

面向可维护性的软件体系结构设计

胡文生¹, 杨剑锋¹, 赵 明^{2,3}

(1 贵州理工学院 信息工程学院, 贵州 贵阳 550003; 2 贵州省可靠性工程中心,
贵州 贵阳 550005; 3 瑞典耶夫勒大学, 瑞典 耶夫勒 SE-80176)

摘 要: 软件产品大部分故障来源于需求阶段和设计阶段, 为了提高软件系统的可维护性、在设计阶段必须采用面向可维护性的软件体系结构, 而提高软件可维护性最佳解决方案是软件系统采用模块化设计. 通过将 UML 类图转化为带权的有向图, 利用 Dijkstra 算法计算出类图中类与类之间的关系权重系数矩阵, 对关系权重系数矩阵实施聚类算法一定能将功能相似的类聚合在一起构成组件, 从而实现软件系统模块化设计.

关键词: 可维护性; 体系结构; 有向图; UML 类图

中图分类号: TP311

文献标识码: A

文章编号: 1000—7180(2016)07—0145—05

Software Architecture Design for Maintainability

HU Wen-sheng¹, YANG Jian-feng¹, ZHAO Min^{2,3}

(1 Faculty of Information Engineering, Guizhou Institute of Technology, Guizhou 550003, China;
2 Reliability Engineering Research Center of Guizhou Province in Guizhou University, Guizhou 550005, China;
3 Department of Technology, University of Gävle, Gävle SE-80176, Sweden)

Abstract: The most of software fault comes from the requirements phase and design phase. To improve the maintainability of the software system, the software architecture for maintainability must be adopted in the design phase. And the best solution of improving the software maintainability is modular design. UML class diagram transformed into the right of directed graph, then the relation weight coefficient matrix between class and class in class diagram can be calculated by Dijkstra algorithm. The clustering algorithm be carried out on the relation weight coefficient matrix. The results of clustering algorithm is that the closely related classes can be clustered into a component. Finally, the software system is modular design.

Key words: maintainability; architecture; directed graph; UML class diagram

1 引言

在软件生命周期中, 设计阶段是一个非常重要的阶段, 它起到承前启后的作用, 主要承担实现需求阶段整理出来的各项需求并以软件体系结构的形式表现出来, 设计阶段产生的软件体系结构是软件开发人员进行源代码编写的指南, 所以设计阶段实施好坏是软件项目是否成功的一个非常重要的前提. 研究表明^[1], 软件设计阶段产生的错误数占软件系统最终错误总数的 27%, 这个比例是相当高

的, 仅次于需求阶段的 56%, 而且设计阶段的错误与需求阶段的错误一样具有涟漪效应, 即设计阶段产生的错误如果没有及时发现将会影响到下游的软件开发、软件测试、软件维护等活动. 所以要想提高软件系统的可维护性、降低软件维护成本、减少软件产品交付后的维护活动、使事后软件维护活动易于进行, 有必要研究可维护性设计, 用“软件可维护性是设计出来的”观点指导设计阶段开展各项工作. 设计阶段最主要的活动就是软件体系结构设计^[2].

软件体系结构作为软件系统的高层设计和早期

设计,在很大程度上将决定未来软件系统的重要特性^[3],因为软件开发都是基于软件体系结构进行的. 开发出来的软件系统是否具有高可维护性,主要是由系统体系结构决定的,所以,要想减少事后软件维护的次数、降低软件维护成本、使事后软件维护活动变得更容易进行、减缓软件架构恶化的速度,就必须在设计阶段提供软件可维护性良好的解决方案. 而提高软件可维护性最佳解决方案是软件系统必须实现模块化设计^[4-5]. 模块化设计最大特点是能够减少模块之间的交互,有利于关注点分隔,这种设计模式不仅有利于组件的重用,而且也有利于提高软件系统的可维护性,使得事后软件维护时,只是在单个模块上进行,从而减少软件维护所造成的影响范围,有利于维护人员集中注意力. 当然,要想实现模块化设计,必须在设计过程中控制好两个设计指标;一是内聚,二是耦合. 衡量模块化设计结果的优劣,通常是以“高内聚、低耦合”的设计目标来考察.

