第35卷 第24期 计 算 机 工 程 2009年12月
Vol.35 No.24 Computer Engineering December 2009
·软件技术与数据库· 文章编号：1000—3428(2009)24—0108—03 文献标识码：A 中图分类号：N945

基于SFTA的桥接模式安全性分析

李国旗，陆民燕，刘 斌
(北京航空航天大学工程系统工程系，北京 100191)

摘 要：采用软件故障树分析法，通过一个应用桥接模式的实例研究在软件设计中引入设计模式对软件安全性的影响。结果表明，单纯引入桥接模式，软件的安全性约降低50%，但引入设计模式使得软件模块之间解耦合，通过加入双余量设计，可以使软件安全性提高2个数量级。该结论对安全关键软件面向对象的设计具有指导作用。

关键词：软件故障树分析；设计模式；软件安全性

Safety Analysis of Bridge Pattern Based on SFTA

LI Guo-qi, LU Min-yan, LIU Bin
(Department of System Engineering of Engineering Technology, Beihang University, Beijing 100191)

【Abstract】 In order to analyze the influence of applying design patterns to software safety in software design, this paper carries out a case study on bridge pattern with Software Fault Tree Analysis(SFTA). The result shows that the application of bridge pattern makes the safety reduce by 50%, but the software modules can be decoupled and redundancy design can be added. Introducing double redundancy can improve software safety by two orders of magnitude. The conclusion is valuable for object-oriented design of safety-critical software.

【Key words】 Software Fault Tree Analysis(SFTA); design pattern; software safety

1 概述

软件安全性是指软件不发生导致人员伤亡、设备损坏或财产损失等意外事件的能力。软件本身对人员财产不会产生威胁，但软件中的错误有可能通过软、硬件的接口使硬件发生误动或失效，造成严重的安全事故。安全关键软件在开发的过程中针对软件安全性进行专门的安全性分析。由于在安全关键的计算机应用系统中，由软件实现的功能所占比重越来越大，因此软件安全性受到越来越多的重视。

软件安全性分析技术主要分为静态分析技术和动态分析技术，静态分析技术有软件故障树分析(Software Fault Tree Analysis, SFTA)[1]、软件失效模式与影响分析(Software Failure Mode and Effects Analysis, SFMEA)等；动态分析技术有基于Petri网的安全性分析等。软件安全性分析是贯穿软件生命周期整个过程的，在软件设计阶段，软件安全性分析技术为软件设计提供安全性评估和设计需求的参考。无论是软件安全性的分析技术还是软件的设计技术都有众多的研究和技术积累，当前的难点在于两者的结合点上，属于交叉和边缘问题，需要重点研究。

设计模式是一套被反复使用、经过分类编目的代码设计经验的总结。是被认为合理、有效、可被重复使用的面向对象的软件架构模块[2]，将其应用到软件的设计过程中可以有效改善设计。本文通过一个应用桥接模式的实例，研究引入设计模式后可能对软件安全性产生的影响。

桥接模式是功能强大、应用广泛的一种设计模式，下文利用软件安全性分析的静态分析工具SFTA分析了桥接模式引入前后对软件的安全性产生的影响。

2 桥接模式引入前后的安全性分析

2.1 应用桥接模式的软件实例

假设一个安全关键软件的功能需求为：软件的基本功能是用不同的图形表示不同的信号；同样的信号有不同的实现方法。软件的设计需求类似于一个绘图程序，它可以画圆形、方形等形状，同时画图程序支持不同版本的函数库，比如V1版、V2版。如果不采用设计模式，而只使用传统的面向对象设计，就会选择类继承，生成V1Rectangle, V2Rectangle, V1Circle, V2Circle等类。图1为用类继承实现设计的UML类图。

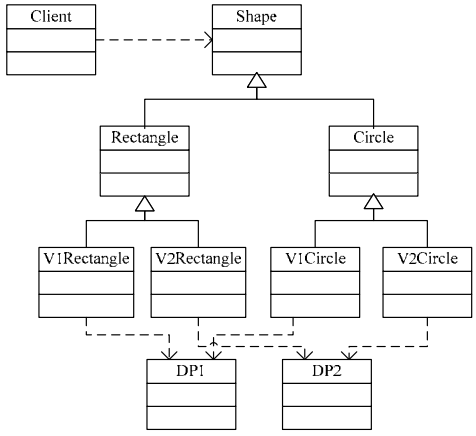

图1 引入桥接模式前的UML类图

使用图1所示类库的客户类的代码如下：

```
class Client {
    public static void main(String argv[]){
```

