ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.10, October 2007, pp.2369−2379
DOI: 10.1360/jos182369

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

模型转换中特性保持的描述与验证*

刘 辉[1,2], 麻志毅[1,2+], 邵维忠[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

Description and Proof of Property Preservation of Model Transformations

LIU Hui[1,2], MA Zhi-Yi[1,2+], SHAO Wei-Zhong[1,2]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

+ Corresponding author: Phn: +86-10-62751790, E-mail: mzy@sei.pku.edu.cn, http://www.sei.pku.edu.cn

Liu H, Ma ZY, Shao WZ. Description and proof of property preservation of model transformations. *Journal of Software*, 2007,18(10):2369−2379. http://www.jos.org.cn/1000-9825/18/2369.htm

Abstract: Model transformations are heavily used in model evolution, refinement and refactorings. Model transformations are carried out against certain constraints to preserve certain properties of the models. During model evolution, model transformations should preserve system interfaces; during model refactoring, model transformations should preserve system behaviors. In order to prove that a software transformation satisfies transformation constraints, constraints should be formalized first. And in order to automate the proof, the process of the proof should be universal to be supported by algorithms. This paper proposes an approach for formalizing transformation constraints with graph productions. With the formalized constraints and software transformation rules, an algorithm is also proposed based on critical pair analyzing technologies to automatically prove whether a transformation rule satisfies a transformation constraint or not. The proposed approach is validated with a motivating example used throughout the paper.

Key words: model transformation; property preservation; behavior-preservation; refactoring; evolution; MDA (model driven architecture)

摘 要: 模型转换主要用于模型的演化、求精以及重构.模型转换需要遵循一定的约束规则以保持模型的某些特性.模型演化通常要求保持已有的接口;模型重构则必须保证重构前后的软件具有相同的外部行为特性.为了严格证明某个模型转换规则是否满足这些约束,特性保持约束必须形式化地加以描述.为了实现证明过程的自动化,需要总结通用的证明过程并给出实现算法.提出了一种基于图转换的特性保持约束描述机制,将模型演化与重构中的转换规则以及特性保持约束都描述为图转换规则.借助图转换的冲突检测机制,给出了严格证明转换规则是否满足特性

* Supported by the National Natural Science Foundation of China under Grant Nos.60473064, 60773152 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z127 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2005CB321805 (国家重点基础研究发展计划(973)); the Key Technologies R&D Program of China under Grant No.2003BA904B02 (国家科技攻关计划)

Received 2006-09-07; Accepted 2007-02-08

保持约束的算法.

关键词: 模型转换;特性保持;行为保持;重构;演化;MDA(model driven architecture)
中图法分类号: TP311 文献标识码: A

模型转换是模型驱动体系结构(model driven architecture,简称 MDA)[1]的核心,也是模型演化与重构的关键技术之一.随着运行环境以及需求的不断变化,软件(模型)也需要不停地演化以满足新的需求.模型重构是指在不改变软件外部行为特性的前提下,通过重新组织模型元素以提高模型的质量.模型重构前后的两个版本处于同一抽象级别并实现相同的功能,但重构后的版本具有更好的可扩展性、可维护性和可读性.模型重构可以单独实施以提高软件的质量,也可以作为模型演化的第一步.

模型的演化与重构需要遵循一定的约束规则,以保持模型的某些特性(property).无论是模型演化还是模型重构,都不是重新构造一个全新的模型,所以,模型转换必须保持模型的某些特性.模型重构要求重构前后的两个版本具有相同的外部行为特性.模型演化虽然没有如此严格的限制,但为了不影响与旧版软件交互的其他软件,模型演化通常要求演化后的模型必须提供原模型所提供的所有接口.对于某些特殊的应用系统,可能还需要满足其他一些特性保持约束.比如对实时系统的重构,通常要求重构之后的系统响应时间和空间需求等性能指标不能比原系统差(性能保持).以上列举的约束都要求模型转换必须保持模型的某个特性,比如外部行为特性、接口等等.这种类型的转换约束一般称为特性保持(property preservation)约束[2,3],主要用于保证模型转换不会破坏模型的某些特性.在模型重构和模型演化中,大部分模型转换约束都属于特性保持约束.这也是本文所关注的重点.