2 UML 类图的模块化设计

2.1 类与类之间的关系

在 UML 类图中,类与类之间的存在四种关系,它们是^[6]:

- (1) 关联关系 (Association, 简称 A);
- (2) 继承关系,有时也叫泛化关系 (Inheritance 简称 I);
- (3) 依赖关系 (Dependency, 简称 D);
- (4) 抽象/实现关系,有时也称抽象/具体关系 (Abstraction/Reality, 简称 R);
- (5) 为了方便对 UML 类图所包含的类进行聚类分析,添加一种新的类与类之间的关系:无关系 (None, 简称 N),即类与类之间不存在任何关系.

2.2 类与类之间关系权重的确定

在图 1 描述的五种关系中,各个关系的强弱是不一致的,其中继承关系与抽象/实现关系被认为是五种关系中最强的关系,因为除了父类的私有成员外,其他公共级别和受保护级别上成员的任何变动都会影响到子类,可以说父类定义了子类的行为,而抽象/实现关系实际上是类与接口之间的关系,该关系也是一种继承关系;其次,关联关系是五种关系中第二强的关系,关联关系是类之间的一种连接,是涉及到此关联的那些类的对象之间的一个语义连接,这种关系是双向的,意思是如果两个类存在关联关系的话,它们是“互相知道”、“被连接在一起”^[7],所以关联关系是仅次于泛化关系、抽象/实现关系的第

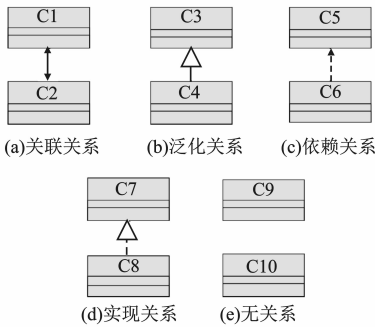


图 1 类图中类与类之间的五种关系图

二强的关系,UML2.0 中类与类之间的关联关系有好几种:双向关联、单向关联、聚合、组合等,本文对这几种关联关系不加以区分;排在第三位的是依赖关系,一个类需要依赖另一个类来达到某种目的,这种依赖关系是单向的;第四种无关系,显然它是最弱的关系. 依据这五种关系的强弱为它们赋予不同的权重,见表 1 所示.

表 1 类与类之间关系权重

类与类之间的关系	权值
继承关系	6
抽象/实现	6
关联关系	4
依赖关系	2
无	0

2.3 类图转换为带权有向图

在 UML 类图中的任意两个类之间的关系一般存在两种情况:直接关系和间接关系. 直接关系如表 1 所列举的五种关系,即泛化、实现、关联、依赖、无;间接关系主要是指在 UML 类图中某两个类通过很多个中间类而发生关系. 如类 A 与类 B 是继承关系,类 A 是父类,而类 B 又依赖类 C,于是类 A 与类 C 之间也存在关系,这种关系是通过类 B 而发生的间接关系. 因此,类与类之间的关系在类图中是比较复杂的,有时候两个类可能是通过多个中间类而发生联系. 为了从错综复杂的关系中找出任意两个类之间的关系,本文提出了一种将 UML 类图转化为带权有向图的方法,利用图论的有关知识来解决这个问题.

UML 类图转化为带权有有向图的过程如下所示.

- (1) 有向图中的结点代表 UML 类图中的类.
- (2) 有向图中的边代表 UML 类图中类与类之间的关系.

(3)有向图中边的权重代表 UML 类图中类与类之间关系权重的倒数.

类图与有向图之间的关系如图 2 所示.

