事件驱动算法演示技术的面向方面实现*)

周晓聪　郑汉雄　罗　达

(中山大学信息科学与技术学院计算机系　广州 510275)

摘　要　算法演示系统是利用图形和动画来表现算法的运行过程的系统。一直以来人们采用事件驱动技术进行算法演示系统的开发,但传统的实现方案容易导致代码混乱和分散,以及可重用性和可维护性低等缺点。提出使用面向方面技术开发算法演示系统的方案,与传统实现方案相比,基于面向方面技术的实现方案具有系统模块松散耦合、可重用性高等优点。

关键词　面向方面,AOP,算法演示,事件驱动

AOP-based Approach to Implement Algorithm Animation

ZHOU Xiao-cong　ZHENG Han-xiong　LUO Da

(Department of Computer Science, School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510275, China)

Abstract　An algorithm animation system visualizes the behavior of an algorithm by means of graphical displays. The traditional implementation of event-driven technology easily lead to code tangling and scattering, and lower reusability and maintainability. This article proposed a method based on aspect-oriented programming (AOP) to implement algorithm animation system. Comparing to the traditional implementation, AOP-based method provides a more loose-couple and reusability approach to modularize the animation system.

Keywords　Aspect-oriented, AOP, Algorithm animation, Event-driven

1　引言

算法演示[1](algorithm animation)属于软件可视化(software visualization)研究的一个分支,它利用图形和动画表现算法的运行过程,通常涉及到算法中的操作和所用到的数据。通过图形、动画等形象的描述,可以帮助算法学习者更好地理解算法的内在工作方式,以及了解算法的优缺点和对算法进行更进一步的优化。

目前的算法演示系统常用的一种方法是事件驱动方法(Event-driven Approach)。事件驱动方法固然有其好处,但随着算法复杂性的提高以及算法多样性的增加,事件驱动方法的缺陷逐渐暴露出来,如代码混乱、代码可重用性低等。本文正是针对这种情况,提出使用面向方面程序设计技术(Aspect-Oriented Programming)实现事件驱动的算法演示,以解决这些问题。

2　事件驱动的算法演示技术

2.1　算法演示系统的分层模型

一个算法演示系统需要解决3个问题:(1)演示什么;(2)如何演示;(3)这两者之间的无缝连接。文献[2]提出了算法演示系统的一个分层模型,其核心是将算法演示系统分为算法层、连接层和表现层,如图1所示。

在这个模型中,算法层存放各种算法的实现代码及其入口,为具体代码的运行提供支持。进一步,算法层收集当前正在演示的算法的状态,并将算法状态传输给连接层。也就是说,算法层主要解决演示什么的问题。

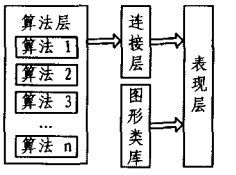

图1　算法演示系统层次模型

连接层接收算法状态,将算法状态整理成可视化所需的数据结构,并为表现层的可视化提供统一的数据接口。也就是说,连接层提供算法层与表示层之间的连接。

最后,表现层解决如何演示的问题,其主要功能是将连接层提供的数据用图形(或动画)的方式展示给用户。表示层需要图形类库的支持。

算法演示系统的这种分层模型通过明确划分各个层次的功能及层次间的接口,可以降低系统开发的复杂度,提高系统的可重用性和可扩展性。

文献[2]没有进一步探讨这种模型的具体实现。我们认为,面向方面的程序设计技术实际上提供了一种合适的实现途径,因为利用方面的编织(weaving)可以实现表现层与算法层的无缝连接,很好地体现这种模型的优势。

2.2　算法演示中的事件驱动

在算法演示系统中,连接层的实现存在两种方法:事件驱动方法和状态驱动方法[3]。我们采用事件驱动方法,因为它比状态驱动方法更容易展示算法的行为。

目前大多数算法演示系统也都采用事件驱动的方法。这种方式首先需要定义程序中那些对应于底层算法基本操作的

*)基金项目:国家自然科学基金项目(No. 60673050)。**周晓聪**　博士,副教授,主要研究方向为软件工程中的新技术、新方法、软件开发形式化方法等;**郑汉雄**　硕士研究生;**罗　达**　硕士研究生。

