基于 XML 的交易界面开发语言与运行时引擎

An XML Based Transaction UI Development Language and Its Runtime Engine

孟庆海 鲁 炜 (中国科学技术大学 管理学院 安徽 合肥 230052)

摘 要: 针对银行、证券、邮政等交易应用环境的特定需求,提出了一种基于 XML 的交易界面开发语言——TXUL,实现了交易界面描述与业务逻辑代码的分离,从而降低了交易界面开发与维护的难度,并利用引用机制使界面描述代码的可重用性得到增强。本文详细阐述了 TXUL 及其运行时引擎的设计和实现过程,通过 UML 描述 TXUL 的语法结构并映射为 XML Schema,给出了一个基于 Eclipse RCP 的 TXUL 应用示例。

关键词: XML 交易界面 柜面交易系统 TXUL 引擎

1 引言

图形用户界面是各种信息系统必不可少的组成部分,在基于交易的应用环境中(银行、证券、邮政、电信等行业),交易界面的开发占据了柜面交易系统开发周期的很大比例。传统的开发模式基于各种指令式语言,包括 C++,Java 等,不仅要求开发人员熟悉编程语言的图形库,关注界面生成的全过程,还造成界面代码与业务逻辑代码混合,使系统测试与维护困难,无法适应业务的快速变化。与此相反,声明式界面语言[1]仅要求程序员将界面描述为独立文档,由运行时引擎将这种中间语言解释为图形用户界面。

XML(eXtensible Markup Language)是一种基于标记的结构化语言,具有良好的可扩展性、结构性和数据描述格式的一致性。作为一种元语言,它提供了通用的语法和结构来描述其他与领域相关的语言。近年来,利用 XML 定义声明式界面语言成为研究热点,基于 XML 的界面描述语言(UIDL)大量涌现,例如,佛吉尼亚人机交互技术中心(Center for HCI)提出的 UIML 实现了跨平台的界面描述机制[1],国际信息技术标准委员会(INCITS)以设备无关性为设计目标提出了 AAIML[2],微软提出了 XAML[3] 并将其应用于 Vista 界面及 Silverlight 的开发中。

本文针对交易应用环境的特定业务需求,提出了一种基于 XML 的交易界面开发语言——TXUL,并实现了 J2SE 平台上的运行时引擎,实际应用证明 TXUL 大幅降低了交易界面的开发难度,缩短了开发周期,能够适应业务需求的频繁变化。

2 交易应用环境分析

交易系统 GUI 的应用环境与一般的桌面应用程序有着明显的区别,主要体现在以下几个方面:

(1)对用户输入的内容具有严格的格式要求,账号、户名、金额、日期的输入控件必须具有可定制的格式化功能。

(2)验证机制,系统常常需要在用户界面层面加入输入验证机制,以保证在客户端屏蔽人为操作错误。

(3)对交易处理速度的要求,交易处理人员直接面对客户,这就要求交易界面支持快速操作,减少鼠标与键盘的切换操作,尽可能将交易处理人员的操作限定在数字键盘区域。

(4)组织的业务随市场环境变化,造成交易界面需求变化频繁,这就要求系统支持界面设计与业务处理逻辑的并行开发模式,并具备足够的灵活性和可扩展性以适应业务的迅速变化。

3 TXUL 的设计

3.1 界面元素结构模型

界面元素结构模型是对界面中各种控件以及控件之间的依赖、继承关系的抽象描述,本文采用 UML 对其建模,该模型最终映射为 XML Schema 和引擎中的数据模型。

交易界面包含多种类型的控件元素,它们可以划

分为两种类型:原子型控件和组控件。原子型控件是界面上最小粒度的控件单元,不能包含其他控件,该类型的控件包括:文本框、密码框、单选和复选框、按钮等。组控件可以容纳和组织其他界面元素,包括面板(Panel)、页签组(TabFolder)、页签面板(Tab)、表格等。

所有交易界面元素都具备一个布局数据(LayoutData)属性,组控件容纳原子控件或其它组控件,利用布局管理器(Layout)根据被容纳控件的布局数据来管理被容纳元素的布局和定位。另外,所有界面元素还具有外观属性,包括前景色、背景色、字体、图片等。将这些界面元素所共有的特征概括出来,构成所有界面元素的抽象父类 Structure。

