

母线设计模式的分析与研究

赵冉¹, 张代远^{1,2,3}

- (1. 南京邮电大学 计算机学院, 江苏 南京 210003;
2. 江苏省无线传感网高技术研究重点实验室, 江苏 南京 210003;
3. 南京邮电大学 计算机技术研究所, 江苏 南京 210003)

摘 要:设计模式可以用来解决软件设计过程中反复出现的问题,并且使用设计模式,可以有效地提高软件的可复用性、可靠性和可维护性。设计面向对象软件比较困难,而设计可复用的面向对象软件就更加困难。设计模式使人们可以更加方便地复用成功的设计和系统结构。文中介绍了一种解决多代软件系统版本问题的设计模式,研究了协调版本、继承和演变的问题。并提出了一种模型,用来评估此设计模式。初步的结果表明该模式具有高的可扩展性和有效性。

关键词:母线;设计模式;软件可靠性;代组件

中图分类号:TP31

文献标识码:A

文章编号:1673-629X(2013)07-0066-04

doi:10.3969/j.issn.1673-629X.2013.07.016

Analysis and Study of Generatrix Design Pattern

ZHAO Ran¹, ZHANG Dai-yuan^{1,2,3}

- (1. College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;
2. Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing 210003, China;
3. Institute of Computer Technology, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: Design patterns are effective solutions to common problems in software development. They are used to improve software reusability, reliability and maintainability. Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. Design patterns make it easier to reuse successful designs and architectures. A design pattern addressing the versioning problems of multi-generational software systems is presented. The problem of coordinated versioning, inheritance, and evolution are considered. A model is proposed for evaluating the pattern. Preliminary results suggest high scalability and effectiveness.

Key words: generatrix; design pattern; software reliability; generational component

1 概 述

文中所描述的设计模式,用于处理大型和小型软件系统设计中产生新一代或是资源释放所遇到的难以维护的相关问题。著名的 GoF 模式首次提出了母线结构的概念^[1]。“母线”的概念最初是出现在几何中,是元素可以生成自己的后续版本或对象。

初步研究如下。这份简短的研究涉及软件工程领域所熟知的应用情况。并提出一种度量标准,用来减少该模式在实际应用中失败的机会,并提升其可靠性。母线模式期望能够给相关的软件从业者和理论学家们提供一种提高系统软件可靠性和可维护性的方向。

2 设计模式

2.1 意 图

一个应用程序或是系统的设计往往由很多的元素组成,而这些组成元素在很多情况下是需要修改和更新的。这使得不同版本和多代的软件设计变得难以管理和更新。在整个软件设计的生命周期中采用一种设计模式是必要的^[2]。这种设计模式应该使产生连续几代的版本系统成为可能。该设计模式应该提供一些方法确保遵守多代设计原则,包括以下:

- (1) 不需要编写重复代码来产生新对象。
- (2) 将同属于某一个特定代的所有组件的构造方

收稿日期:2012-10-10

修回日期:2013-01-18

网络出版时间:2013-04-08

基金项目:江苏高校优势学科建设工程资助项目(yx002001)

作者简介:赵冉(1990-),男,硕士研究生,研究方向为智能计算技术与应用;张代远,教授,博士,研究生导师,研究方向为人工智能计算理论、方法与应用,计算机体系结构,计算机在通信中的应用。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20130408.1607.034.html>

法进行封装。

(3) 外部客户通常能尽可能方便地获得最新的功能,而无需显示指定特定的代。

(4) 能够方便地获取特定的一代的规则和功能。

(5) 并发联机或是对象的构造应该是可维护的^[3],在更新代操作时不会导致元素的交叉污染^[4]。

(6) 不要让设计模式局限于某种特定的编程语言、数据、网络、框架、协议或是应用程序的类型。

(7) 尽管类继承机制可能适用于小范围的设计,它并不能够很好地应用于大量并发版本生产控制,例如多种企业规则相互关联的情形^[5]。

开发人员会发现有很多类似的准则需要遵守,但这些准则本身并不作为设计模式的一部分。

2.2 动机

随着企业软件系统不断的发展,数据模型和功能可能建立在多个版本系统的基础上。配置管理层总是在努力确保不同代的版本具有可维护性,可测试性和便于其他的管理。软件测试人员力求能够提出一种标准,以便让所有的上述工作在实际应用中成为可能^[6]。几乎所有的工作都依赖于人们使用工具来作出相关的判断和决定。尽管有版本控制和质量确保的相关维护工具,但软件系统变得难以使用。