关键点，当程序执行到这些关键点时，相应的参数化事件就会被传递到系统中的可视化部件，从而显示出相应的画面。

对于这种事件驱动的说明方式，我们需要做的工作有[3]：(1)定义算法中与可视化相关的动作。通常将这种相关动作称为感兴趣事件(interesting events)。比如排序算法中的交换操作就可以作为感兴趣事件。(2)将每个感兴趣事件和适当的演示场景联系起来。例如，对于一个排序算法，用一些长柱形表示元素，长柱的高低代表元素值的大小，算法中两个元素的交换就可以和演示画面中两个长柱形的交换联系起来(如图 2)。

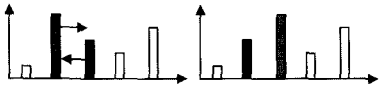

图 2　简单的排序算法可视化

2.3　事件驱动算法演示的实现

下面以一个冒泡排序算法的可视化为例说明其实现思想和步骤。基于事件驱动的思想，该函数实现框架如下：

```
// Animation. java
……
public void run()
{
    // 假设待排序数组为 num
    int[] num = {23,98,12,43,34};
    int n = 5;// 数组长度
    int i,j;
  // 冒泡排序循环
    for (i = n-1;i > 0;i--)  { // 外层循环
        for (j = 0;j < i;j++) { // 内层循环
            // 显示出待比较的 2 个数字
            // 比较如果需要交换，就进入
            if (num[j+1] > num[j])  {  //比较
                // 开始交换的过程
                // 首先确定 2 个数字的坐标
                // 可视化第 1 步：分离
                // 可视化第 2 步：上下交换
                // 可视化第 3 步：靠拢
                /* 实现代码(略) */
                // 冒泡排序算法中对数字进行交换
                /* 实现代码(略) */
                // 再刷新画面
                /* 实现代码(略) */
            }
        }
    }
}
```

2.4　小结

事件驱动的算法演示技术便于开发者在开发过程中快速定位感兴趣事件，而且可以灵活地实现感兴趣事件的可视化。但是，根据上面的说明，我们不难发现事件驱动方法的一些缺陷：

(1)代码混乱，可维护性低

在上面的程序中，我们可以看到算法层的代码(粗体字部分)被“插入”了许多可视化代码。这使得演示程序的结构比较混乱，开发者难以清晰地构造算法本身的流程。一旦开发者要对算法进行调整，就不得不修改相应的可视化代码，降低了算法的可维护性。

(2)代码分散，可扩展性低

不难发现在存在多种算法的情况下，每个算法的可视化都需要调用相应的绘图函数，这些函数分散在各个算法中，加大了维护的难度。此外，如果此时系统中需要加入一些额外功能，如记录算法执行情况日志或者异常检测等，对程序的扩展将十分困难。

鉴于事件驱动方法存在的这些问题，我们提出用面向方面技术实现算法演示系统的分层模型，从而使得系统具有更好的可维护性和可扩展性。

3　面向方面技术

传统的面向对象程序设计技术(Object-Oriented Programming，简称 OOP)在具有“垂直”结构的业务逻辑方面体现了强大的威力，但面对横切关注点这类“水平”结构则往往束手无策。图 3 给出了垂直结构的业务逻辑与横切关注点之间关系的示意图。虽然人们提出了诸如多重继承、拦截器模式等解决方案，但仍无法从根本上解决这一问题。相反，过多地依赖这类解决方案往往会导致代码混乱和代码分散的缺点。

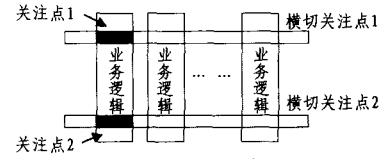

图 3　关注点及横切关注点

面向方面程序设计技术(Aspect-Oriented Programming，简称 AOP)正是为解决横切关注点实现的问题而产生的一种新的程序设计范型[4]。通过方面(Aspect)的使用，开发者可以很容易地实现关注点分离，最终建立一个具有正交特点的系统。目前 AOP 技术主要应用于日志、验证、线程安全、缓存管理等具有明显横切关注点的领域。