如何定义模型转换中的特性保持约束,依然是一个尚未完全解决的难题[2].以模型重构为例,它要求重构前后的两个版本具有相同的行为特性.但是到目前为止,软件行为特性依然没有严格的、公认的定义,而且事实证明,要保证软件行为特性的完全等价几乎是不可能的[3].所以,比较可行的策略是为特定应用中的模型转换定义一组较为宽松的约束[3].这就意味着,不同的应用会有不同的特性保持约束.

如何描述特性保持约束,以及如何证明某个转换规则是否满足某个(或某些)特性保持约束,也是有待深入研究的问题.目前的证明方法主要有两类:第一类是通过测试来证明.Opdyke[4]在其博士论文中提出,行为保持就是要求对于相同的输入,重构前后的软件应该得到相同的输出.对于这样的定义,理论上可以使用回归测试来验证某个转换是否保持软件的行为特性.但在通常情况下,我们并不会运行完整的测试集,因为这样会消耗大量的资源.如果运行不完整的测试集,这样的验证是不严格也不充分的;第二类验证方式是手工的非形式的证明.比如,提出重构规则的研究开发人员通常会以自然语言的形式解释为什么他所提出的重构规则不会改变软件的行为特性[4,5].这两类验证方式都不是形式的严格的证明,所以带有不确定性.要实现形式证明,就必须将特性保持约束形式化.但到目前为止,特性保持约束依然主要靠自然语言描述.

本文提出了一种形式地描述特性保持约束的机制,同时给出了严格证明某个转换规则是否满足特性保持约束的方法.

本文第 1 节简要总结特性保持方面的研究,并指出现有研究的不足.第 2 节给出一个设计模型的实例,这个实例将贯穿整篇文章用于解释和验证相应的概念与方法.第 3 节给出模型转换的形式化描述机制.第 4 节给出描述特性保持约束的机制.第 5 节给出验证某个转换规则是否满足特性保持约束的方法.第 6 节讨论本文所提出方法的优缺点以及该方法所无法处理的某些特殊情况.第 7 节总结本文的工作,并提出进一步研究的方向和可能的研究方法.

1 相关工作

本节总结软件的演化与重构中的特性保持方面的研究,同时通过总结现有的特性保持约束为本文提出的描述和自动验证机制提供分析案例.

对于形式语言来说,特性保持相对比较直观.使用形式语言描述的软件,其语义(或特性)也可以用形式化的

形式给出.所以,可以用定理证明的方式证明某个转换是否保持了软件的语义或特性.Proietti 和 Pettorossi[6]针对逻辑编程语言 Prolog 给出了证明某个重构是否保持程序语义的证明方法.Gheyi 等人[7,8]则针对形式建模语言 Alloy 给出了证明重构是否保持软件语义的证明方法.对于形式语言来说,关键是如何确定模型转换中需要保持哪些特性,并给出这些特性的形式描述.