基金项目：北京航空航天大学青年创新基金资助项目“基于CORBA的可重用组件的可靠性与安全性分析研究”(2008030)

作者简介：李国旗(1977－)，男，讲师、博士，主研方向：软件工程，软件安全性分析；陆民燕、刘 斌，教授、博士生导师

收稿日期：2009-04-10 **E-mail：**keep_thinking@hotmail.com

```
        Shape shape[] = new Shape[2];
        Shape[0] = new V1Rectangle( );
        Shape[1] = new V2Circle ( );
    for(int i=0; i<2; i++)
    Shape[i].drawing();
    }
    }
```

由上述代码画出的顺序图如图 2 所示(简单起见，本文只给出画矩形功能的顺序图，画其他形状的顺序图与此类似)。

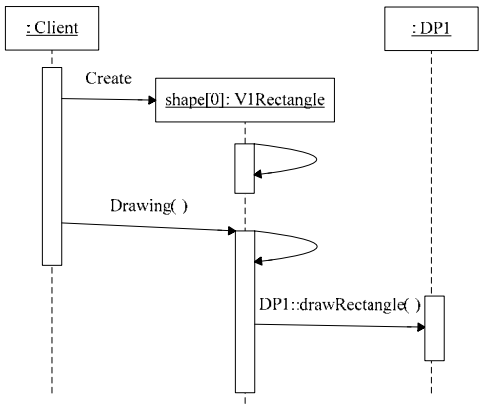

图 2　引入桥接模式前画矩形功能的顺序图

图 1 所示的用类继承实现的设计存在缺陷，假如需要在矩形和圆形的基础上再增加个三角形，就要添加 V1Triangle, V2Triangle；再增加一个版本的画图函数库就必须给每个形状增加一个类。以这种模式扩展程序最后的结果就是类爆炸。而 Bridge 模式将函数库的变化从 Shape 中分离出来，作为一个 Drawing 的接口。Client 调用 Shape 类，Shape 接口再自己调用 Drawing 接口。图 3 为引入桥接模式后的 UML 类图。

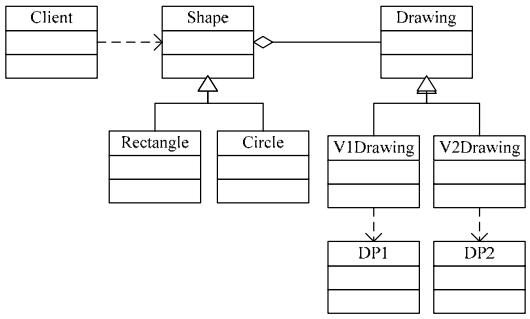

图 3　引入桥接模式后的 UML 类图

引入桥接模式后类库的客户类的代码如下：

```
class Client {
    public static void main(String argv[]){
        Shape shape[] = new Shape[2];
        Drawing drawing[] = new Drawing[2];
        Drawing[0] = new V1Drawing();
        Drawing[1] = new V2Drawing();
        shape[0] = new Rectangle(drawing[0]);
        shape[1] = new Circle (drawing[1]);
        for(int i=0; i<2; i++)
shape[i].drawing();      }
}
```

根据以上代码画出的顺序图如图 4 所示(为了简单清楚，本文只给出了画矩形功能的顺序图，画其他形状的顺序图与此类似)。

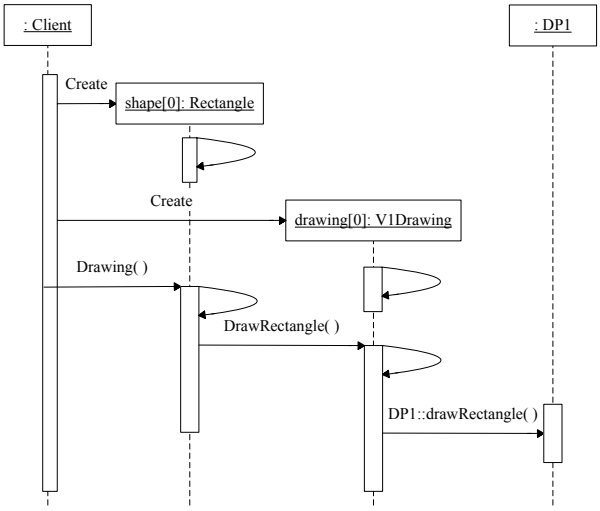

图 4　引入桥接模式后画矩形功能的顺序图

引入设计模式后，画特定的图形和使用特定的画图函数版本这 2 个事务被分离开，从而可以加入容错设计，即一个画图函数库失效时可以使用另一个画图函数库。

加入容错设计后的代码如下：

```
class Client {
    public static void main(String argv[]){
        Shape shape[] = new Shape[2];
        Drawing drawing[] = new Drawing[2];
       Drawing[0] = new V1Drawing();
       v2drawing[1] = new V2Drawing();
       for(int i=0; i<2; i++){
           shape[i] = new Rectangle(drawing[0]);
                if( fail = shape[i].drawing()){
                    shape[i] = new
 Rectangle(drawing[(0+1)%2]);
                    shape[i].drawing();                    }
        }
        for(int i=0; i<2; i++)
shape[i].drawing();
    }}
```

