面向对象程序可视化过程中的一种布局算法*)

A Layout Creation Method for Object-Oriented Program Visualization

钱 宇 李必信 郑国梁

(计算机软件新技术国家重点实验室 南京大学计算机科学与技术系 南京210093)

Abstract Diagrams have been commonly and effectively used in software visualization. The layout method is the critical component that helps to provide a complete, accurate, clear, and beautiful show of the program structure. That will be useful in software comprehension, re-engineering, and reverse engineering. Since object-oriented program has special hierarchical structure and message transferring, the relations among inherited classes, functions, couplings and object references are so complex and hard to understand. Its graph performance will be greatly improved if a good layout method is applied. We present a three-phrase (creation-optimization-drawing) approach for two-dimensioned graphs layout that makes distribution of vertices and edges symmetrical, at the same time the edge crossings are few. This method has been applied in an object-oriented programming support environment (OOPSE) and the performance is fairly good.

Keywords Visualization, Layout, Object-oriented, Software comprehension

1 引言

随着面向对象技术的广泛应用,对面向对象程序的理解也变得日益重要,理解方法之一就是软件可视化,通过对软件所有方面的属性及行为进行图形化显示,包括:设计和分析方法、系统、程序和算法,使软件理解更直观准确。和软件可视化相近的另一个概念是程序可视化,在八十年代后期提出时其意义和软件可视化相同,现在认为是软件可视化中的一个部分,其目的是帮助理解、排错,进行程序性能分析[6],程序可视化意味着更多地考虑具体的程序执行和其数据而不是算法的可视化。从图1可看出软件可视化的分类。

J. J. Van Wijk 将可视化过程概括为四个步骤:数据生成阶段、数据丰富和提炼阶段、可视化匹配阶段和提交阶段[9],要进行可视化的完整工作,不仅要有可视化编程系统支持,还需要运用可视化语言[12],可视化语言有两层含义:第一意味着语言处理的对象是可视的,第二意味着语言本身(符号集)是可视的,前者意味着处理可视信息的语言或叫可视信息处理语言,后者意味着用可视表达式编程的语言或叫可视编程语言。一般情况下的可视化工作是通过图表用户界面(diagrammatic user interface,DUI)来完成的,它是一个用来处理特定图表的工具集合,通常被嵌在一个大工具包里用来生成和处理图形并和用户交互。比如 Danny B. Lange 和 Yuichi Nakamura 的 Program Explorer[7],一种用于 C++ 的程序可视化处理器,它把面向对象程序中的动态信息(对象)和静态信息(类)结合起来进行程序分析和理解,使得对 C++ 程序的理解方便、快捷、高效。在可视化过程中,图形的生成是必不可少的,而一个图形的布局直接关系到信息表达是否准确和清晰,结点和边如何分布决定了这个图的易读性和使用效率,所以布局方法是可视化过程中非常重要的一个部分。根据 Bernd Meyer 在文[1]中的提法,一个理想的图形布局应该具备这样的性质:整个图形是匀称的;结点分布均匀;边的长度均衡;边交叉数最小化;曲线或折线数少。

还有其他一些要求:信息的一致性,图形的易控制性和合理的响应时间,对于设计者而言还要有易重用性、易扩展性和一定的灵活性。T. Lin 和 Peter Eades 在文[2]中把传统布局方法分成两大类:

1)基于算法的布局:开发人员非常清楚布局要达到的特定要求,然后针对这些要求设计相应的算法,并

*)本文工作得到江苏省自然科学基金(编号 BK99038)资助。钱 宇 硕士研究生,研究方向为程序理解,可视化技术;李必信 博士研究生,研究方向为面向对象支撑技术及其环境和工具,程序理解等;郑国梁 教授,博士生导师,研究方向为软件工程,面向对象技术等。

试图使计算资源最小化，因此能获得非常高的易读性和效率。算法通用性不强。

2)基于声明的布局：背景是人工智能，注重表现和一般性，最大化灵活性，给予使用者一个良好的控制机制。然而通常效率不高，甚至不能保证得到结果。

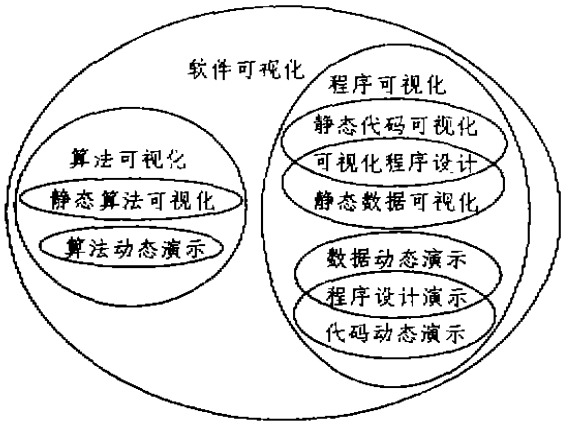

图1 软件可视化分类

本文要介绍的COD算法在基本思想上更接近于第一类，主要是针对面向对象程序的类继承图和函数调用图，但扩展了通用性，在设计时尽量使各步骤无关，起到限制作用的只是步骤间接口的数据结构，因此并不仅限于层次性很强的图，只要是用链表作为数据结构的图就都能绘制出来。但由于一般的链表只能表示二维图，所以COD算法仅限于绘制二维图形。