对于非形式化的复杂编程(或建模)语言来说,特性保持的证明相当困难[3].对于软件重构所关注的行为特性,Mens[3]认为,要证明两个软件的行为特性完全等价几乎是不可能的.所以,人们总是针对特定应用场景给出一系列需要保持的特性,并要求该应用场景内的所有模型转换都必须保持这个系列内的所有特性. Bergstein[9]认为,对面向对象的类图来说,模型重构需要保持的特性就是对象(即对象保持约束):类图的演化和重构不能改变类图所定义的对象集合.Bergstein 所讨论的类图非常简单,只包含两种节点类型:Construction(构造,相当于 UML 的类)和 Alternation(替代,相当于 UML 的抽象类).所以,对象保持就是要保证转换前后的类图具有相同的非抽象类,而且每个非抽象类在转换前后拥有(直接拥有或者通过继承拥有)相同的属性.Bergstein[9]给出了证明两个类图是否对象等价的公式,但没有给出静态地(实施转换之前)证明某个转换规则是否满足类图的对象保持约束的方法.Bergstein 提供了一系列的原子转换规则,这些转换规则是最小而且完备的(任意转换规则都可以通过这些原子规则组合得到).但是,Bergstein 并没有严格地证明这些原子规则是否满足对象保持约束. Opdyke[4]认为,行为保持就是要求对于相同的输入,重构前后的软件应该得到相同的输出.但是,这种定义难以用于证明两个软件是否等价,更无法静态地证明某个重构规则是否会改变软件的行为特性.Straeten 等人[2]认为,为子类和父类之间的行为继承约束(在需要父类对象出现的任何地方都可以使用子类对象代替)也同样适用于模型重构前后的两个版本之间.某个类 C 经过重构之后变成类 D,如果类 D 和类 C 满足行为继承约束,则可以认为该重构保持了软件的行为特性.Straeten 等人[2]给出了判断两个程序是否满足行为继承关系的方法,但没有给出静态地证明某个重构规则是否遵循行为继承约束的方法.Mens 等人[10]针对面向对象编程语言给出了 3 个特性保持约束:调用保持、访问保持和更新保持.调用保持是指某种方法 *Ma* 如果调用了另一种方法 *Mb*,那么,重构之后 *Ma* 也必须调用 *Mb*.如果在重构前某种方法 M 访问(更新)了变量 V,而且重构之后方法 M 也访问(更新)了变量 V,那么就可以说这个重构是访问(更新)保持的.Mens 等人[10]以图表达式(graph expression)的形式对这 3 个约束进行形式化,并借助这些形式化的描述证明了几个重构确实满足这 3 个重构约束.但是,他们没有给出通用的证明过程或算法,所以,证明的过程依然不能由计算机来完成.

综上所述,对于非形式化的复杂编程(或建模)语言来说,人们还无法为其定义完整的语义模型.所以只好退而求其次,定义一系列更为宽松的语义或行为约束:首先,现有的形式化地描述特性保持约束的机制都是为描述特定的特性保持约束而提出来的,这些描述机制还不够通用.此外,现有的验证方法主要用于证明两个软件(模型)之间是否满足特性保持约束,而无法静态地(在执行转换之前)严格证明(而不是解释)某个转换规则是否满足某个或某些特性保持约束.静态证明的好处之一是可以一次证明多次使用:对一个转换规则只需要证明一次,而不是每次转换之后都需要进行验证;其次,动态证明的复杂度(执行转换之后再证明转换后的软件是否具有转换前的某些特性)则往往与软件的规模成正比.软件越大,证明的复杂性就越高.此外,如果动态证明的结果是转换没有保持软件的某些特性,则还需要执行恢复操作.对于许多转换(重构)规则,恢复操作的定义和执行是相当困难的.而静态证明很好地避免了这些问题.虽然某些形式化的约束[10]可以用于静态地证明某个转换规则是否满足这些约束,但是证明的过程过于复杂,未能给出证明的一般过程或者算法.

针对以上不足,本文提出了一种形式化地描述特性保持约束的机制.该描述机制足够通用,可以描述现有研究中提出的大部分特性保持约束.同时,基于本文提出的特性保持约束的描述机制,本文还给出了严格证明某个转换规则是否满足某个或某些特性保持约束的方法.首先,这是一个静态证明过程(在转换之前证明一次即可),所以避免了动态证明的局限性;其次,本文还给出了形式证明的算法描述.借助图转换理论,证明过程可以由计算机自动完成.

2 一个例子

为了清楚地阐述本文的思想,我们以一个例子贯穿全文以解释相关概念.这个例子也作为一个案例用于验证本文提出的特性保持约束的描述与验证机制.

图 1 是简化的多媒体手机设计模型.多媒体手机(Multimedia mobile)同时具有一般手机(mobile)和 MP3 播放器(MP3Player)的特征.所以,类 MultimediaMobile 同时继承了类 Mobile 和 MP3Player.作为多媒体手机的新功能,发送短信(SendMessage)作为类 MultimediaMobile 的公开操作出现.MP3 播放器的操作 Rm2MP3 可以将 RM 格式的音乐文件转换为 MP3 格式以供 MP3 播放器播放.因为格式转换功能是专供播放器自己使用的(对用户来说是不可见的),所以,操作 Rm2MP3 设计为类 MP3Player 的私有操作.

