状态过渡到另一个状态. 在 Flex3 模型中, Flex 组件外观是只负责组件的图形部分的资源. 更改组件外观的任何其他方面, 如布局或状态可视化, 需要将组件子类化并直接编辑 ActionScript 代码. 而在 Flex4 模型中, 所有这一切都可以在外观类中以声明方式进行定义.

3 拓扑图形系统

拓扑学是几何学中一个分支, 它是研究几何图形在连续改变形状时还能保留不变的一些特点, 它只考虑物体之间的位置关系而不考虑它们的距离和大小. 拓扑图也具有上述的特点. 因此, 有人称之为 “相对位置图”. 拓扑变换在各种类型的空间研究中有着广泛的应用.

在拓扑图形系统中, 节点的变现形式往往是多样的, 以流程图为例, 其中可能含有流程、判定、子流程、开始、结束等多种节点, 他们的形状和特性各不相同. 传统的图形系统中, 我们要实现这几种节点, 通常会从基类中继承出多个子类. 为了实现节点, 我们甚至还需要实现一系列的基本图形对象(直线、曲线、矩形、圆、弧线、图片等), 用这些基本图形对象组合才能形成节点. 但如果我们剥离出其中的业务特性, 就会发现其中他们是同一类节点[2-6]. 使用皮肤机制, 我们可以简化图形系统的结构, 所有和显示有关的内容都由皮肤实现, 图形系统只需要考虑核心功能, 比如节点之间的连接关系, 图形的布局, 拓扑分析功能等等, 这样就大大简化了拓扑图形系统的设计.

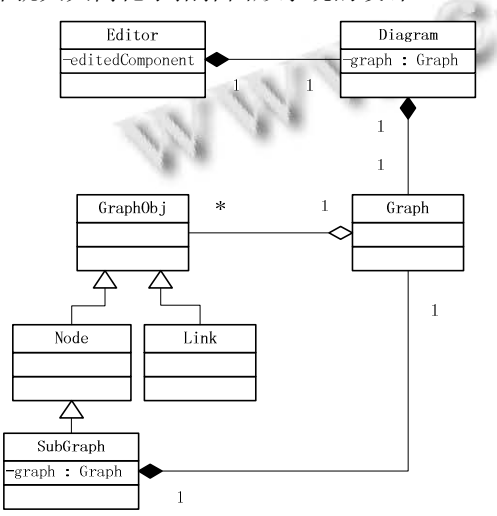

图 1 拓扑图形系统类图

简化的拓扑图形系统只包含了核心架构: 一般会具备以下的构成要素: 编辑器(Editor), 图纸(Diagram), 图形(Graph), 节点(Node), 连线(Link), 子图(SubGraph)等.

图形(Graph)是纯粹的图形概念, 是一个放置图形对象的容器. Graph 一般作为图纸或子图的组成部分而存在.

图纸(Diagram)通过 Graph 来展示图形. 支持图形浏览, 如选择、移动、缩放、画面平移等.

编辑器(Editor)处理用户交互并提供一系列图形编辑功能, 如图形对象的增加、修改、删除等.

GraphObj 是图形对象的基类. 从该类中继承出的节点(Node)、连线(Link)和子图(SubGraph)构成了图形的全部内容.

子图(SubGraph)继承自节点, 拥有节点的所有特征, 子图中的容器(Graph)中可以放置图形对象. 子图可以嵌套.

4 皮肤机制在图形系统中的应用

4.1 皮肤的实现

拓扑图形系统中使用皮肤的组件包括: 图纸(Diagram), 节点(Node), 连线(Link), 子图(SubGraph)

图纸类使用皮肤可以丰富图形的显示内容, 用户可以根据需要定制自己的皮肤, 例如在图纸中加入网格或背景图片. 节点、连线和子图使用皮肤可以灵活定制图形对象的显示样式. 以节点为例, 创建支持皮肤的义组件需要以下步骤.

首先需要定义一个主组件类 Node, 该类继承自 SkinnableComponent.