3.1　AOP 的相关概念

面向方面的程序设计引入了一些新的概念，用以阐述软件开发过程中的面向方面内容及其表现[5,6]。

(1)关注点(concern)

关注点是指一个实际问题、概念或者一个应用程序感兴趣的区域。

(2)横切关注点(cross-cutting concern)

横切关注点就是指那些跨越了多个类的关注点。

(3)方面(Aspect)

方面类似于 Java 中的类，可以在方面中定义属性和方法，也可以实现从其他类或方面中扩展以及实现接口。AOP 在方面中实现了横切关注点，通过方面中定义的规则将横切模块和业务模块编织(weave)在一起，从而实现多维度上的横切关注和业务关注的分离。

(4)连接点(Join Point)

连接点是程序中需要强调的事件，是可以被用于当作切入点(Pointcut)的执行点。连接点是 AOP 正交关系中主程序和方面的交点，它指明了可以织入横切行为的位置。

(5)切入点(Pointcut)

切入点就像过滤器。随着程序的运行，将产生一串连接点，而切入点的工作就在于挑选出其中它所感兴趣的连接点。

同时，切入点还可以为在连接点上执行的通知提供上下文。

(6)通知(Advice)

通知描述了切入点处要执行的具体操作。通知有 before、around、after 三种类型，分别对应切入点响应前、响应时、响应后的不同情况。通知实际上是一个拦截器，如果程序运行到切入点处，系统会触发拦截器并执行相应的操作。

3.2 AOP 的实现

显然在软件开发的分析和设计阶段对横切关注点、方面以及通知等进行建模十分重要，但目前对这个问题的研究还很不成熟[7]。不过在实现阶段，我们认为可以将面向方面的程序设计简单地分为以下步骤：

(1)方面分解

方面分解，也就是标识出软件系统中的横切关注点。一个系统的关注点通常包括核心关注点和横切关注点两种。其中，核心关注点主要是关注系统的业务逻辑；横切关注点主要关注系统级的服务，供业务逻辑使用。在 AOP 开发中，我们需要将这两者分解出来。

(2)关注点实现

各个横切关注点分开实现，彼此独立。对于核心关注点使用面向对象(OO)技术可以很好地解决；对于横切关注点则可以使用 AOP 技术实现。

(3)方面重组

最后，各个横切关注点被重组，或者被织入最终的软件系统中。我们可通过方面指定重组的方法与规则，并使用编织机制来构建最后的程序。

4 基于面向方面的算法演示

我们使用 AspectJ[8] 进行面向方面开发。AspectJ 是 Java 语言的扩展，是目前 AOP 最成熟的 Java 实现。

4.1 设计

针对事件驱动实现算法演示存在的不足，我们引入 AOP 实现关注点分离。一个好的系统应尽量降低各个模块间的耦合度。我们将连接层用方面实现，通过 AOP 的拦截机制对算法层中的关注点实现拦截。在 AOP 拦截获得所需的算法状态后，表现层就可以对这些状态实现可视化。

4.2 实现

根据上述的算法演示系统分层模型，应该将算法本身与可视化代码分离。以冒泡排序为例，考虑到排序算法的关注点主要在交换(swap)和比较(compare)两元素的值的操作上，因此我们分别将这两个操作作为关注点抽取成两个函数 swap()和 compare()。用一个类 Operation 来存放所有需要关注的操作。