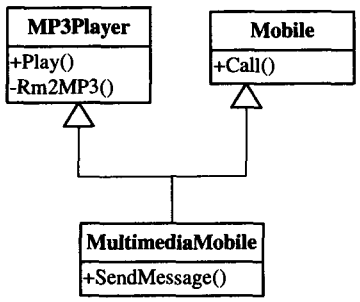

Fig.1 Class diagram of multimedia mobile system
图 1 多媒体手机设计图

为了形式化地描述重构(转换)规则以及特性保持约束,必须首先把多媒体手机的设计类图转换为某种形式化的描述.UML 模型本身就是一个图形表示,所以,使用图(graph)来表示 UML 模型是很自然的选择[3,11].对于源代码,也可以使用图的形式来描述[10].图具有一定的数学理论基础,可用于数学推理和定理证明.图转换(graph transformation)已经形成了一套比较完整的数学理论[12],可用于证明图转换的某些性质.如果使用图来描述 UML 模型,以图转换来描述模型转换规则,那么就有可能借助图转换理论来证明模型转换规则的某些特性.将图 1 所示的多媒体手机设计图用图形式描述,得到如图 2 所示的图.其中,顶点标签包含两部分,前一部分是顶点的类型,后一部分是顶点的名称,类型和名称之间以冒号(:)分隔.图中边的标签是边的类型名,比如 Generalization(继承)、Operation(操作)和 Visibility(可见性)等等.

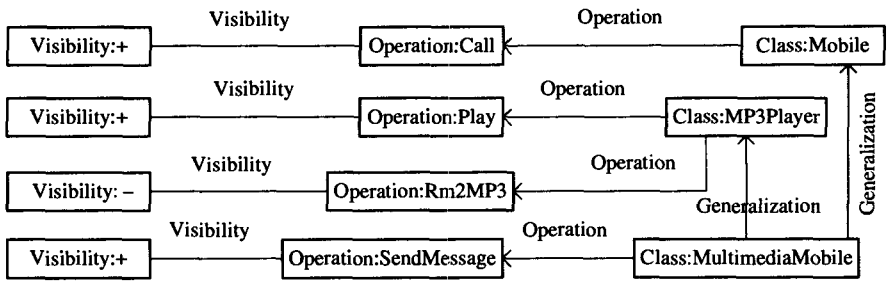

Fig.2 Graph description of the multimedia mobile class diagram
图 2 多媒体手机设计图的图描述

3 转换规则及其表示

如第 2 节所述,因为图及图转换具有一定的数学理论基础,而且以图的形式描述模型(包括源代码)非常自然、直观,所以,许多模型转换技术都采用了图转换的方式[10,11,13],将模型转换规则描述为图转换规则(production).图转换规则可以形式地表示为 $P: L \xrightarrow{f} R$,其中,L 和 R 分别称为左图(也称为左模式)和右图(右模式),图映射 f 则用于指明 L 和 R 之间的关系[12].当且仅当 L 的某个顶点 v 在图转换中没有被删除也没有被修改时,存在顶点 $v' \in R$ 满足:$(v,v') \in f$.通常,可以通过给 L 和 R 中的顶点编号来描述图映射 f:如果 $v \in L, v' \in R$ 而且 v 和 v' 具有相同的编号,那么,$(v,v') \in f$.应用图转换规则 P 对图 G 进行一次转换,可以形式化地表示为 $t: G \xrightarrow{p,m} H$,其中,图映射 m 是一个匹配 $m: L \to G$.在图转换 t 中,模式 L 在 G 中的一个实例 $m(L)$ 被 R 的实例 $m^*(R)$ 所取代——$m(L-R)$ 被删除,同时添加 $m^*(R-L)$.其中,m^* 是 m 的同匹配(co-match),m^* 和 m 一起构成如图 3 所示的推出(pushout)[12].图转换的描写是模型转换的难点之一.图转换规则必须保持软件的特性,这也是本文的研究重点.