2.2　UML 顺序图到 SFTA 的转化

SFTA 可以由 UML 图直接得到。具体地说，可以由 UML 的顺序图或 UML 状态转移图依据相应的转化规则画出 SFTA[3]，在本文的实例中，SFTA 由 UML 的顺序图得到。UML 顺序图到 SFTA 的转化规则定义如图 5、图 6 所示，这 2 个转化规则的建立参考了文献[3]。由此得到的桥接模式引入前、引入后的 SFTA 如图 7、图 8 所示。

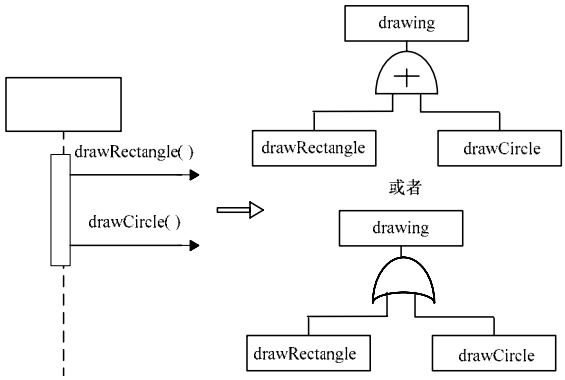

图 5　UML 顺序图到 SFTA 的转化规则 1

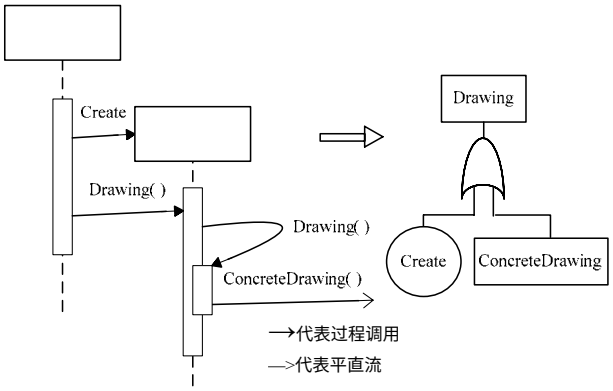

图 6　UML 顺序图到 SFTA 的转化规则 2

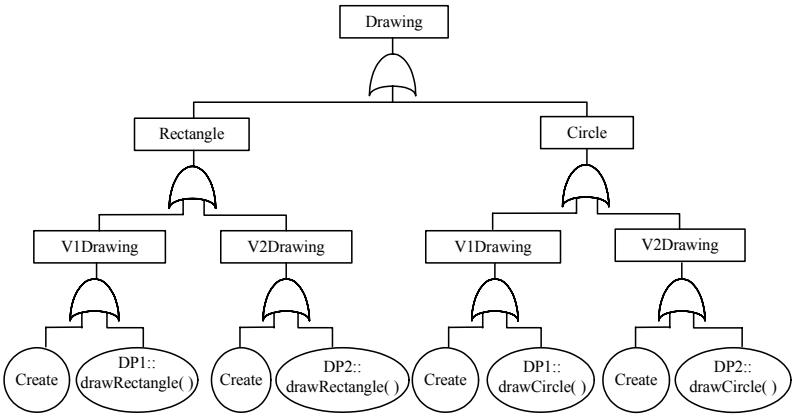

图 7　引入桥接模式前的 SFTA

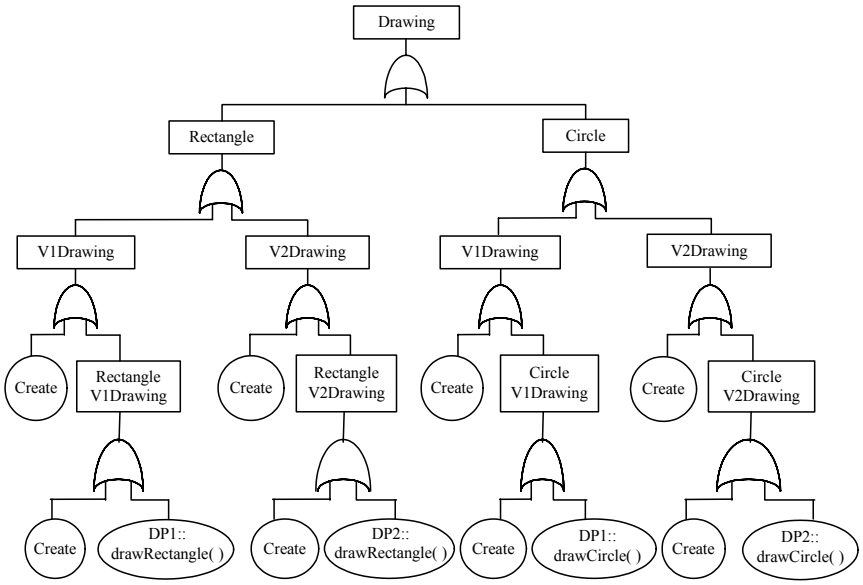

图 8　引入桥接模式后的 SFTA

加入冗余设计后的 SFTA 只是把图 8 中第 2 层的逻辑门由或门改为与门，这里不再画出。

2.3　基于 SFTA 的安全性分析

故障树的评定包括定性和定量 2 个方面，定性评定的方法有：求出所有的最小割集，基于 Metric[4]的方法等；定量评定即根据基本事件发生的概率求出顶端事件发生的概率。

首先将图 7、图 8 的故障树进行简化。顶端事件用 T 表示，基本事件按照从左向右的顺序从 1 开始编号，则引入桥接模式前、引入桥接模式后和加入冗余设计后的顶端事件分别为

$$T_1 = x_1 \cup x_2 \cup x_3 \cup x_4 \cup x_5 \cup x_6 \cup x_7 \cup x_8 = \bigcup_{i=1}^{8} x_i$$

$$T_2 = \bigcup_{i=1}^{12} x_i$$

$$T_3 = \left(\bigcup_{i=1}^{6} x_i\right) \cap \left(\bigcup_{j=7}^{12} x_j\right)$$

由概率论知识可知：$P\{x \cap y\} = P\{x\}P\{y\}$。

实际中常见的系统其基本事件的失效概率都比较小，可以使用下面的近似方法：