然后，在方面中使用切入点拦截 Operation 类中的操作，在对应的通知中实现可视化。这样，当切入点定义的连接点被拦截时，系统就会自动执行通知中的可视化代码。

我们抽取一个名为 Sorting 的类，该类实现各种排序算法。对于用户感兴趣的事件(即需要可视化的行为)，只需在相应位置调用 Operation 类中的对应操作即可。而这种调用几乎不影响算法本身的独立性，如下例所示：

```
// Animation.java
.....
    public void run() {
Sorting.DoBubbleSort();  // 演示冒泡排序
}
```

```
// Sorting.java
public class Sorting {
    // num 为待排序数组
    public static int num[] = { 23,76,32,98,21 };
    public static int n = 5;  // 数组长度
    public static void DoBubbleSort() {
        for (int i = n-1;i > 0;i--)
            for (int j = 0;j < i;j++)
                if (Operation.Compare(j+1,j)) {
                    Operation.Swap(j+1,j);
                }
    }
}
// Operation.java
public class Operation {
    // 在该类中实现用户感兴趣事件
    // 交换两元素值，入参为 num 数组的下标
    public static void Swap(int i,int j) {
        int k = Sorting.num[i];
        Sorting.num[i] = Sorting.num[j];
        Sorting.num[j] = k;
    }
    // 比较两元素值大小，入参为 num 数组的下标
    public static boolean Compare(int i,int j) {
        return Sorting.num[i] > Sorting.num[j];
    }
}
// MyAspect.aj 在方面中实现横切关注点
public aspect MyAspect {
    // 对 Swap 函数设置切入点
    pointcut testSwap(int i,int j):
        execution(* Operation.Swap*(..))&&args(i,j);
    // 对 Compare 函数设置切入点
    pointcut testCompare(int i,int j):
        execution(* Operation.Compare*(..))&&args(i,j);
    // 拦截 Swap 函数的执行并实现可视化
    before(int i,int j):testSwap(i,j) {
        /* 可视化代码 */
    }
    // 拦截 Compare 函数的执行并实现可视化
    before(int i,int j):testCompare(i,j) {
      /* 可视化代码 */
    }
}
```

4.3 小结

通过引入 AOP 技术，算法演示系统的实现存在以下优点：

(1)算法代码可以独立运行，模块间耦合度低

AOP 使我们完整地实现了算法演示系统的层次结构。算法层与连接层之间的接口是隐式连接的，降低了模块间的耦合性，而提高了算法的可读性。算法层的实现与可视化代码分离开来，算法代码本身可以独立运行，便于开发者调试和修改算法。

图 4 给出了传统基于事件驱动的算法演示系统的结构图。不难看出，这时由于每个算法都需要和表现层相联系，导致整体结构比较混乱。而图 5 显示的面向方面的算法演示系

(下转第 257 页)

适用于面向对象软件的测试。最近研究表明，面向对象软件系统内部结构具有小世界(SW)效应和无标度(SF)特性，其中代码重用技术引起了“富者愈富”现象的产生。根据面向对象软件系统内部拓扑结构这一特点，本文从软件系统复杂网络的特性出发，对面向对象的集成测试进行了研究。提出通过分析类之间的交互复杂性和聚集复杂性来确定软件测试顺序的两种不同的测试策略，其中重点分析了基于交互复杂性的第一种测试策略的使用，而对于基于聚集复杂性的第二种测试策略将是我们下一步重点研究的内容。目前，基于复杂网络研究软件测试的工作还很少，以往的对面向对象软件测试的研究大多忽略了面向对象软件复杂网络的特性。本文的创新之处是将两者结合起来，在目前新的复杂网络研究成果的基础上，对面向对象软件系统的集成测试进行研究，目标是在提高软件可靠性的同时减少测试的工作量，提高测试工作效率。

参 考 文 献

[1] Watts J S, Strogatz S H. Collective dynamic of small-world networks. Nature, 1998, 393(6684): 440-442

[2] Batabasi A L, Albert R. Emergence of scaling in random networks [J]. Science, 1999, 286: 509-512

[3] Sole' R V, Ferrer R, Montoya J M, et al. Tinkering and mergence in complex networks [J]. Complexity, 2002, 8 (1): 20-33

[4] Valverde S, SoléR. Hierarchical Small-worlds in Software Architecture [R]. Santa Fe Institute working paper, SFI/ 03-07-044. 2003

[5] Jorgensen P C, Erickson C. Object-oriented Integration Testing. Communications of the ACM, 1994, 37(9): 30-38

[6] Mc Gregor J. Let's Don't and Say We Did. Journal of Object-oriented. Programming, 1998, 11(5): 6-11, 14

[7] Kung D, Gao J, Hsia P, et al. Class Firewall, test order and regression testing of object-oriented programs. Journal of Object-Oriented Programming, 1995, 8(2): 51-65