其次需要定义皮肤类. 皮肤类继承自 Skin, 使用 MXML 来制作; 皮肤中主要需要指定主对象:

[HostComponent("com.xxx. Node")]

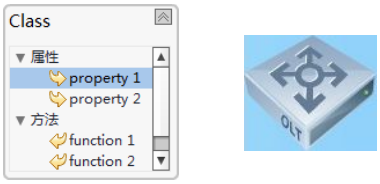

图 2 节点的不同显示样式

皮肤的具体显示内容使用 FXG 语法来编写, 也可以使用 Adobe illustrator 绘制后导入. 下图展示了两个

不同显示样式的节点(左图为手工编写的类图节点，右图为 Adobe illustrator 绘制的通信设备节点).

最后需要绑定组件和皮肤. 通常有以下几种方式:

在 css 中指定

```
Node{
skinClass:ClassReference("com.xxx. NodeSkin");}
```

在 mxml 中指定

```
<Node skinClass="com.xxx. NodeSkin " />
```

在 as 中使用 setStyle 指定

```
Node.setStyle( "skinClass", Class(NodeSkin));
```

4.2 动态皮肤的加载

动态皮肤可以在不修改主程序的前提下灵活修改图形的显示样式. 只需要了解图形对象的少量基本信息，任何人都可以发布一个皮肤库. 而开发者也有能力来控制哪个皮肤可以被替换，哪个皮肤只能使用原来的皮肤. 将制作好的皮肤放到主程序可以访问到位置即可，这样可以有效的避免主程序的频繁修改，并保持主程序的体积不会太大. 具体的实现方式如下:

首先为皮肤资源建立一个外壳 swf，并将其放置于主程序可以访问到的位置. 有了这个外壳 swf，主程序就可以实时加载其中的皮肤资源.

其次在主程序中使用 Loader 加载外壳 swf，并且将其转化为类. 以下是一些关键行:

```
var loader:Loader = new Loader();
var request:URLRequest = new URLRequest(swf);
var context:LoaderContext = new LoaderContext();
context.applicationDomain=ApplicationDomain.currentDomain;
loader.load(request,context);
```

然后从加载的 swf 中生成类，并实例化

```
var class:Class = loader.contentLoaderInfo.applicationDomain.getDefinition(className) as Class;
var obj:Object = new class();
```

最后利用 setStyle 为对象指定皮肤即可.

4.3 显示与编辑的分离

拓扑图形系统需要提供编辑功能以实现节点和连线的编辑，例如，节点大小的修改、连线外形的修改、连接关系的修改，通常的操作方式是直接针对图形对象本身进行操作. 这种操作方式具体实现时需要为每一类图形对象添加编辑相关的功能，实现比较繁琐. 而借助于皮肤，我们可以实现显示和编辑的分离. 我们可以为同一类图形对象(例如节点)设定编辑皮肤，编辑皮肤中主要包括了节点的显示轮廓、编辑控制手柄(handle)等，只有在图形对象进入编辑状态时编辑皮肤才会出现，例如当鼠标移动至节点上的时候. 此时我们可以使用鼠标控制编辑手柄进行编辑操作，所有的编辑操作都针对编辑皮肤进行，编辑皮肤的宿主节点再根据编辑皮肤的修改信息修改自身.

在拓扑图形系统的设计中，我们有时会遇到这样的问题，某种类型节点的功能是另一种节点的子集，例如我们有两种类型的拓扑节点，一种拓扑节点具有全部的编辑功能，包括节点外形的编辑能力、多方向连接拓扑连线的能力等，而一种节点可能不需要修改外形，只需要连接能力即可，此时我们可以设计两种编辑皮肤，一种具备全部功能，而一个只具备连接边界能力的编辑皮肤，在其中屏蔽修改外形的控制手柄. 使用两种编辑皮肤与同一类节点结合，即可实现不同编辑能力的图形对象. 下图展示了不同编辑能力的两种皮肤. 左图不具备编辑能力，中图具备部分编辑能力，而右图具备全部编辑能力.

图 3 不同编辑能力的节点

4.4 皮肤的组合

皮肤继承自 Skin 类，而 Skin 又继承自可视元素的容器基类 Group，因此皮肤具备组合能力，可以进行灵活修改.

