计算机科学 2005Vol. 32№. 11

GUI抽象描述模型研究

张 鹏 徐 鹏

(清华大学计算机科学与技术系知识工程研究室 北京 100084)

摘 要 图形用户界面(GUI)是各种计算机应用的一个重要组成部分。但是越来越多的编程语言和GUI工具包使得开发跨平台的GUI也变得越来越繁琐。同时,日趋庞大和复杂的数据,也迫使开发者在维护GUI相关数据方面付出更多的努力。本文从分析MVC设计模式入手,力图从更高的层次上对GUI及其相关数据的描述模型进行归纳和抽象,提出并形式化描述了一种GUI抽象描述模型,基于该模型的GUI数据建模算法和绑定模型。最后简单介绍了该描述模型基于XML的实现语言——GUI XML。

关键词 图形化用户界面,抽象描述模型,MVC,GUI XML

Research of GUI Abstract Description Model

ZHANG Peng XU Peng

(Knowledge Engineering Group, Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract GUI, Graphical User Interface, is one of the most important technologies of various computer applications. However, it has been becoming more and more sophisticated to develop a cross-platform application because of the more and more programming languages and GUI toolkits. At same time, the larger volume and higher complexity of data drive developers to pay more attention to how to maintain the data related to GUI. In this paper, we begin with analyzing MVC pattern, and try to induce and abstract the description model of GUI and relative data from higher level. And a definition of GUI abstract description model is formally described as a result. Based on this description model, the GUI data modeling algorithm and binding model are illustrated. Finally, it briefly introduces an implementation of this model - GUI XML which is based on XML.

Keywords GUI, Abstract description model, MVC, GUI XML

1 引言

图形用户界面,简称GUI,作为计算机应用的重要组成部分,其设计和实现方法一直受到研究人员和开发者的重点关注。但是面对不断变化的需求,GUI设计和开发工作的难度却越来越大。其原因主要有以下两个方面:

首先,GUI开发工具包种类的增多导致多版本GUI应用的开发工作量增大,而重复性工作在其中占据了很大比例。各种高级编程语言都提供了丰富和强大的GUI开发工具包。例如Java语言就提供了包括AWT,Swing以及SWT在内的多种GUI工具包。不同的GUI开发工具包在语法定义和对像建模方式上均存在差异。繁多的工具包使开发者在设计GUI的多个工具包版本时,必须付出大量时间,针对不同的工具包编程接口重复实现相同的用户界面。例如,开发者需要实现M个不同的GUI,且每个GUI都需要在N种GUI工具包上实现,就必须编写$M*N$个不同的程序。

其次,应用所支持的数据源类型和数据量的不断增大提高了GUI开发工作的复杂程度。随着网络的发展和普及,数据日益趋向复杂化,数据量也越来越大。这必然使得用于显示和采集数据的GUI也日益复杂。同时,存储和提供数据的数据源类型也不断发生变化,从关系数据库到XML文档,甚至XML数据库。因此,GUI开发者需要花费更多的精力用于维护支撑GUI运行的数据,以及数据与GUI控件之间的关系。这也加大了开发复杂GUI应用的困难。

为了提高GUI的设计效率和复用性,MVC设计模式[1]被提出,并最终被广为接受。MVC模式中,GUI被分成模型(Model)、视图(View)、控制器(Controller)三个模块。MVC模式充分发挥了面向对像技术的优势,从软件工程的角度解决了GUI设计和开发的部分问题。但是以上两个问题仍然没有得到根本的解决。

为了从根本上解决多GUI工具包和复杂数据给GUI设计开发工作带来的困难,我们分析比较了多种工具包,对其中的共同点以及差异进行归纳和总结。并参考MVC设计模式,在此基础上进一步抽象得到本文所阐述的GUI抽象描述模型。该模型在与具体工具包无关的前提下,提供一套强大,且拥有良好扩展性的描述方法。借由此抽象描述模型,开发者能够从更高的层次上描述复杂GUI,并摆脱对具体GUI工具包的依赖。抽象描述的GUI通过基于具体工具包的解释工具方便地翻译成具体的GUI实现。并且解释工具可以做到完全地复用,从而大大减少GUI设计开发中的重复工作。因此,同样在N个工具包上开发M个GUI应用,利用该模型可以将问题的复杂度从$M*N$简化为$M+N$。