2 COD(Creation-Optimization-Drawing)算法

算法的思想是首先调整结点的上下关系，亦即层次关系；然后由下至上一层层地调整层中每个结点的左右关系，使交叉数最少；最后根据具体窗口和结点及边数目算出具体坐标绘制图形。整个算法定义了四个全局数组变量：

```
LineCount=new int[n+1];
To=new int[n+1];
LayOut=new int[(n+1)*(n+1)];
Relation=new int[(n+1)*(n+1)];
```

其中n为结点个数。由于只有n个结点，所以在横向上和纵向上都不会超过n层或n个，这是将LayOut定义为(n+1)*(n+1)的原因。LayOut记录了各个结点在屏幕上的位置，Relation则记录了各个位置间的关系(如Virtual调用)，LineCount记录每一个层上有几个结点，To是在图形优化时用来记录每层已优化过的结点个数。

2.1 生成算法(Creation)

目的是正确反映结点层次关系，同时使结点上下分布均匀。

输入：一个链表，其各个节点即为要绘制的图形中的结点，节点的属性中包含了此节点与其他节点的关系信息，比如说类继承图中的类，其父类的名称会被记录在表示这个类的节点的属性中。

输出：一个按结点纵向关系分层的图形，即一个纵向关系已调整完毕的图，可用一个二维数组(LayOut)来表示，比如说一个继承层次已调整好的类继承图。

首先通过循环遍历数据源链表，调用深度函数Depth求出每一个结点的深度也就是它在LayOut中所处的行，把它的ID依次赋给LayOut的相应位置，就得到了层次性已确定的LayOut，同时生成了一个一维数组LineCount[i]记录第i行有几个结点。

结点的深度采用递归计算，以类继承图为例：

一个类如果无父类(顶层类)，其深度等于0；有父类的则深度等于其深度最大的父类的深度加1。即：

```
Depth(class)=Depth(max(Depth(class's Father1),
Depth(Class's Father2),…))+1 if class is a subclass or
Depth(class)=0 if class is a top class;
I.给LineCount,LayOut,Relation赋初值0;
I.遍历链表调用深度函数求出每一个结点的深度，深度对应的层的结点个数加1，将该结点ID赋给LayOut的相应位置：
i from 1 to n
begin
        int L←Depth(Node[i]+1);
        LineCount[L]++;
        LayOut[L*(n+1),LineCount[L]]←Node[i].ID;
end
```

2.2 优化算法(Optimization)

目的是减少交叉，同时使结点左右分布均匀。即便是最基本的布局优化要求比如边交叉数最小化或边最大长度最小化都是NP完全问题[11]，所以无法对这个算法给出证明。

输入：生成算法的输出

输出：一个按结点横向关系排列好的图形，即一个横向关系已调整好的图，同样用二维数组来表示，比如说一个交叉数最少的图形。

优化方向：先自底向上后自顶向下(在类继承图例子中因为数据源中给的是父类信息，由于C++语言本身的语法特性，可以很容易地通过源程序扫描得到一个类的直接父类，信息的获取方式是从子类到父类，故未用自顶向下)

优化方法：向左靠齐(与向右靠齐等价)

从最下一层开始，用一个一维数组To记录每一行已挪动过的位置，避免层之间的重复挪动增加开销。使用一个四重循环，前两重遍历除最顶层外中的每一个结点；后两重负责检查除最底层外每一个未挪动过的结点是否应该和其后的结点交换位置。简而

言之优化就是把相互之间有关系的结点尽可能地放在一起。

```
Ⅰ.给数组To赋初值0;
Ⅱ.自底向上优化,n为结点总个数,LineCount[i]为第i行的结点个数
for(int loop0=n;loop0>1;loop0--)
{for(int loop1=1;loop1<=lineCount[loop0];loop1++)
{for(int loop2=loop0-1;loop2>0;loop2--)

 int count=To[loop2];. *Count位置前是已挪过的结点,不能再挪动*/
 for(int loop3=count+1;loop3<=LineCount[loop2];
 loop3++)
    {
     if在[loop0,loop1]位置的结点和位置[loop2,loop3]结点之间有连线
     then
     将[loop2,loop3]位置的结点和[loop2,count+1]位置的结点交换;
     count++;
    }
  To[loop2]=count;/*记录下这一行已挪动到什么位置*/
 }
}
Ⅲ.自顶向下优化(可选),将loop0的循环范围变为从1到n,其余同Ⅱ.
```