本节对如图 1 所示的例子进行演化,给出模型演化规则并将演化规则描述为图转换规则.假设选定的编程语言不支持多继承,那么,首先必须想办法去除多继承.邵维忠和杨芙清[14]提供了几种去除多继承的方法,其中的一种方法就是保留一个继承关系,将其他继承关系用聚合关系取代.如图 4 所示,多媒体手机对手机的继承关系得到保留,同时,多媒体手机对 MP3 播放器的继承关系由聚合关系取代.

第 2 个变化是公开用于转换音乐格式的操作 Rm2MP3.用户希望可以借助 MP3 播放器将 RM 格式的音乐文件转换为 MP3 格式,然后用 MP3 播放器或者个人计算机播放 MP3 音乐.此时,格式转换功能对用户和外接设备来说是可见的,所以,在设计中把操作 Rm2MP3 的可见性改为公开的(public).

第 3 个改变是把多媒体手机(MultimediaMobile)的短信功能(SendMessage)上移至其父类 Mobile.这么做的主要原因是,一般手机的功能逐渐增强,使得普通手机也具备了多媒体手机的某些功能.

经过以上 3 个方面的修改之后,得到的多媒体手机设计图如图 4 所示.

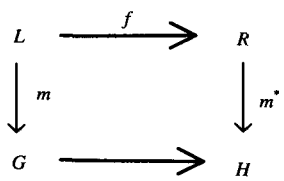

Fig.3 Pushout

图 3 推出

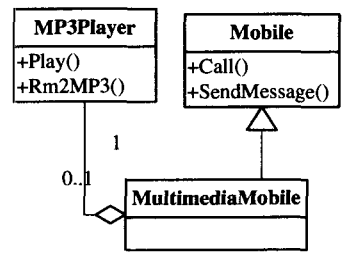

Fig.4 New version of the multimedia mobile class diagram

图 4 演化版多媒体手机设计图

以上 3 个转换的相应转换规则可以采用图转换的方式来描述,得到如图 5~图 7 所示的图转换规则.图 5 以图转换规则的形式描述如何移除多继承:将类 3 对类 1 的继承关系改用聚合关系取代.图 6 是公开一个私有操作的图转换规则.图 7 的转换规则用于将子类的某个操作上移到父类.

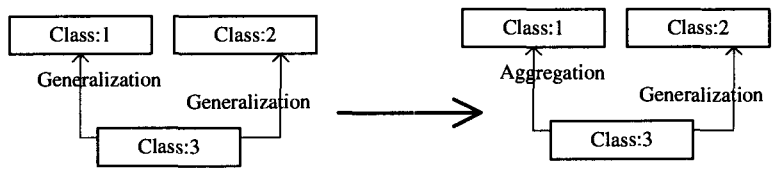

Fig.5 Rule 1: Remove multi-inheritance

图 5 规则 1:消除多继承

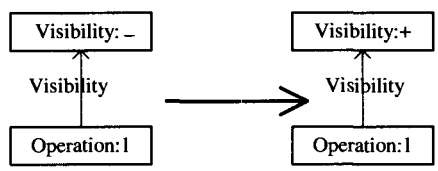

Fig.6 Rule 2: Publish an operations

图 6 规则 2:公开操作

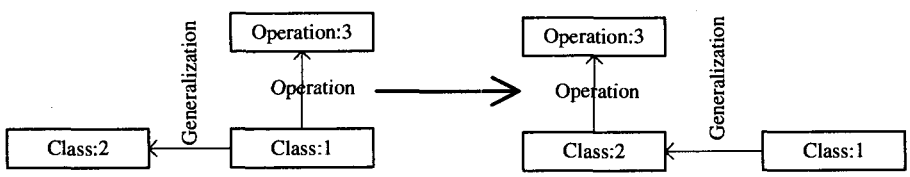

Fig.7 Rule 3: Move up an operation

图 7 规则 3:上移操作

4 特性保持约束及其表示

如前文所述,模型的演化、求精和重构都不是随意进行的无约束的模型转换.模型转换必须满足一定的特性保持约束.因为 UML 模型没有一个严格的行为语义描述模型[2],所以,对于 UML 模型的演化和重构必须根据需要定义一套模型转换的特性保持约束.我们根据相关工作中所讨论的特性保持约束,给出 UML 类图的重构和演化所需要遵守的一些特性保持约束.我们的目标并不是给出完整的约束集,而是通过常见的特性保持约束总结出特性保持约束的一般特征,进而确定应该如何形式化地描述特性保持约束.