GUI抽象描述模型不仅仅描述GUI,还为支撑GUI运行的数据提供统一的抽象数据描述方法。该数据描述方法屏蔽了数据源和数据结构的差异,保证数据有效地与GUI部分集成和实现互操作,形成一个有机整体。基于该数据描述方法,本文还阐述了GUI数据建模和维护算法,以及数据与GUI间的绑定模型。它们使开发者不用再耗费巨大的精力来针对

张 鹏 硕士研究生,主要研究方向:XML应用及处理平台;Web Services等。

异构的数据进行维护和处理。

本文的第2节从MVC设计模式入手，分析GUI设计的要素，进而采用形式化描述的方式给出GUI抽象描述模型的定义，并简单分析其与MVC模式的关系。第3节和第4节分别阐述基于该抽象模型的数据建模算法和数据绑定模型。然后，一种基于GUI抽象描述模型和XML语法的GUI描述语言——GUI XML将在第5节中做简单介绍。

2 GUI抽象描述模型

2.1 MVC设计模式

MVC设计模式被广泛认为是设计复杂图形用户界面应用的最优秀设计模式之一[2]。它将GUI分离成模型、视图和控制器三个相对独立，但是又相互影响和制约的模块，大大增强了GUI设计的灵活性和复用性。仔细分析MVC模式的每个模块可以发现它们都由一类实体以及实体间的关系构成：

视图(View)：如果将控件抽象成控件实体，而布局关系抽象成这些实体间的关系，视图则可以表示为控件实体的集合以及定义在这些实体上的关系。

控制器(Controller)：无论控制器控制的是模型还是视图，这些操作本身都是由设计者预先定义的逻辑片断，也是一种实体。因此，控制器可以认为是操作实体的集合。

模型(Model)：在面向对象的设计环境中，很容易就能将模型描述为数据实体以及实体之间的关系的总合。

MVC的三个模块在定义上相对独立，但是它们之间同时也存在着各种关系，例如视图利用控件显示模型内的数据，并激发控制器的操作逻辑等等。

综上所述，MVC模式下的GUI可以分解成各种实体以及实体之间的各种关系。GUI设计就是定义出这些实体和关系的过程。

2.2 GUI抽象描述模型定义

GUI抽象描述模型是一个与GUI实现工具包和数据源无关的描述模型。它将一个GUI实例表示为如下的二元组：

$$I=(E,R) \tag{1}$$

其中，I 表示一个独立的GUI应用实例，E 为GUI实例中各种类型实体的集合，而 R 是实体间各种关系的集合。

2.2.1 实体集合E 实体集合 E 定义GUI中的各种实体，例如用于显示的控件，支撑GUI运行的数据和预定义的操作逻辑等。在面向对象的环境下，这些实体的共同特点可以用相对独立的对像表示。根据实体的类型和作用，E 定义为：

$$E=C\cup D\cup A \tag{2}$$

其中，C 是由GUI中控件实体组成的集合，D 是数据实体的集合，A 是GUI设计者定义的对控件和数据的操作实体集合。

控件集合 C

在不同的GUI实例中，控件集合C包含不同的控件，即使相同的控件也可能拥有不同的样式信息和可触发的GUI事件。因此，对于任意的控件 $c\in C$，又采用下面的三元组来定义，即：

$$c=(class,P,Event) \tag{3}$$

其中，class指明控件 c 的类型名称，如按钮等；P指定控件的样式属性集合，例如大小、颜色等；Event则定义控件所能触发的GUI事件的集合，例如鼠标点击等。GUI事件不仅拥有其唯一的标识，还包含该事件的简单描述信息，例如事件源、事件类型等等。

数据集合 D

数据集合 D 中包含支撑GUI运行所需的所有数据。这里所讲的数据，指的是即使脱离GUI也能独立存在，并仍保持其原有意义的数据。显然，GUI控件的大小、位置等等，都不能称为数据。数据由数据源提供，或由用户输入。其最小组成单位是数据项。因此，D 又被称作数据项集合。