[8] 刘玲. 基于面向对象形式规格说明的测试用例生成技术. 博士学位论文. 上海大学, 2004

[9] 金凌紫. 面向对象软件测试技术进展[J]. 计算机研究与发展, 1998, 35(1): 6-13

[10] Pressman R S. 软件工程—实践者的研究方法. 第5版. 梅宏, 译. 机械工业出版社, 2002

[11] 李兵, 王浩, 李增扬, 等. 基于复杂网络的软件复杂性度量研究. 电子学报, 2006: 2371-2375

[12] Potanin A, Noble J, Frean M, et al. Scale-free geometry in object-oriented programs [J]. Communications of the ACM, 2005, 48(5): 99-103

[13] 闫栋, 祁国宁. 大规模软件系统的无标度特性与演化模型. 物理学报, 2006, 55(8): 2800-2804

(上接第 249 页)

统则很好地解决了这一问题。

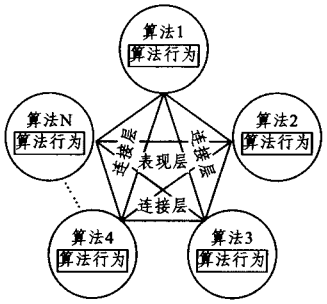

图 4 事件驱动的算法演示系统

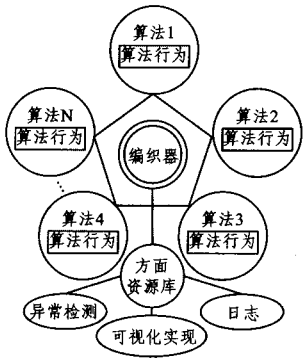

图 5 面向方面的算法演示系统

(2)可扩展性强，便于定义感兴趣事件

如果需要新增算法，只需将直接算法代码添加到系统中即可。对于感兴趣事件，即待可视化的事件，可通过在关注点处使用预先约定好的函数(Operation 类成员函数)，而这些函数的使用完全不影响算法本身的结构和可独立运行的能力，这种替换是非侵入性的。如果要修改感兴趣事件，也只需对 Operation 类进行改变即可。

(3)提高代码可重用性

通过 AOP 对关注点的拦截，系统可以有效地实现代码重用。在排序类算法的演示中通常需要对数组元素的交换操作作可视化，通过在方面中写入可视化代码，就可以轻易实现所有对应关注点上的可视化。

结束语 本文介绍了一种使用面向方面技术来实现算法演示系统的方法。使用面向方面技术，克服了传统事件驱动的算法演示系统存在的代码混乱、可维护性和可扩展性低的问题。相比较而言，本文介绍的 AOP 方案更为清晰灵活，各模块间耦合度大大降低，提高了算法演示系统的可维护性、可扩展性以及可重用性。

我们下一步将针对面向方面技术的特点，将上述算法演示系统的分层模型进一步完善，构建一个面向方面的算法演示系统通用框架，并开发一些中间件，支持各种算法的演示。

参 考 文 献

[1] Kerren A, Stasko J T. Algorithm Animation - Introduction[C] //Diehl S, ed. Software Visualization, LNCS. Springer-Verlag, 2002, 2269: 1-15

[2] 黄永红, 黄卫红, 符开耀, 等. 开放式算法动态演示平台模型[J]. 长沙电力学院学报, 2004, 19(1): 23-25

[3] Demetrescu C, Finocchi I, Stasko J T. Specifying Algorithm Visualizations: Interesting Events or State Mapping[C]// Diehl S, ed. Software Visualization, LNCS. Springer-Verlag, 2001, 2269: 16-30

[4] Kiczales G, et al. Aspect-Oriented Programming[J]// Aksit M, Matsuoka S, eds. ECOOP'97, LNCS. 1997, 1241: 220-242

[5] Gradecki J D, Lesiecki N. 精通 AspectJ[M]. 王欣轩, 吴东升, 等译. 清华大学出版社, 2005

[6] Colyer A, Clement A. Eclipse AspectJ 中文版[M]. 钱竹青, 邹正武, 等译. 清华大学出版社, 2006

[7] Masuhara H, Kiczales G. Modeling Crosscutting in Aspect-Oriented Mechanisms[J]// Cardelli L, ed. ECOO P2003, LNCS. 2003, 2743: 2-28

[8] The AspectJ Team. The AspectJ Programming Guide. Palo Alto Research Center, 2003. http://www.eclipse.org/aspectj/doc/released/progguide/index.html