当拓扑图形系统在监控系统中使用时，一个很重要的需求是告警功能，各种监控系统对告警功能的要求是不同的，简单的情况下也许只需要以高亮方式显示出节点即可，而另一些可能要求除了高亮展示、还需要显示出告警信息，例如错误的总数，错误的类型等等，直接将这些告警显示编写在代码中显示是不够灵活，后期修改比较麻烦. 而使用皮肤则可以很灵活的修改告警. 我们可以制作多种告警皮肤、将这些告警皮肤添加到节点的皮肤中. 后期甚至可以由二次开发人员完成告警皮肤的制作.

4.5 子图的实现

子图 SubGraph 继承自图形节点类 Node. 其中包含了一个放置图形对象的容器 Graph，借助于皮肤的

状态 states, 我们可以实现出一个可以收缩和展开的子图. 首先我们需要在 SubGraph 中设置一个标识子图收缩或展开的属性 collapsed, 并重载 getCurrentSkinState 方法, 定义子图的多种状态. 例如, 我们可以获得如下的两个状态"collapsed"和 “expanded”:

```
protected override function getCurrentSkinState():
String{
    return (collapsed ? "collapsed" : “expanded”);
}
```

在子图的皮肤中我们同样需要定义这两个状态, 并分别为其指定所属的状态组 "Collapsed" 和 "Expanded".

```
<s:states>
    <s:State name="collapsed" stateGroups=
    "Collapsed"/>
    <s:State name="expanded" stateGroups=
    "Expanded"/>
</s:states>
```

然后我们需要为皮肤中的元素设定在不同状态下的可见性 visible, 以及在不同状态下的大小. 例如:

```
<s:Group id="graphGrp"
    visible.Expanded="true"
    visible.Collapsed="false"
    .....
</s:Group>
```

通过这些工作, 我们就实现了一个可以收缩和展开的子图, 只需要通过子图的 collapsed 属性就可以进行控制, 和传统的设计方式相比, 概念更为清晰, 实现更为简洁. 下图展示了子图的展开状态和收缩状态.

图 4 子图的收缩和展开状态

4.6 动画的实现

拓扑图形系统的使用中经常会遇到动画的需求, 例如信息系统中需要在拓扑连线上显示出信息的流动. Flex4 提供了较为完善的动画功能. 借助了皮肤, 我们可以制作出赏心悦目的动画效果. 比如我们需要在一根连接线上实现信息的流动, 可以按以下方式实现:

首先我们定义出动画对象 information, 用来表示信息. 并为其定制皮肤, 皮肤可以是一张栅格图片, 也可以按 FXG 格式定义为矢量图形, 然后为其指定一个取值范围在 0-1 的 position 属性, 用来控制动画对象位置. Position 在 0-1 区间内变化时, 动画对象的位置在连线的起点和终点之间变化. 最后我们需要用到 Flex4 提供的动画效果类

```
<s:Animate id="animation" target="{information}"
duration="delay">
    <s:SimpleMotionPath property="position"
    valueFrom="0.0" valueTo="1.0" >
    </s:SimpleMotionPath>
</s:Animate>
```

这样我们就定义出了信息流动的效果, 使用 animation.play()即可播放动画.

5 结论

本文研究了 Flex4 的皮肤机制, 并基于皮肤机制研究了通用的拓扑图形系统的架构, 结合拓扑图形系统的多个功能模块研究了皮肤机制的实际应用. 实践证明, Flex4 的皮肤机制不仅可以简化系统设计架构、使程序编码更为简便明晰, 而且可以充分支持二次开发, 定制出更加炫目的现实效果.

参考文献

1 Frishberg R. Introducing skinning in Flex 4. Adobe Developer Connection / Flex Developer Center, 2009.
2 纪陵,蒋衍君,施广德.基于 SVG 的电力系统图形互操作研究.电力自动化设备,2011,7:105−109.
3 张海梁,袁荣湘,孙婉胜.基于可视化组件技术的 SCADA 图形子系统开发.自动化与仪表,2006,3:57−60.
4 韩丹,景绍洪,邓丽.监控软件图形组态子系统的研究与设计.济南大学学报(自然科学版),2006,1:55−57.
5 林济铿,覃岭,罗萍萍.基于 Visual Graph 的电力图形系统开发.电力系统自动化,2005,15:73−76.
6 王亚民,一种地理信息系统矢量图形编辑组件.现代图书情报技术,2005,12:67−70.