操作集合 A

A 表示操作集合，包括由设计人员预先设计或GUI工具包预定义，并在运行时由GUI事件激发，GUI实例在无用户干预的条件下自动完成的事务逻辑。

2.2.2 关系集合R 根据前面的定义，关系集合 R 包括GUI中所有实体间的关系。R 中的关系均为二元关系。R 定义如下：

$$R=\{r\mid r=(e_i\rightarrow e_j),e_i,e_j\in E,e_i\neq e_j\} \tag{4}$$

其中 e_i,e_j 表示构成关系的两个实体。根据 e_i,e_j 的类型和特点，可以将 R 划分为多个不相交的子集。如下式：

$$R=R_C\cup R_D\cup R_{CD}\cup R_{AD}\cup R_{CA}\cup R_{AC} \tag{5}$$

下面分别解释这些子集的含义。

控件关系集合 R_C

任何由两个以上控件构成的GUI实例中，所有控件按照一定的布局关系组合在一起。根据GUI实例中控件布局的父子关系，R_C 可以定义为：

$$R_C=\{r\mid r=(c_i\rightarrow c_j),c_i,c_j\in C,c_i\text{ 是 }c_j\text{ 的直接父容器}\} \tag{6}$$

这里定义的关系是静态的，无法在运行时进行改变。考虑到数据对GUI的影响，并希望在运行时根据数据决定控件间的关系，则需要对 R_C 内元素的定义做一定的扩充。扩充后的 R_C 定义如下：

$$R_C=\{r\mid r=(c_i\xrightarrow{D'}c_j),c_i,c_j\in C,c_i\neq c_j,D'\subseteq D,\text{当 }D'\text{满足其条件时 }c_i\text{ 是 }c_j\text{ 的直接父容器}\} \tag{7}$$

对于与数据无关的静态控件关系，可以看作是定义(5)中 D' 为空集的情形。因此，R_C 中的元素描述了GUI实例中所有控件之间的静态和动态的直接父子关系。如果将控件表示为节点，R_C 的元素表示为连接相应节点的边，则可以得到表示GUI实例的树或者森林，称之为控件树或控件森林。

数据依赖集合 R_D

数据依赖集合 R_D 中的元素定义了数据集合内数据项之间的依存和制约关系，称为数据依赖关系。例如在一个订单中，订单总额是所有订单项的金额之和。订单总额与每个单项的金额之间就存在数据依赖关系。R_D 描述如下：

$$R_D=\{r\mid r=(d_i\rightarrow d_j),d_i,d_j\in D,d_i\neq d_j,d_i\text{ 依赖于 }d_j\} \tag{8}$$

数据绑定集合 R_{CD}

数据绑定关系建立在控件和数据之间，定义为：

$$R_{CD}=\{r\mid r=(c_i\rightarrow d_i),c_i\in C,d_i\in D\} \tag{9}$$

与其他关系集合不同的是，该集合有对称性，即：

$\forall c_i\in C,d_i\in D,r=(c_i\rightarrow d_i)\in R_{CD}\Rightarrow r'=(d_i\rightarrow c_i)\in R_{CD}$，反之亦然。也即，$c_i$ 和 d_i 之间的绑定是双向的：两者中任何一方由于外界因素发生变化，必然会影响到另一方。

数据操作关系集合 R_{AD}

R_{AD} 中定义的是操作和数据之间的关系。定义如下：

$R_{AD}=\{r\mid r=(a_i\rightarrow d_i),a_i\in A,d_i\in D,a_i$ 的执行将影响

$d_i\}$ (10)

事件激发关系集合 R_{CA}

R_{CA} 定义的是控件与操作之间的关系，定义如下：

$$R_{CA}=\{r|r=(c_i \xrightarrow{e_i} a_i),\ c_i=(class_i,P_i,Event_i)\in C,\ a_i\in A,e_i\in Event_i \text{ 激发 } a_i\} \quad (11)$$

控件操作关系集合 R_{AC}

与 R_{AD} 的定义类似，定义如下：

$$R_{AC}=\{r|r=(a_i\rightarrow c_i),a_i\in A,c_i\in C,\ a_i \text{ 的执行将影响 } c_i\} \quad (12)$$