软件行业总是强烈需求一些能够解决设计模式方面问题的方案。这种需求总是归结为到底什么版本系统是最优的。某些可行的产品提供的功能非常强大,但可能只是针对解决特定类型的应用例如 SOA^[7]。另外一些则是针对某些特殊语言支持的配置管理系统。

在所有的情况下,多代意识来源于管理系统或是框架的最终目的,而不是作为一个软件设计过程中的自然特性出现的。

母线模式在传统经典设计模式的基础上提出了一种将域元素的多代性质融入到软件体系结构的具体设计中的初步设想。

2.3 适用性

设计模式应该适用于任何一种系统,包括 SOA,单独的桌面系统,实时控制系统,金融或是工程系统等。在实际设计过程中,当开始偏离模式时,设计模式应有助于开发人员进行往返设计。尽管设计模式是用类似于传统 UML 格式绘图来描述^[8],但它应该独立于特定的编程语言,框架,软件环境等。

下面,将开始研究母线模式的特点。某一组件可能有一个来自于上一代版本系统中的祖先,它可能有单个后代或没有显示的派生出下一代。这种关系的具体实现方法并没有在母线模式中给出。例如,可以通过职责链模式或是拦截过滤来实现。

某一特定的组件可能有两个或是两个以上的后

代,也可能是在没有任何祖先的情况下作为新一代而产生。这种关系使得该设计模式变得更加有趣,但它并不会破坏母线模式。

2.4 结构

这一部分将介绍母线模式的结构成分,图 1 通过典型方式描述了该设计模式的结构:

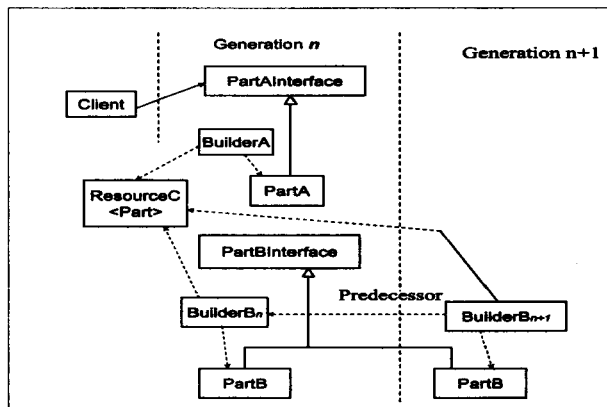


图 1 母线结构类图

(1) Client: 客户是一个具体的系统或是组件,用来进行具体的实现操作,具体表现为提交某种请求,消息,或是方法调用等等。

(2) The two generations: 两代组件(纵向虚线分开)表示这种类似的关系可能会延续到很多代。

(3) The two lineages of Parts: 这两部分出现在每一代中,有类似关系的 Part 数量可能是任意多。在给定的一代中,一个 Part 可能有也可能没有直接的祖先。可能会有任意多此类的模块存在。一个“血统”包括一个 Part 模块以及该模块的所有祖先。

(4) Part: 一个 Part 是至少包含一个接口的类或是模块。接口用于完成具体的应用需求处理,其内部结构并没有限制。

(5) Builder: 在该设计模式中,Builder 要比生成器模式更为一般化。它表示一种对象创建模式,并可能包含原型模式,装饰者模式,部分构造函数,工厂模式,适配器模式等等。可以通过桥接模式,代理模式或是其他的设计模式来实现。