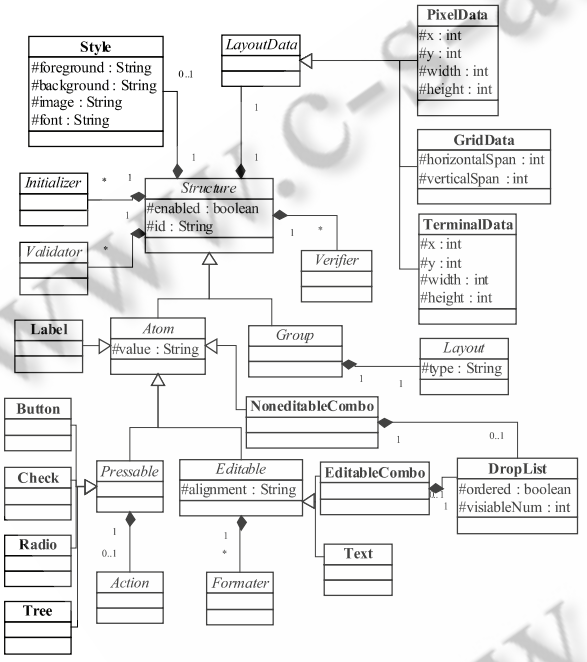

图 1　TXUL 界面元素结构模型

原子型控件包括 Text、Label、Button、EditableCombo 等 8 种实现类。其中文本框(Text)和可编辑下拉列表(EditableCombo)能够接受字符输入,为了满足交易环境中对键盘输入内容的格式化需求,这两种可编辑控件可以包含一个或多个格式化器(Formatter)。根据需求,TXUL 中包含 11 种预定义格式化器,如表 1 所示。这些细粒度的格式化器可以在控件中通过不同的排列和组合来实现交易中需要的文本格式。除预定义格式化器以外,用户可以实现引擎中提供的 Formatter 接口来添加自定义格式化器。

表 1　预定义格式化器

名　称	功　能
CurrencyFormatter	为给定的数字添加货币符号
NumericFormatter	为给定数字添加数字分隔符号
PercentFormatter	将给定数字转化为百分数形式
DateFormatter	将给定字符串格式化为日期形式
TimeFormatter	将字符串转化为时间形式
AccountIDFormatter	根据给定的帐号格式参数格式化
UppercaseFormatter	将符字串中的字母转换为大写
LowercaseFormatter	将字符串中的字母转化为小写
DoubleByteFormatter	将字符串中半角字符转化为全角
SingleByteFormatter	将字符串中全角字符转化为半角
PaddingFormatter	在左侧或右侧填补指定字符使字符串达到指定长度

组控件中 Tab 与 Panel 扩展抽象类 StructureHolder,可以容纳 Structure,而 TabFolder 仅用于容纳 Tab。表格(Table)控件必须通过表格列(TableColumn)来间接纳原子型控件,如果一个列包含一个原子控件,那么该列中的任何一个单元格在编辑状态时均显示该类型的控件。

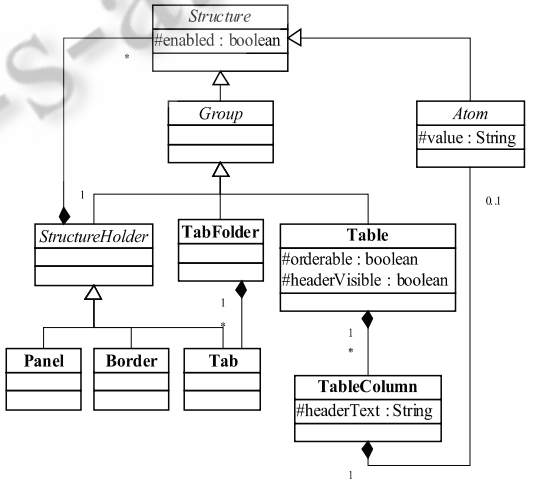

图 2　组控件静态结构

利用 XML 元数据交换(XMI)[4]将以上基于 UML 的界面元素结构模型转化为 XML Schema,把它作为 TXUL 的语法基础。

3.2 业务逻辑

从 TXUL 的界面元素结构模型可以看出,TXUL 的界面描述文档中并不包含业务逻辑代码,但 UI 作为人机交互的界面必须提供响应机制来驱动业务逻辑的执行。根据业务逻辑执行的不同时机,将业务逻辑执行的入口分为 4 种类型:初始化器(Initializer)、输入时验证器(Verifier)、输入后验证器(Validator)和动作(Action),对应的执行过程如图 3 所示。