由于操作集合 A 由开发者预先定义，且运行时无法修改，因此，R_{AD}，R_{CA} 和 R_{AD} 也在设计时定义，运行时不发生变化。

2.3 GUI 抽象描述模型与 MVC

由上一节对 GUI 抽象描述模型的定义可知，该模型对 GUI 的抽象虽然简单，但是涵盖了 GUI 设计的各个方面。

GUI 抽象描述模型从分析 MVC 模式入手，将 GUI 抽象成实体以及实体间的各种关系的组合。同时从 GUI 抽象描述模型的定义中仍然可以很容易地抽取出 MVC 模式下的三个模块：定义中的数据集合 D 以及数据关系 R_D 构成 MVC 中的模型(Model)；控件集合 C 和控件关系 R_C 则构成视图(View)；而操作集合 A 则表示控制器(Controller)。

此外，GUI 抽象描述模型将实体间的关系作为单独的因素加以考虑，并统一描述。不仅定义控件之间和数据之间的关系，而且对于不同类型实体之间的关系也有描述。由图 1 可以看出，这些关系的定义使 MVC 模式中各模块之间的交互变得更清晰。

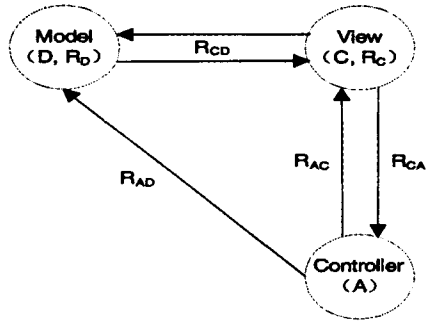

图 1 抽象描述模型与 MVC 模式

3 基于抽象描述模型的数据建模

GUI 抽象描述模型里，为了屏蔽异构数据源的影响，MVC 模式中的模型(Model)被抽象地描述为数据项集合 D 和数据依赖关系集合 R_D。但是这种集合式的结构需要花费大量计算时间进行遍历和维护，并不适合直接在实际应用中使用。为了提高数据的处理和维护效率，需要在抽象描述的基础上进一步构造数据的拓扑结构和计算模型，称为数据建模。本节给出基于抽象描述模型的数据建模和更新算法。

3.1 数据依赖图

将 D 作为顶点集合，R_D 作为边集合，抽象描述的数据构成一个有向图，称该图为数据依赖图。可以根据 D 和 R_D 方便地构造数据依赖图，算法描述如下：

```
初始化空图 G;
for(D 的每个元素 d){
  在图 G 中增加一个数据项顶点 Vd;
  Vd 的入度置 0;
}
for(RD 的每个元素 r=(di→dj)){
  从 Vdi 到 Vdj 作有向边;
  Vdj 的入度加 1;
}
```

清单 1 数据依赖图构造算法

由以上构图算法可知，数据依赖图是一个有向图。图中有可能出现多顶点环。该图描述数据项之间的拓扑排序。在对数据进行更新时，必须依照拓扑排序来对数据项依次更新。

3.2 数据更新子图

根据数据依赖图能够方便地进行数据更新操作。但是，对于大量的数据项，数据依赖图会显得庞大而复杂。直接根据该图来做更新计算，必然会消耗大量的系统资源。在绝大多数情况下，需要更新的只是数据依赖图的一个子图。并且，考虑到数据备份和发生错误后的数据恢复等因素，更新计算可以在数据依赖图的一个复制子图上进行，该复制子图称为更新子图。

不妨设由外部操作修改的数据项所对应的顶点列表为 L_c，则更新子图的构造算法如下：

```
for(列表 Lc 中的每个顶点 x)
  if(x 的副本不在子图里){
    将顶点对(NIL, x)入栈;
    while(栈不为空){
      从栈中弹出顶点对(v, w);
      if(w 的副本不在子图里){
        将 w 的副本 wS 加入子图;
        在 w 和 wS 中加入彼此的引用;
        for(w 每条入边的起始顶点 r)
          将(w, r)入栈;
      } else 从 w 中获得 wS;
      if(v 不是 NIL){
        从 v 中获取 vS;
        在子图中添加 wS 到 vS 的有向边;
      }
    } // end while
} // end if
```