Bergstein[9]的对象保持约束也适用于 UML 类图.对象保持约束可以细分为如下两个约束:

(1) 如果类图包含非抽象类(具体类)C,则实施模型演化(重构)之后的类图还必须包含类 C.

(2) 如果非抽象类 C 拥有(直接拥有或者通过继承拥有)特性(property,包括属性和操作)F,则演化(重构)之后类 C 也必须拥有(直接拥有或者通过继承拥有)特性 F.

一个系统(或者模块)通常都需要与其所在的环境进行交互,这种交互往往通过软件接口完成.在设计图上,可以使用 UML 接口(interface)显式地说明软件的接口,也可以通过类、属性和操作的可见性来说明软件的对外接口.为了不影响与系统交互的其他系统(或模块),通常要求软件的版本演化应该保持旧版软件所提供的所有接口.该约束的详细描述如下(假定可见性只有公开的、受保护的和私有的 3 种情况):

(3) 如果类图中某个特性 F 的可见性是公开的,则演化(重构)之后,特性 F 的可见性也必须是公开的.

(4) 如果类图中某个特性 F 的可见性是受保护的(protected),则演化(重构)之后,特性 F 的可见性也必须是公开的或者是受保护的.

从描述的模式上看,以上 4 个约束都符合这样一个模式:"如果原类图具有...(A)...,则要求演化(重构)之后的类图应该具有...(A')...".如果按照这个模式重新组织以上 4 个约束,可以得到如下等价描述:

(1) 如果原类图包含非抽象类(具体类)C,则实施模型演化(重构)之后的类图也必须包含类 C.

(2) 如果原类图包含非抽象类 C,而且类 C 拥有(直接拥有或者通过继承拥有)特性 F,则演化(重构)之后的类图也必须包含非抽象类 C,而且类 C 必须拥有(直接拥有或者通过继承拥有)特性 F.

(3) 如果原类图包含可见性为公开的特性 F,则演化(重构)之后的类图必须包含特性 F,而且 F 的可见性也必须是公开的.

(4) 如果原类图包含可见性为受保护的特性 F,则演化(重构之)后的类图必须包含特性 F,而且 F 的可见性必须是公开的或者受保护的.

在这个约束描述模式中,A 和 A'代表两个模式.从表述上看,A 和 A'应该具有相同的行为语义(或者在满足其他约束之后,A 和 A'应该具有相同的行为语义).如果 A 和 A'是一样的(许多情况下,A 和 A'确实是一样的),那么,A 和 A'显然具有相同的语义.但是这不是必须的,比如约束规则(4)就具有不同的 A 和 A'.

对于转换规则,特别是用图转换形式描述的转换规则,也符合这个描述模式.如第 3 节所述,一个转换规则可以描述为图转换规则 $P:L\xrightarrow{f}R$,其中,L 和 R 分别是两个模式.在语义上,这个转换可以解读为:如果原系统中包含 L 的一个实例,则转换后的系统必须包含一个 R 的实例[15].

鉴于转换规则的描述模式和特性保持约束的描述模式之间的相似性,可以将转换规则的描述机制用于特性保持约束的描述.特性保持约束"如果原类图具有...(A)...,则要求演化(重构)之后的类图应该具有...(A')..."可以用图转换规则 $p:A\xrightarrow{f}A'$ 表示.将上述 4 个特性保持约束描述为图转换规则得到的结果如图 8~图 11 所示.