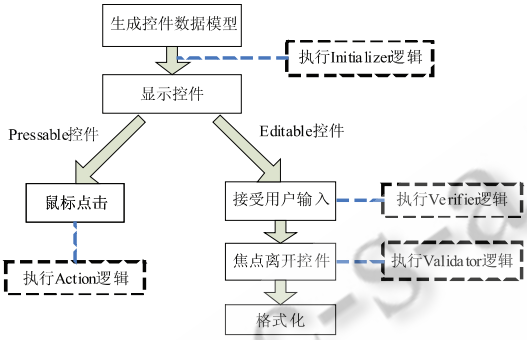

图 3 业务逻辑执行过程

初始化器对应的执行时机是控件的初始化过程,这时执行其中用户自定义的业务逻辑例如从服务器获取控件的初始值等操作,然后显示控件。

对于可编辑(Editable)控件,用户的每次输入操作都会触发输入时验证器的逻辑来对新输入的值进行判断,如果符合输入条件,则用输入的值更新数据模型与控件显示,否则阻止此次输入。与格式化器一样,TXUL 提供多种预定义验证器,例如加载 NumericVerifier 的文本框只能够输入数字。输入时验证器只能保证输入字符的形式正确,所以当结束输入后,焦点离开控件时,控件还需要执行输入后验证器的逻辑,对输入字符串进行整体验证,或结合界面上其他控件的数据进行综合验证,以保证输入字符串的逻辑正确。

对于可点击(Pressable)控件,当鼠标点击时,控件执行动作(Action)对应的业务逻辑。

相应地,引擎提供这四类逻辑入口的接口,具体实现由用户根据业务需求用指令式语言开发,并在 TXUL 文档的标签中声明实现类的类名,引擎在解析 TXUL 文档时通过反射(Reflection)来加载用户自定义实现类,并在上述的四种时机调用业务逻辑。通过这种方法 TXUL 实现了界面描述与业务逻辑的分离。

3.3 TXUL 运行时引擎的设计

可重用性(reusability)能够有效提高系统开发效率、可靠性、一致性和降低代码维护难度[5]。为了增强界面元素定义的可重用性,实现界面元素一次定义多处引用,TXUL 中定义了引用机制。

TXUL 将每个交易界面组织成一个独立的 XML 文档,在一个文档中定义的控件可能需要在多个界面中出现,这就需要 TXUL 的引用机制具有跨文档引用的能力。界面描述文档中,在需要重用界面元素的位置加入 refStruc 标签,同时在 refId 属性中指定被引用界面元素的 id,若被引用元素不在当前文档中,则必须指定其所在文档的 URI,引擎以懒加载的方式处理 refStruc 标签,即在解析 refStruc 标签时,引擎才将 URI 对应的文档解析为数据模型,然后以 id 搜索被引用控件的定义,并将模型转化为界面控件。

4 TXUL 运行时引擎的设计

TXUL 运行时引擎采用 MVC 设计模式基于 J2SE 平台实现,其架构如图 4 所示。

模型部分的数据访问模块利用 JDOM 解析和验证界面文档,并建立从文档到引擎数据模型的映射,处理访问数据模型的请求。控制器部分是引擎的核心,包括 API、模型可视化、逻辑调用器、事件处理等子模块以及业务逻辑接口。引擎 API 提供交易界面生命周期管理、控件数据访问的统一接口,控件的业务逻辑代码可以通过调用引擎的 API 来操作其他控件。模型可视化模块负责处理从数据模型到实际界面控件的转化过程,并利用逻辑调用器加载业务逻辑实现类和格式化器。事件处理模块是事件 – 监听器模型的实现,利用监听器捕捉来自界面的事件,并通过逻辑调用器调用相应的业务逻辑。引擎的视图部分提供了对 Eclipse SWT 界面控件和 Swing 界面控件的支持,但由于这两种界面类型并不兼容,为了屏蔽其中的差异,向控制器部分提供统一的编程接口,视图部分加入一个适配器层建立引擎控件到 SWT、Swing 控件的映射和事件、监听器的封装,这样就使引擎的模型和控制器部分不依赖于具体的界面实现,与 GUI 保持松耦合,保证模型和控制器在 Swing 和 SWT 界面环境下均可重用。