清单 2 更新子图构造算法

此算法采用深度优先的方法搜索相关的顶点和边。并且在原图的顶点和更新子图的顶点之间建立双向引用，以便于计算结束后更新原图。对于原图中的环，子图中舍弃了环内的一条边，以保证计算不会陷入无穷循环。

3.3 数据更新算法

在数据更新子图上计算数据的算法改用宽度优先的方法，以保证计算严格按照顶点的拓扑排序进行。算法描述如下：

```
for(Lc 中的每个顶点 x){
  标记 x 为已计算;
  if(x 的入度不为 0)
    将到 x 的所有入边的起点入栈;
}
while(栈不为空){
  从栈中弹出顶点 v;
  if(v 的入度不为 0)
    for(v 的每条入边的起点 w)
      w 入栈;
  if(v 每条出边的终点都已标记为已计算)
    if(v 未计算)
      计算 v 的数据，并标记 v 为已计算;
  else {
    v 入栈;
    v 的出边终点中未计算者入栈;
  }
}
```

清单 3 数据更新计算算法

对于 D 或者 R_D 中元素发生增加或删除的情况，则需要在数据依赖图上先增加或删除相应的顶点或边，然后使用更

新子图的算法更新数据。

按照以上三个算法，很容易建立基于 GUI 抽象描述模型的数据模型，从而大大简化开发者在维护数据方面所做的工作。同时，数据更新算法为后面介绍的数据绑定模型提供了良好的基础。

4 数据绑定模型

R_C 定义的关系将控件按照直接父子关系构成一棵树或一片森林。其中的节点通过 R_{CD} 定义的关系与数据项绑定。我们将这种绑定分为两种方式：显示绑定和逻辑绑定。

显示绑定使得数据项和控件的样式属性之间同步变化：数据项取值的变化，会在控件上通过样式属性的改变直观体现；而对控件样式属性的修改也会直接影响数据项的取值。这正是 R_{CD} 对称性的体现。显示绑定关系集合 R^v_{CD} 用形式化定义表示为：

$$R^v_{CD}=\{r\mid r=(c_i \xrightarrow{p} d_i), c_i=(class_i, P_i, Event_i)\in C, p\in P_i, d_i\in D\} \quad (13)$$

R^v_{CD} 是 R_{CD} 的子集。

逻辑绑定是数据项与控件在逻辑上的对应关系，不可视。但是该绑定关系决定了该控件所有后代控件的数据绑定上下文。即当逻辑绑定关系发生变化时，该控件的所有后代控件对应的数据绑定关系都将相应地改变。逻辑绑定关系集合记为 R^l_{CD}，则：

$$R_{CD}=R^v_{CD}\cup R^l_{CD} \quad (14)$$

这两种绑定方式就构成了数据绑定模型。针对面向对像环境下的 GUI 设计，该绑定模型解决了数据与控件间的同步问题。在下面的阐述中，以 DOM 数据的绑定为例说明模型是如何工作的。

图 2 所示的是某 GUI 实例初始状态下的数据依赖图 Data 和控件树 Controls，以及实体间的关系。数据依赖图中的每个节点都表示了 DOM 中的一个 Element 或 Attribute。直线箭头表示数据依赖关系或者控件间关系；弧形箭头表示绑定关系，其中虚线表示逻辑绑定，实线表示显示绑定。

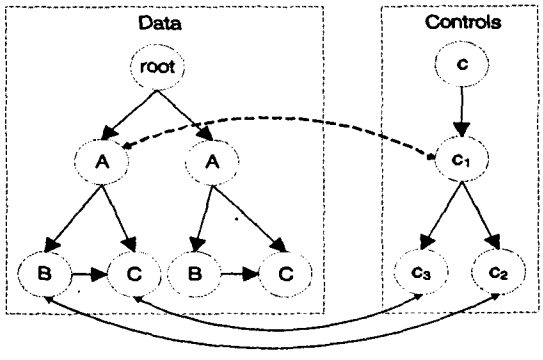

图 2 数据依赖图和控件树实例

以 XPath[3] 的路径表达式代替数据项，可以写出该状态下的三个绑定关系：

$r_1=(c_1\rightarrow/root/A[1])^*$，$r_2=(c_2 \xrightarrow{p_2} B)$，$r_3=(c_3 \xrightarrow{p_3} C)$，其中，$p_2$，$p_3$ 分别是 c_2，c_3 的样式属性。由于 r_1 是逻辑绑定，为 r_2 和 r_3 提供了数据上下文，因此 r_2 和 r_3 中的路径表达式使用相对形式，而不会造成歧义。在下列两种情形中，数据绑定模型将起到数据同步的作用：