(6) Predecessor: 祖先构件概念的引入帮助说明了职责链模式和过滤器链模式。一个重要的原则是组件不能引用它所派生对象,否则将会违反一代只能知道本身和它的父辈原则。

(7) Resource: 它本身就是一个 Part。它像其他的部件一样可能因不同代而异,具体是实现方法也可能需要重新设计。

2.5 参与者

应用母线模式构造的系统的主要参与者有:

客户端可能有某种“代亲和力”,即对特定的一代兼容性有兴趣。或者可能有想要获得最新信息的意

图。

生产线的子系统在有代亲和力的情况下都有可能被调用。当一个子系统受到代更新的影响时,就可以应用母线模式。

3 初步研究

3.1 可维护性问题的研究

下面列出应用设计模式中所遇到的典型问题,并讨论了初步的解决方案:

(1) 整个模块的复制和修改:当需要实现新的数值和方法时,开发人员可能会拷贝整个模块而不仅仅改变相关的属性和方法。在每一代,大量非常类似的代码会成为代码库的一部分。相应的结果是,出现很多版本的代码,并且无法决定目前正在使用的版本。

(2) 新一代直接替换旧一代:尽管一个或多个旧版本需要维持其目前的形式,开发人员试图有条件地改变逻辑块让模块做些不同操作。这会使代码变得难以控制。

(3) 改变数据库以满足新的需求:考虑在一种具有多代的特定模式下,数据库管理员(DBA)将主要的精力用于为新数据提供支持,而开发者则需要在老地方字段定义业务逻辑引用。

(4) 逻辑无序扩张:决策逻辑的增长趋向于出现在多个地方,而不是比较集中化。这在某种程度上降低了设计的可理解性,使得其他人难以维护和扩展。

(5) 修正了一处,破坏了另一处:为了适应新的需求,一个模块在更新的时候可能会被损坏,并且软件测试可能无法发现出现该问题。

第一个问题的出现是由于不好的设计和可靠性差所导致的,这与可靠性度量标准相一致。为了适应新的需求或是去除已经不再需要的特性所做的操作将会导致失败几率的增长。但很显然,如果只是为了编码的便利而去复用以前版本的代码,失败的几率将会成倍增加。

第二个问题会导致可靠性降低,当新的模块直接替换本来是作为祖先的模块时将会产生相同的效果。这不仅仅会增加新设计中失败的几率,还可能会导致上一代模块功能受损并无法检测。

拷贝整个模块和替换旧的模块在一定程度上不兼容。首先,试图为新一代保持干净的基线,这将会使维护的负担成倍增加。另一方面,如果不做任何拷贝,只是对代码库进行简单的修改,将会冒代码库被污染的风险。

母线设计模式试图解决这种明显的冲突。拥有“血缘”的 Part 概念的提出可以使得设计得到优化,在特定的一代设计成员关系时,没有应用到新功能的部

分将不再被拷贝。该部件可能需要相当精细的设计。

粒度往往决定了整个大的模块是否需要被复制。对于粒度问题,母线模式允许分支^[9]。图 2 中显示派生出两个 Builder 组件以满足新一代的新需求。PartB 可以由两种路径的任一种来构造,但其具体的设计是不变的。

这种分歧使得祖先对象一直停留在原地。也许未来的某些时刻,不再需要时可以将之直接从系统中移除。正确的部件构建者的加载不应该使用特设的编码策略,例如:给名称附加一个字母或数字等。