2.3 绘制算法(Drawing)

目的是根据具体显示环境得出窗口坐标,得到较好的显示效果。

输入:优化算法的输出,即一个记录了各结点相对位置的二维数组。

输出:记录了各个结点的窗口坐标和结点间线段坐标的数组。

算法根据四个全局变量的值进行布局,计算各个结点在窗口上的位置坐标,计算各个结点间连线的起始坐标与终止坐标。然后绘图。

结点的相对位置已经确定,Drawing算法所做的只是将它变成绝对位置,Relation[i,j]的值即为结点i和结点j的关系,比如用1代表一类关系,2代表另一类关系,在绘制时根据数值绘制不同的线条,比如颜色、粗细或实虚线等。

假设结点用矩形来表示,窗口长为XLength,高为Ylength,

位置(i,j)的结点的Rectangle[(x1,y1),(x2,y2)]坐标为:

```
x1=int((Xlength/(LineCount[i]*2+1))*(2*j-1));
x2=int((Xlength/(LineCount[i]*2+1))*2*j);
y1=int(YLength/((2*n-1))*(2*i-2));
y2=int(YLength/((2*n-1))*(2*i-1));
```

有了结点的坐标,根据Relation就可计算出结点间线条的起始坐标和终止坐标。

3 和几种常见算法的比较

3.1 和Force-Directed Layout的比较

Peter Eades在文[6]中提出的一个弹簧嵌入模型(sprug-embedder model)是目前比较成熟的布局算法,由它引发了许多弹力制导的布局算法(Force-Directed Layout):把每一个结点看成是一个钢环,边被看成是连接各个钢环的弹簧,开始时结点可分布在任意位置,通过计算作用在环上的力来移动环,该算法就是一个不断减少这个物理系统能量状态的循环过程,最终达到平衡。从几何学上来看这意味着结点和边长能够均匀分布而不需要确定图形各个结点的分布,因为一个物理系统是可以有多个平衡态的。这是这个算法的最大优点。然而缺点也很明显:1)计算量大,提高这个算法的效率一直是布局方面文章讨论的热点,事实上,通用就是以牺牲效率为代价的。2)对于层次结构比较特殊的图形该算法不适用。

3.2 和J. H. CROSS的语言无关的布局比较

J. H. CROSS提出了一种中间语言Markup Language[3],然后给出了针对这种语言的可视化过程,当然其中包括图形生成的布局,实际上他是把可视化分成了两个阶段,首先将程序翻译成标准语言程序,然后再进行可视化的工作,这样的布局方法就不再依赖于程序设计语言,而只要研究Markup Language的结构就可以了。这的确反映了将来可视化发展的一个趋势,但目前第一步的工作还是很困难的。

3.3 和ALF(Automatic Layout Facility)的比较

Paola Bertolazzi和G. D. Battista等人在文[4]中假设有一个针对不同语言的布局算法库,然后设计了一个ALF来管理这个库,用户可见的完全是ALF在工作,和J. H. CROSS的方法比这无疑又是一种解决思路,然而建立不同语言的布局算法库的工作量之大可想而知。

3.4 和Sugiyama method的比较

Sugiyama method[10]是针对特定结构的图形的布局,要求图形具有严格的层次结构并由此可以逐层地计算来达到布局的优化,优化仅在层内进行,这样就避免了做繁杂的全局优化,应该说在这一点上COD算法和它是一样的,但在做层内优化时Sugiyama method是通过将结点从左向右排来减少交叉[5],结点序列并不一定是确定和最优的排列,而且Sugiyama method应用范围有限。

COD算法尽管是针对面向对象程序的,但由于面向对象技术的广泛使用,当今大多数程序都是用面向对象语言书写的,所以COD算法针对面并不狭窄,同时计算量也不太大,层次结构无论多特殊只要在初始的关系链表中给出就都能绘制出来。

4 在OOPSE中的作用

OOPSE (Object-Oriented Programming Support

Environment)是一个针对C++语言的程序分析和理解工具，总体结构如图2。COD布局算法用在图形生成部分，得到的一个例子C++程序的类继承图，当鼠标指向某一个类时，在文本窗口显示出这个类的基本信息。