情形一：某时刻 c_3 的样式属性 p_3 被用户修改。同时，与之绑定的 DOM 数据节点值也被修改。当进行数据更新时，数据依赖图的更新算法将计算左侧的 A 和 B，以及 $root$ 节点值。由于 c_2 与数据间的显示绑定关系，数据绑定模型将强制控件 c_2 更新其样式属性，以显示 B 节点数据的变化。

情形二：运行时，c_1 上的逻辑绑定关系变化为 $r_1=(c_1\rightarrow/root/A[2])$。当应用需要根据数据更新其可视界面时，$c_2$ 和 c_3 从 c_1 查找绑定上下文，即绑定路径表达式，并重新获取绑定的数据。此时，虽然 c_2 和 c_3 绑定的绑定路径表达式没有变化，但是绑定关系已经变化，如图 3 所示。

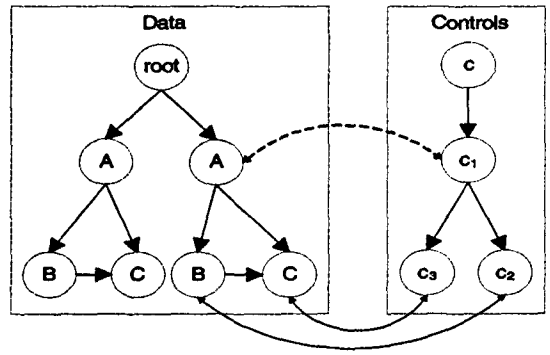

图 3 绑定关系的变化

由以上实例可以看出，数据绑定模型为数据与控件间的映射提供更灵活更强大的定义方式，同时也能有效地解决它们之间的双向同步问题。

5 基于 XML 的抽象描述模型实现

综上所述，GUI 抽象描述模型以及建立其上的数据建模算法和绑定模型为简化 GUI 设计开发工作提供了坚实的理论依据。在此基础上，我们设计了 GUI XML——完全基于 XML 语法的实现语言。图 4 是 GUI XML 的设计框架[6]。

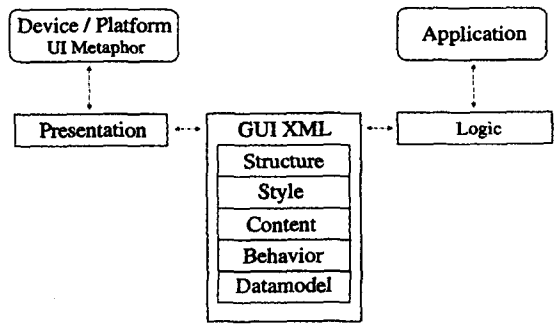

图 4 GUI XML 的设计框架

在 GUIXML 中，对 GUI 实例的 XML 描述分为五个主要组成部分：

结构(Structure)— 描述实例控件集合以及控件与数据的绑定关系，并利用 XML 天然的层次结构定义控件关系。结构部分定义抽象描述模型中的控件集合 C，控件关系 R_C 和数据绑定关系 R_{CD}，并完全实现第 4 节介绍的数据绑定模型。

样式(Style)和内容(Content)—为了更好地重用字符串、外部文件和控件样式属性等各种资源，样式和内容部分将这

(下转第 217 页)