$$P\{x \cup y\} = 1-\left(1-P\{x\}\right)\left(1-P\{y\}\right) =$$

$$P\{x\} + P\{y\} - P\{x\}P\{y\} \approx P\{x\} + P\{y\}$$

基本事件 $P\{x\}$ 和 $P\{y\}$ 大约在 10^{-2} 的数量级[5]，引入桥接模式前、后和加入冗余设计后的顶端事件发生的概率可近似为

$$P\{T_1\} = \sum_{i=1}^{8} P\{x_i\} = 8\times10^{-2}$$

$$P\{T_2\} = \sum_{i=1}^{12} P\{x_i\} = 12\times10^{-2}$$

$$P\{T_3\} = \sum_{i=1}^{6} P\{x_i\} \sum_{j=7}^{12} P\{x_j\} =$$

$$36\times\left(10^{-2}\right)^2 = 36\times10^{-4}$$

在该实例中，引入桥接模式使得软件故障发生的概率由 8×10^{-2} 提高到 12×10^{-2}，安全性降低了 50%；但是引入设计模式使得软件模块之间解耦合，从而可以加入容错设计，而容错设计的加入可以使软件故障发生的概率由 12×10^{-2} 降低到 36×10^{-4}，软件的安全性提高了 2 个数量级。

3　结束语

设计模式的核心思想之一是在类结构的设计中优先使用类组合而不是类继承。而类组合的引入一般会增加函数调用的深度。从函数的调用次序容易看出，这种设计方式本质上是通过分治策略实现事务分离；而从软件安全性的角度来讲，事务分离本身并不能提高安全性，为了提高安全性，需要针对安全性需求进行设计方面的改进。从故障树上来看，就是将逻辑“或门”改为逻辑“与门”，从而有效地提高软件的安全性。

从方法论的角度进一步分析。为了控制复杂性而引入的针对事务分离的设计往往会增加软件系统的体积，进一步提高了系统的复杂性；从故障树上来看，就是故障树的高度增加了，这显然会降低软件的安全性。可是另一方面，优秀的设计可以有效地降低系统子模块的耦合度，使得可以通过加入容错设计，提高系统的余量来提高安全性，最终提高软件的安全性指标。

后续的研究将分别选择合适的软件安全性分析技术分析各个重要的设计模式，最终得到所有经典设计模式的安全性设计指南，为软件设计技术和软件安全性分析技术的交叉点的研究和应用提供参考。

(下转第 113 页)