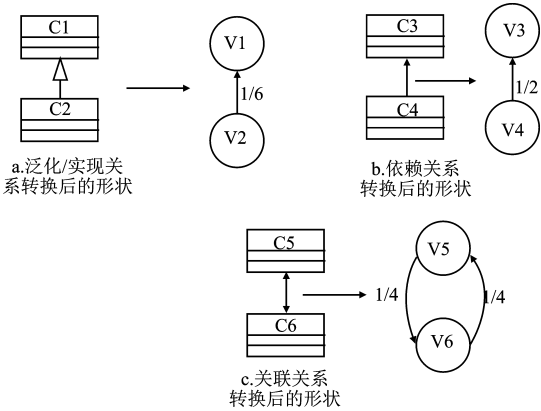


图 2 类图与有向图之间的关系

UML 类图经过上述转化后变成一个带权的有向图 $G=\langle V, E, W\rangle$. 其中 V 是图 G 的结点的集合, E 为边的集合, W 是边的权重的集合, 边的权重代表两个节点之间的距离, 这里将有向图中边的权重定义为类与类之间关系权重的倒数, 能够形象地反映出两个节点所代表的两个类之间的关系紧密程度, 如类 D 与类 C 在类图中是继承关系, 根据表 1 可知, 它们的关系权重为 6, 反映到有向图 G 中是代表类 D 与类 C 的两个节点之间边的权重为 $1/6$, 表示它们的距离很近, 即这两个节点关系很密切, 而如果这两个类之间没有关系, 则根据表 1 可知, 它们的关系权重为 0, 反映到有向图 G 中对应的两个节点之间边的权重为 $1/0=\infty$, 表示它们之间的距离很远.

2.4 基于带权有向图的聚类分析

问题提出:假设某个 UML 类图包含有 k 个类, 记为: $CP=\{C_{m1}, C_{m2}, \cdots, C_{mk}\}$, 如何求出集合 CP 中任意两个类之间的关系权重值.

问题求解如下所示.

(1) 将 UML 类图转化为带权的有向图 $G=\langle V, E, W\rangle$, 转换方式如上节所述.

(2) 在图 G 中找出任意两个结点之间的最短距离值.

最短距离值的计算可以根据 Dijkstra 算法来实现, 算法设计如下所示.

算法 1: PNS 中两两节点最短距离

设 $CP=\{C_{m1}, C_{m2}, \cdots, C_{mk}\}$ 对应图 $G=\langle V, E, W\rangle$ 中的节点集合

$PNS=\{V_{m1}, V_{m2}, \cdots, V_{mk}\}$

Input: PNS and $G=\langle V, E, W\rangle$.

Output: PNS 中两个节点之间的距离最小值

BEGIN

$N \leftarrow PNS = \{\text{mark each node in the PNS unchecked}\}$

WHILE($N_PNS \neq \emptyset$)

BEGIN

Pick one node P_i from the N_PNS and mark it checked

WHILE($(N_PNS \neq \emptyset) \& \&$

(each $P_j \in N_PNS$))

BEGIN

Dijkstra(G, P_i, P_j);

ENDWHILE

ENDWHILE

END

算法 2: Dijkstra 算法

Dijkstra(G, s, t)

Input: $G=\langle V, E, W\rangle$ and $s \in PNS$ and $t \in PNS$.

Output: the minimum distance between s and t .

BEGIN

$dis[v] = \infty (v \in V, v \neq s), dis[s] = 0, S = \{s\};$

$Q = Q - S;$

WHILE($Q \neq \emptyset$)

BEGIN

Pick one node u from the Q

St. $dis[u] = \min\{dis[v], v \in Q\}$

$S = S \cup \{u\};$

$Q = Q - S;$

IF $u = t$ THEN break;

FOR each edge(u, v) outgoing from u

IF $dis[v] > dis[u] + w(u, v)$ THEN

$dis[v] = dis[u] + w(u, v);$

ENDFOR

ENDWHILE

END

(3) 第(2)步结点之间的最短距离的倒数即为结点所对应的类与类之间的关系权值, 如果计算的结果中某两个类之间的关系权值存在两个, 则取其中最大的一个, 将这些关系权值构造成一个关系矩阵.

(4) 对关系矩阵实施聚类分析, 将密切联系的类聚合在一起从而构造出相应的组件.

3 实例分析

假如某个类图为图 3 所示的 8 个类 $C1, C2, C3, C4, C5, C6, C7, C8$, 对这 8 个类实施聚类算法以封装成组件的过程如下所示.

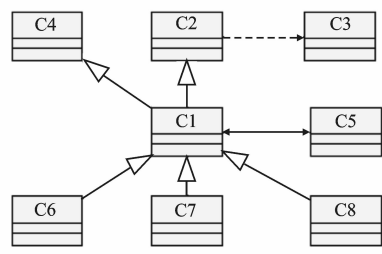


图 3 类图

- (1) 类图转化为一个有向图(其中类 C_i 对应于有向图中顶点 V_i).
- (2) 利用算法 1 和算法 2 计算出有向图 4 中 8 个结点 $V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8$ 两两之间的最短距离值. 具体情况见表 2 所示.