*假设 DOM 树中同层节点的序号自左往右递增。

gramming Languages (POPL'98). New York: ACM Press, 1989. 179～190
14 Thistle J G, Wonham W M. Supervision of infinite behavior of discrete-event system. SIAM Journal on Control and Optimization, 1994, 31(4): 1098～1113
15 Mazala R. Infinite Games. In: Grädel E, Wolfgang T, Wilke T, eds. Automata, Logics, and Infinite Games-A Guide to Current Research. Berlin: Springer-Verlag, 2000. 23～38
16 Büchi J R. On a decision method in restricted second order arithmetic. In: Proc. of the Intl. Congress on Logic, Methodology and Philosophy of Science. Stanford: Stanford University Press, 1962. 1～11
17 Klauck H. Algorithms for Parity Games. In: Grädel E, Wolfgang T, Wilke T, eds. Automata, Logics, and Infinite Games-A Guide to Current Research. Berlin: Springer-Verlag, 2000. 107～129
18 Alur R, Henzinger T A, Kupferman O, et al. Alternating Refinement Relations. In: Proc. of the 9th Intl. Conf. On Concurrency Theory (CONCUR'98). Berlin: Springer-Verlag, 1998. 163～178
19 Berardi D, Calvanese D, Giacomo G D, et al. Automatic Composition of e-Services that Export their Behavior. In: Proc. of the 1st Intl. Conf. on Service Oriented Computing (IC-SOC03). Berlin: Springer-Verlag, 2003. 43～58
20 Bultan T, Fu X, Hull R, et al. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In: Proc. of the 12th Intl. World Wide Web Conf. (WWW2003). New York: ACM Press, 2003. 403～410
21 Fu X, Bultan T, Su J. Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services. In: Proc. of the 8th Intl. Conf. on Implementation and Application of Automata (CIAA 2003). Berlin: Springer-Verlag, 2003. 188～200
22 Moisan S, Ressouche A, Rigault J-P. Towards Formalizing Behavioral Substitutability in Component Frameworks. In: Proc. of the 2nd Intl. Conf. on Software Engineering and Formal Methods (SEFM'04). Los Alamitos: IEEE Computer Society, 2004. 132～141
23 Rajamani S K, Rehof J. A behavioral module system for the pi-calculus. In: Proc. of Static Analysis Symposium (SAS'01). Berlin: Springer-Verlag, 2001. 375～394
24 Lumpe M, Achermann F, Nierstrasz O. A formal Language for Composition. In: Leavens G T, Sitaraman M, eds. Foundations of Component-based System. Cambridge: Cambridge University Press, 2000. 69～90
25 Salaün G, Bordeaux L, Schaerf M. Describing and Reasoning on Web Services using Process Algebra. In: Proc. of the IEEE Intl. Conf. on Web Services (ICWS'04). Los Alamitos: IEEE Computer Society, 2004. 43～51
26 Lano K, Bicarregui J, Maibaum T. Composition of Reactive System Components. In: Leavens G T, Sitaraman M, eds. Foundations of Component-based System. Cambridge: Cambridge University Press, 2000. 267～283
27 Han J. Interaction Compatibility: An Essential Ingredient for Service Composition. In: Proc. of the 2nd Intl. Workshop on Grid and Cooperative Computing (GCC2003). Berlin: Springer-Verlag, 2003. 59～66
28 Hamadi R, Benatallah B. A Petri Net-based Model for Web Service Composition. In: Proc. of the 14th Australasian Database Conf. (ADC2003). New South Wales: Australian Computer Society Inc, 2003. 191～200
29 Tao X, Jiang C. Formalizing Web Service and Modeling Web Service-Based System Based on Object Oriented Petri Net. In: Proc. of the 2nd Intl. Workshop on Grid and Cooperative Computing (GCC2003). Berlin: Springer-Verlag, 2003. 1008～1011
30 Lynch N, Tuttle M. An introduction to input/output automata. CWI Quarterly, 1989, 2(3): 219～246
31 Wen Y, Wang J, Qi Z. Bridging Refinement of Interface Automata to Forward Simulation of I/O Automata. In: Proc. of the 6th Intl. Conf. Formal Engineering Method (ICFEM2004). Berlin: Springer-Verlag, 2004. 259～273
32 Lee E A, Xiong Y. Behavioral Type for Component-Based Design. University of California, Berkeley, CA 94720, USA, Technical Memorandum UCB/ERL: M02/29, 2002
33 Chakrabarti A, de Alfaro L, Henzinger T A, et al. Interface Compatibility Checking for Software Modules. In: Proc. of the 14th Intl. Conf. on Computer-Aided Verification (CAV2002). Berlin: Springer-Verlag, 2002. 428～441
34 Bennet K, Layzell P, Budgen D, et al. Service-Based Software: The Future for Flexible Software. In: Proc. of the 7th Asia-Pacific Software Engineering Conf. (APSEC'00). Los Alamitos: IEEE Computer Society, 2000. 214～221
35 Allen R, Garlan D. A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, 1997, 6(3): 69～90
36 Min H G, Choi S W, Kim S D. Using Smart Connectors to Resolve Partial Matching Problems in COTS Component Acquisition. In: Proc. of the 7th Intl. Symposium on Component-Based Software Engineering. Berlin: Springer-Verlag, 2004. 40～47
37 杨芙清，梅宏，吕建，等. 浅论软件技术发展. 电子学报，2002, 30(12A): 1901～1906
38 Henzinger T A. Automata for Specifying Component Interfaces. In: Proc. of the 8th Intl. Conference on Implementation and Application of Automata (CIAA2003). Berlin: Springer-Verlag, 2003. 1～2
39 de Alfaro L, Henzinger T A. Interface theories for component-based design. In: Proc. of the First Intl. Workshop on Embedded Software (EMSOFT). Berlin: Springer-Verlag, 2001. 148～165
40 de Alfaro L, Stoelingal M. Interfaces: A Game-Theoretic Framework for Reasoning About Component-Based Systems. In: Proc. of the 2nd Intl. Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA 2003). New York: Elsevier, 2004. 3～23
41 de Alfaro L. Game Models for Open Systems. In: Proc. of Intl. Symposium on Verification (Theoryin Practice). Berlin: Springer-Verlag, 2003. 269～289
42 Henzinger T A. Rich Interfaces for Software Modules. In: Proc. of the 18th European Conf. on Object-Oriented Programming (ECOOP 2004). Berlin: Springer-Verlag, 2004. 516～517
43 Griss M L. Software Agents as Next Generation Software Components. In: Heineman G T, Councill W T, eds. Component-Based Software Engineering: Putting the Pieces Together. Boston: Addison-Wesley, 2001. 641～657