这一方案解决了目前讨论的两个候选问题。旧的不再是被简单的覆盖,拷贝操作可能发生在构造新对象时,但也只局限于必须修改的部分。

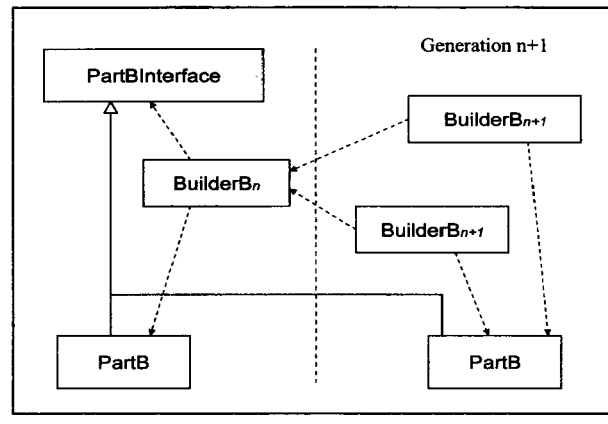


图 2 代间提前分支

3.2 解决方案可靠性研究

母线模式提出了一种在一开始就创建一个高度可维护性设计的前景。或者,它可以用来辅助设计,让其变得更容易维护。这要求开发人员具有丰富的经验来确保有效性。

在具体设计的过程中,相对于有效产出,失败机会显著增加。通过前面问题的论述可见一斑。因此需要一种简单的度量标准^[10],来衡量正在开发的系统的可维护性和可靠性^[11]。

母线模式在可靠性方面的实施措施并不是用来衡量该领域最终产品的性能。虽然这是软件可靠性通常的定义。文中给出了一种新的度量标准,用来衡量软件设计和维护的可靠性。它可以测试设计的特性,并可以用来预测系统开发、测试、维护状态的优劣。这种度量标准侧重于可能派遣错误的实现的数量,而不考虑具体的实现细节。

将通过检验代码库和设计发现的有效派遣操作数量定义为 v ,并将失败机会的数量定义为 f ,将理想的可靠性度量标准定义为:

$$R = v/f \tag{1}$$

这个简单的度量准则,能满足相关文献中的所有可靠性指标标准^[12]。

预测有效性:衡量标准(1)只侧重于设计符合规范的好坏程度。

前提有效性:公式(1)的前提是知道 v 和 f 的值。软件测试可以被用来确定 v 的值,代码扫描将会给出 f 的值。

适用性:该设计模式不依赖于某种编程语言或是系统类型。

简单性:它很容易为人们所理解。

可测试性:这种度量标准是基于可测试的基础上提出的,用来评估的元素是可以计数的。

4 结束语

文中提出了一种设计模式,以解决跨应用程序中的并发代设计和软件维护问题。它适用于组件会按照代步骤进行演化的系统设计,系统组件需要同时支持多代和有代意识,以便客户可以选择具有代亲和力的选项。

初步的研究表明,该设计模式提供了一些用于解决维护困难的初步方案。今后的工作将侧重于该设计模式在实际应用中的相关经验积累。

参考文献:

- [1] Gamma E, Helm R, Johnson R, et al. Design patterns: elements of reusable object-oriented software [M]. [s. l.]: Addison-Wesley Professional, 1995.
- [2] Ramanathan J, Sarkar S. Providing customized assistance for software lifecycle approaches [J]. IEEE Transactions on Software Engineering, 1988, 14(6): 749-757.
- [3] Pomeranz I, Reddy S M. Concurrent online testing of identical circuits using nonidentical input vectors [J]. IEEE Transactions on Dependable and Secure Computing, 2005, 2(3): 190-200.
- [4] Zhao Yang, Chakrabarty K. Cross-contamination Avoidance for Droplet Routing in Digital Microfluidic Biochips [J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 2012, 31(6): 817-830.
- [5] Armstrong J M, Mitchell R J. Uses and abuses of inheritance [J]. Software Engineering Journal, 1994, 9(1): 19-26.
- [6] Sedlet J. Quality Assurance and Control Considerations in Environmental Measurements and Monitoring [J]. IEEE Transactions on Nuclear Science, 1982, 29(3): 1233-1236.
- [7] Erl T. SOA design patterns [M]. [s. l.]: Prentice Hall, 2009.
- [8] Dzidek W J, Arisholm E, Briand L C. A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance [J]. IEEE Transactions on Software Engineering, 2008, 34(3): 407-432.
- [9] Meng Yue, Schlueter R. Bifurcation subsystem identification [C]//Proc. of Power Engineering Society Summer Meeting. [s. l.]: [s. n.], 2002: 1599-1604.
- [10] Itzfeldt W D. Quality metrics for software management and engineering [C]//Proc. of Managing Complexity in Software Engineering. [s. l.]: [s. n.], 2005: 127-152.
- [11] IBurton R M, Howard G T. Optimal Design for System Reliability and Maintainability [J]. IEEE Transactions on Reliability, 1971, R-20(2): 56-60.
- [12] IKan S H. Metrics and models in software quality engineering [M]. [s. l.]: Addison-Wesley Professional, 2002.
- [13] Krecklau L, Kobbelt L. Procedural Modeling of Interconnected Structures [J]. Computer Graphics Forum, 2011, 30(2): 335-344.
- [14] Smelik R, Tutenel T, de Kraker K J, et al. Integrating procedural generation and manual editing of virtual worlds [C]//Proceedings of the 2010 Workshop on Procedural Content Generation in Games. New York: ACM, 2010.
- [15] Hildebrandt K, Schulz C, von Tycowicz C, et al. Interactive spacetime control of deformable objects [J]. ACM Transactions on Graphics, 2012, 31(4): 71-71.
- [16] Bazargan K, Falquet G. Specifying the Representation of Non-geometric Information in 3D Virtual Environments [C]//Proceedings of the 13th International Conference on Human-Computer Interaction, Part II. Berlin: Springer Science + Business Media, 2009: 773-782.
- [17] 徐 赓, 孙济洲, 于 策, 等. 层级式可视化并行程序建模系统研究 [J]. 计算机工程, 2011, 37(8): 1-3.
- [18] 刘 伟, 刘芳兵, 李 静. 面向对象的交互式在线图形化仿真机建模系统开发 [J]. 热力发电, 2010, 39(8): 82-87.

(上接第65页)


manics and procedural generation: key enabling factors for declarative modeling of virtual worlds [C]//Proceedings of the FOCUS K3D Conference on Semantic 3D Media and Content. France: Inria Sophia Antipolis - Méditerranée, 2010.

- [8] Smelik R M, Tutenel T, de Kraker K J, et al. Semantic 3D media and content: a declarative approach to procedural modeling of virtual worlds [J]. Computers and Graphics, 2011, 35(2): 352-363.
- [9] 龙 勇, 袁 静, 康凤举, 等. 可视化仿真中三维建模策略研究 [J]. 系统仿真学报, 2011, 23(12): 2682-2687.
- [10] 杨 斌, 齐玉东, 孟凡磊. 概念建模研究综述 [J]. 计算机与现代化, 2012(1): 44-48.
- [11] 赵新灿. 虚拟环境概念模型建模方法研究 [J]. 计算机应用研究, 2010, 27(3): 995-998.
- [12] Benes B, Massih M A, Jarvis P, et al. Urban ecosystem design [C]//Proc of Symposium on Interactive 3D Graphics and Games. New York: ACM, 2011: 167-174.
- [13] Krecklau L, Kobbelt L. Procedural Modeling of Interconnected

母线设计模式的分析与研究

作者：[赵冉](#)，[张代远](#)，[ZHAO Ran](#)，[ZHANG Dai-yuan](#)

作者单位：[赵冉, ZHAO Ran\(南京邮电大学计算机学院, 江苏南京, 210003\)](#)，[张代远, ZHANG Dai-yuan\(南京邮电大学计算机学院, 江苏南京210003;江苏省无线传感网高技术研究重点实验室, 江苏南京210003;南京邮电大学计算机技术研究所, 江苏南京210003\)](#)

刊名：[计算机技术与发展](#)

英文刊名：[Computer Technology and Development](#)

年，卷(期)：2013, 23(7)

本文链接：http://d.g.wanfangdata.com.cn/Periodical_wjtz201307016.aspx