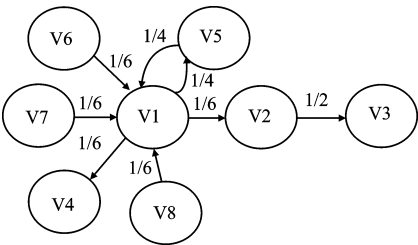


图 4 图 3 对应的带权的有向图

表 2 结点之间的最短距离

编号	结点对	最短距离值	编号	结点对	最短距离值	编号	结点对	最短距离值	编号	结点对	最短距离值
1	(V1,V1)	1	17	(V3,V1)	∞	17	(V5,V1)	1/4	17	(V7,V1)	1/6
2	(V1,V2)	1/6	18	(V3,V2)	∞	18	(V5,V2)	$1/4+1/6$	18	(V7,V2)	$1/6+1/6$
3	(V1,V3)	$1/6+1/2$	19	(V3,V3)	1	19	(V5,V3)	$1/4+1/6+1/2$	19	(V7,V3)	$1/6+1/6+1/2$
4	(V1,V4)	1/6	20	(V3,V4)	∞	20	(V5,V4)	$1/4+1/6$	20	(V7,V4)	$1/6+1/6$
5	(V1,V5)	1/4	21	(V3,V5)	∞	21	(V5,V5)	1	21	(V7,V5)	$1/6+1/4$
6	(V1,V6)	∞	22	(V3,V6)	∞	22	(V5,V6)	∞	22	(V7,V6)	∞
7	(V1,V7)	∞	23	(V3,V7)	∞	23	(V5,V7)	∞	23	(V7,V7)	1
8	(V1,V8)	∞	24	(V3,V8)	∞	24	(V5,V8)	∞	24	(V7,V8)	∞
9	(V2,V1)	∞	25	(V4,V1)	∞	25	(V6,V1)	1/6	25	(V8,V1)	1/6
10	(V2,V2)	1	26	(V4,V2)	∞	26	(V6,V2)	$1/6+1/6$	26	(V8,V2)	$1/6+1/6$
11	(V2,V3)	1/2	27	(V4,V3)	∞	27	(V6,V3)	$1/6+1/6+1/2$	27	(V8,V3)	$1/6+1/6+1/2$
12	(V2,V4)	∞	28	(V4,V4)	1	28	(V6,V4)	$1/6+1/6$	28	(V8,V4)	$1/6+1/6$
13	(V2,V5)	∞	29	(V4,V5)	∞	29	(V6,V5)	$1/6+1/4$	29	(V8,V5)	$1/6+1/4$
14	(V2,V6)	∞	30	(V4,V6)	∞	30	(V6,V6)	1	30	(V8,V6)	∞
15	(V2,V7)	∞	31	(V4,V7)	∞	31	(V6,V7)	∞	31	(V8,V7)	∞
16	(V2,V8)	∞	32	(V4,V8)	∞	32	(V6,V8)	∞	32	(V8,V8)	1

表 3 类与类之间的关系矩阵

编号	类	C1	C2	C3	C4	C5	C6	C7	C8
1	C1	1	6	1.5	6	4	6	6	6
2	C2	6	1	2	0	2.4	3	3	3
3	C3	1.5	2	1	0	1.1	1.2	1.2	1.2
4	C4	6	0	0	1	2.4	3	3	3
5	C5	4	2.4	1.1	2.4	1	2.4	2.4	2.4
6	C6	6	3	1.2	3	2.4	1	0	0
7	C7	6	3	1.2	3	2.4	0	1	0
8	C8	6	3	1.2	3	2.4	0	0	1