(上接第 201 页)

些公共的资源抽取出来，单独定义，以便全局引用。

行为(Behavior)— 行为部分定义控件的事件集合 Event，操作集合 A，以及联结这两者的 R_{CA}。

数据(Datamodel)— 即数据模型定义。它将 XML 格式的数据，根据数据依赖定义，使用第三节给出的数据依赖图相关算法构造其拓扑结构。

综合以上五个部分，GUI XML 可以很自然地定义各种 GUI 实例，并通过解释引擎实例化得到运行时的实例。

其他基于 XML 的 GUI 描述语言还有 XUL[4]，Bamboo-kit[5]等。与 GUI XML 相比，它们只针对某种 GUI 工具包进行描述，而且缺乏对数据描述模型的定义。GUI XML 由于采用了抽像描述模型以及相关理论，在扩展性、数据集成能力等方面都具有明显的优势。

结论 针对目前 GUI 设计和开发过程中由多 GUI 工具包和复杂数据所造成的困难，本文提出 GUI 抽象描述模型以及建立在该模型之上的数据和绑定模型。该模型采用实体和关系集合的方式定义 GUI，满足了描述的通用性。数据模型和绑定模型将异构的数据组织成有向图结构，并在该图与控件树之间建立多种绑定关系，从而解决了数据维护以及数据与控件间的同步问题。GUI XML 从实际应用的角度证明了以上模型的良好描述能力和通用性。

参 考 文 献

1 Buschmann F, Meunier R, Rohnert H, Sommerland P, Stal M. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley and Sons, 1996
2 Hussey A, Carrington D. Comparing the MVC and PAC architectures: a formal perspective, Software Engineering. IEE Proceedings, 1997, 144(4)
3 Clark J, DeRose S. XML Path Language (XPath) Version 1.0 http://www.w3.org/TR/xpath/ Nov. 1999
4 Deakin Neil. XUL Tutorial 15 April 2004
5 Bambookit Tutorial. http://www.bambookit.com
6 清华大学—IT Frontier 株式会社知识工程联合实验室. GUI XML for JFC Swing 规范. Aug. 2004
7 张鹏，徐鹏. Java 图形用户界面的 XML 描述方法. 见：第七界全国 Java 大会，北京：2004