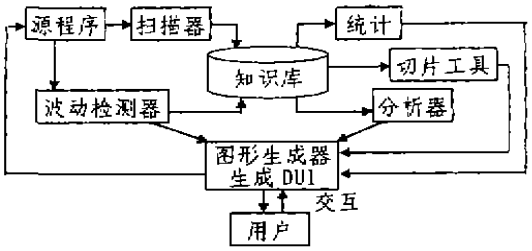

图2 OOPSE总体结构

结论 本文提出了一种层次结构图形的布局算法，对于典型的面向对象语言程序的二维图形生成，此种实现方法的思想都是可应用的，其最大优点是由于将布局阶段的工作提出来单独完成，在前端有一个数据源接口，后端有一个绘图接口，优化不仅和语言无关同时占用计算资源极少，效率很高，响应时间极短，然而由于毕竟是基于算法的布局，对数据源的数据结构还是有一定要求，并且在绘图接口还没有应用处理大数量结点的方法，这也是今后改进的一个主要方向。

参 考 文 献

1 Meyer B. Competitive Learning of Network Diagram Layout. In: Proc. 1998 IEEE Symposium on Visual Languages

2 Lin T, Eades P. Layout Creation Methods For Software Visualization. Software Visualization, World Scientific Publishing Co Pte Ltd, 1996. 61～82

3 Cross J H, Hendrix T D Language Independent Program Visualization. Software Visualization, World Scientific Publishing Co. Pte Ltd, 1996. 27～45

4 Bertolazzi P, et al. Parametric graph drawing. IEEE Trans. on Software Engineering, 1995, 21(8): 662～673

5 Eades P, Sugiyama K. How to draw a directed graph. Journal of Information Processing, 1991. 424～437

6 Eades P. A heuristic for graph drawing. Congressus Numerantium, 1984, 42: 149～160

7 Lange D B, Nakamura Y. Program Explorer A Program Visualizer for C++. USENIX Association, Conference on Object-Oriented Technologies(COOTS) 1995. 26～29

8 Oudshoorn M J, et al. Aspects and Taxonomy of Program Visualization. Software Visualization. World Scientific Publishing Co. Pte. Ltd., 1996. 3～26

9 Van Wijk J J. Scientific Visualization. Computer Systems and Software Engineering, Kluwer Academic Publishers, 1992. 387～396

10 Sugiyama K. A cognitive approach for graph drawing. Cybernetics and Systems: An International Journal, 1987, 18, 447～488

11 DiBattista G, et al. Algorithms for drawing graphs: an annotated bibliography. Journal of Computational Geometry Theory and Applications, 1994, 4: 235～282

12 Price B, et al. An Introduction to Software Visualization. Software Visualization Programming as a Multimedia Experience, The MIT Press, 1998. 3～28

(上接第29页)

7 Weyuker E Evaluating Software Complexity Measures. IEEE Trans. On Software Eng, 1998, 14: 1365～1375

8 Kolling M, Risenberg J. Support for Object-Oriented Testing. TOOLS 28, Noverber 1998. 23～26

9 MacDonald A, Carrington D. Object-Oriented Design. TOOLS 28, Noverber 1998. 23～26

10 Periyasamy K, Liu X. A New Metrics Set for Evaluating Testing Effort for Object-Oriented Programs. IEEE TSE, 1999. 84～93

11 Yacoub S M, et al Dynamic Metrics for Object Oriented Designs. IEEE Trans. Sotware Eng, 1999. 50～61

12 Fenton N E. Software Measurement: A Necessary Scientific Basis, IEEE Trans. Software Eng, 1994, 20(3): 199～206

13 Baker A L, et al. A Philosophy for Software Measurement, J. System Software, 1990, 12: 227～281

14 Fenton N E. Software Metrics. A Rigorous Approach. Chapman and Hall, London, 1991. 337

15 Kearney J K, et al. Software Complexity Measurement. Comn. ACM, 1986, 29: 1044～1050

16 Zues H. Software Complexity. Mesures and Methods. Walter de Gruyter, Berlin, 1990. 605

17 Halstead M H. Elements of Software Science. New York: Elsevier North—Holland, 1997

18 Macabe T J. A Complexity Measurement. IEEE Trans. Software Eng., 1976, 2(4): 308～320

19 Albrecht A J. Measuring Application Development Productivity, In: Proc. Join SHARE/GUIDE/IBM Application Development Symposium. 1979. 34～43

20 徐家福，张幸儿. 程序复性度量. 南京大学学报，1979

21 袁峰，陈佩佩，徐家福. 程序复杂性量度的一些分析. 计算机应用与软件，1984，6

22 王振宇. 树的枚举与算法复杂性分析. 国防工业出版社