(4) 使用统计工具对表 3 得到的类与类之间的关系权重矩阵实施聚类分析,得到如图 5 所示的树状图。

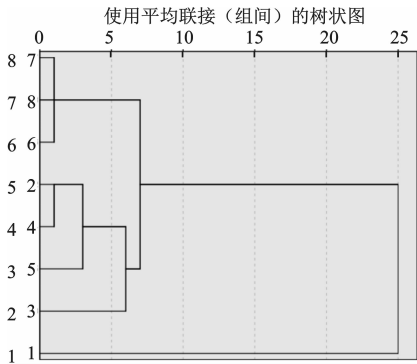


图 5 聚类后的树状图

设置重新调整距离为 15,从图 5 可以看出,该软件系统可以划分为三个组件: $S_1 = \{C_1\}$; $S_2 = \{C_2, C_3, C_4, C_5\}$; $S_3 = \{C_6, C_7, C_8\}$. 从图 3 所示的类图来看这种划分组件的方法是合理的,因为类 C_1 在整个类图中是属于关键类,它的修改变动会影响类 C_2, C_3, C_4, C_5 ,而类 C_6, C_7, C_8 的修改变动会影响类 C_1 ,从而间接影响类 C_2, C_3, C_4, C_5 ,所以可以单独将类 C_1 作为一个组件,有利于软件系统交付后的维护活动;而把类 C_2, C_3, C_4, C_5 聚合在一个组件上是因为它们都受到类 C_1 的影响;类 C_6, C_7, C_8 聚合在一个组件上,是因为它们具有共同的特征,就是它们的每一个修改变动都会影响类 C_1 ,从而进一步影响类 C_2, C_3, C_4, C_5 .

4 结束语

UML 类图是软件设计阶段最重要的产品,为了提高软件系统的可维护性,减少软件产品交付后的维护活动、降低软件维护成本、使维护活动易于进行,模块化设计是最佳的设计方案,如何组织 UML 类图中的类,以便构造可以再用、易于维护的组件,本文提出了一种基于 UML 类图中类与类之间的关

系将 UML 类图转化为带权的有向图,利用图论的相关理论最终实现将关系相近的类组合成一个组件,从而实现软件系统模块化设计的目的。

参考文献:

[1] Wensheng Hu, Ming Zhao, Shiyi Wu, Jianfeng yang. Requires analysis based on software maintainability [C]// Reliability, Maintainability and Safety(ICRMS),2014 International Conference Washington, DC: IEEE 2014: 354-357.

[2] GAOHui, ZHANG Li, Li Lin. Software flexibility prediction model in software architecture level[J]. Journal of Software,2010;21(9):2118-2134.

[3] 杨芙清,梅宏. 构件化软件设计与实现[M]. 北京:清华大学出版社,2008.

[4] Griswold W G, Sullivan KJ, Song Y,et al. Modular Software Design with Crosscutting Interfaces [J]. IEEE Software,2006,23(1):51-60.

[5] Olaf Seng, Markus Bauer,Matthias Biehl, et al. Search-based improvement of subsystem decomposition [J]. In GECCO'05:Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA: ACM Press,2005:1045-1051.

[6] Zhang Yan, Mei Hong. Non-functional attributes oriented description and verification in UML class diagrams [J]. Journal of Software, 2009, 20 (6): 1457-1469.

[7] WangXB,Wang H,Liu C. Automatic Hierarchical Layout Algorithm for UML Class Diagram[J]. Journal of Software, 2009,20(6):1487-1498.

作者简介:

胡文生 男,(1969-),博士,副教授.研究方向为软件可靠性、可维护性. E-mail:1204489713@qq.com

杨剑锋 男,(1987-),博士,副教授.研究方向为软件可靠性、可维护性.

赵 明 男,(1962-),教授,博士生导师.研究方向为软件可靠性、应用统计.

(上接第 144 页)

[5] AEEC. Avionics Application 65394 Software Standard Interface Part 1 - Required Services[S]. 2015-12-01.

[6] AEEC. Avionics Application 65394 Software Standard Interface Part 2 - Extened Services[S]. 2015-01-22.

[7] AEEC. Avionics Application 65394 Software Standard Interface Part 3 - Conformity Test Specification[S]. 2016-10-11.

[8] AEEC. Avionics Application 65394 Software Standard

Interface Part 4 - Subset Services[S]. 2012-06-28.

[9] QNX. Adaptive Partitioning User's Guide[R]. 2014-10-16.

作者简介:

郝继锋 男,(1982-),硕士,工程师.研究方向为嵌入式实时操作系统、TCP/IP 网络协议栈、UNIX/Linux 环境编程. E-mail:haojifeng1187@163.com.