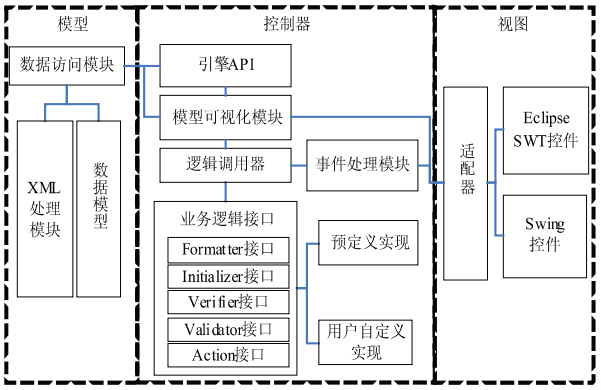

图 4　运行时引擎架构图

5　TXUL 应用示例

以银行柜面业务系统的一个简化的取款交易界面为示例，其 TXUL 文档代码片段如下：

```
<Panel >
  <Layout type = " pixel" / > <！ －－像素布局控制器－－ >
  <Label value = " 取款" > <！ －－取款标签定义－－ >
    <Sytle font = " 宋体,BOLD,18" / >
    <PixelData x = " 22" y = " 16" width = " 52" height = " 28" / >
  </Label >. . . . . .
  <Text id = " accountNo" > <！ －－帐号文本框定义－－ >
    <PixelData x = " 97" y = " 63" width = " 193" height = " 18" / >
    <Verifiers >
      <NumericVerifier intDigit = " 19" / >
<！ －－预定义数字验证器，保证输入不超过 19 个整数位－－ >
    </Verifiers >
    <Formatters >
      <AccountIDFormatter pattern = " #### －#### －#### －#######" / >
      <！ －－预定义帐号格式化器－－ >
    </Formatters >
  </Text >
```

```
  <refStruc refId = " amountPanel"
    uri = " https://9.186.121.16/amountPanel.txul" >
    <PixelData x = " 48" y = " 121" width = " 242" height = " 26" / >
  </refStruc > <！ －－引用 id 为 amountPanel 的控件－－ >
  <Button value = " 取款" > <！ －－取款按钮声明－－ >
    <Action
      implClass = " com.banksample.WithdrawalAction" / >
    <！ －－取款业务逻辑－－ >
  </Button >. . . . . .
</Panel >
```

该文档是取款交易的主界面描述，其中利用 refStruc 标签引用了服务器端另一文档 amountPanel.txul 中 id 为 amountPanel 的控件定义，其代码如下：

```
<Panel id = " amountPanel" >
  <Layout type = " pixel" / >
  <Label value = " 金额" >
    <Sytle font = " 宋体,BOLD,9" foreground = " 102,87,180" / >
    <PixelData x = " 22" y = " 16" width = " 52" height = " 28" / >
  </Label >
  <Text id = " amount" alignment = " right" >
    <PixelData x = " 49" y = " 3" width = " 112" height = " 3" / >
    <Verifiers >
      <NumericVerifier intDigit = " 13" fraDigit = " 2"
        negativeAllowed = " false" / >
<！ －－数字验证器，允许 13 位整数，2 位小数的正数－－ >
    </Verifiers >
    <Formatters >
      <NumericFormatter / >
      <CurrencyFormatter currency = " ￥" / >
    </Formatters >
  </Text >
</Panel >
```

图 5 基于 Eclipse SWT 的取款交易界面

在该示例中,柜面业务系统客户端基于 Eclipse RCP 实现,TXUL 运行时引擎作为交易界面的控制核心将部署于服务器端的 TXUL 文档解析为 Eclipse SWT 界面。取款交易界面如图 5 所示,其中左侧导航视图和右侧"交易"视图均由引擎解析生成,上述代码对应于"交易"视图中的界面。

6 结论

如何让业务系统适应需求的快速变化已成为企业信息化领域的研究热点。本文所提出的交易界面标记语言 TXUL 及运行时引擎实现了交易界面设计与业务逻辑的分离,以支持界面与逻辑的并行开发模式,同时利用引用机制使界面代码具备高度的可重用性。目前,TXUL 已在实际项目中得到应用,表现出良好的可扩展性与灵活性,大幅降低了系统开发的风险与成本。

参考文献

1 Constantios P. UIML: A Device – Independent User Interface Markup Language [Ph. D. Thesis]. Blacksburg, VA: Virginia Polytechnic Institute and State University, 2000.

2 Gottfried Z, Gregg V. Prototype implementations for a universal remote console specification. In: Conference on Human Factors in Computing Systems. New York: ACM, 2002. 510 – 511.

3 Charles P. Code Name Avalon: Create Real Apps Using New Code and Markup Model. Microsoft MSDN Magazine, 2004, 19 (1): 21 – 24.

4 Ayesha M. Design XML schemas using UML. 2003 – 02. http://www.ibm.com/developerworks/library/x – umlschem/.

5 Bertrand M. Object – Oriented Software Construction. Second Edition. 2nd ed., NJ: Prentice Hall PTR, 1997. 68 – 73.