Fig.8 Constrain 1: Class preservation

图 8 约束 1:类保持

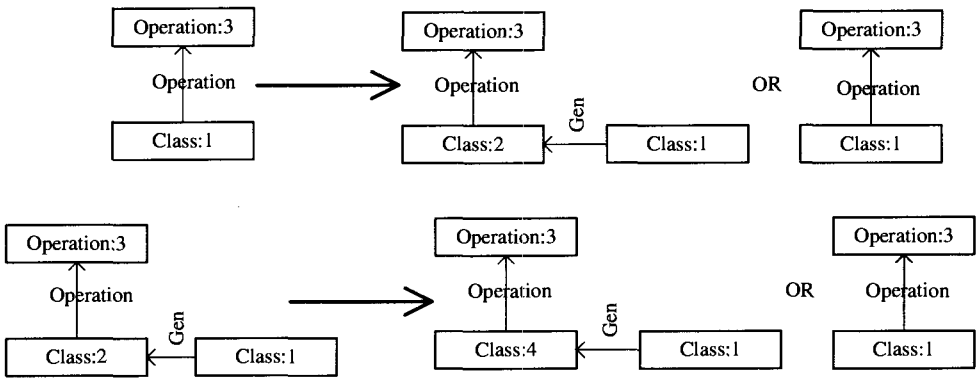

Fig.9 Constrain 2: Property (operation) preservation

图 9 约束 2:特性(操作)保持

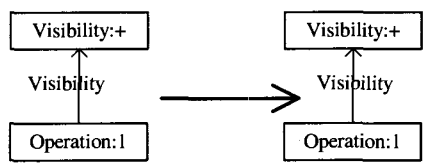

Fig.10 Constrain 3: Visibility (public) preservation

图 10 约束 3:可见性(公开的)保持

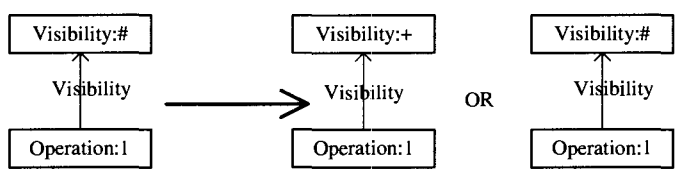

Fig.11 Constrain 4: Visibility (protected) preservation

图 11 约束 4:可见性(受保护的)保持

操作符"OR"表示"或"操作,比如约束 4 中受保护的特性(操作)的可见性可以保持不变或者变成公开的.约束 2 要求类直接或间接拥有的特性(包括属性和操作)不能丢失.间接拥有特性主要是指通过继承而拥有父类的特性,而且这个继承可以是任意多层次的.也就是说,直接拥有这个特性的可以是该类的直接父类或者更高层的祖先.如果用常规表达式来表示任意层次的继承,则应该表示为"Generalization*",其中,"*"表示任意多.但是现有的图转换工具,比如 AGG[16],不支持带"*"号的常规表达式.变通的方法是新增一个关系类型"Gen"用于关联子类和该类的所有父类及祖父.图 12 给出的两个转换规则可以自动为类图添加"Gen"关联.

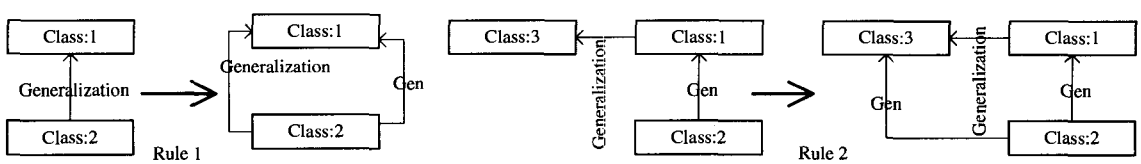

Fig.12 Preprocess rules

图 12 预处理规则

特性保持约束虽然采用了与转换规则相同的描述方式(图转换规则),但是它们依然是有区别的:首先,特性保持约束不能用于改变软件模型.人们只能执行转换规则以改变软件模型,特性保持约束只用于检查转换(或者

转换规则)是否满足特性保持约束;其次,在特性保持约束 $p:A\xrightarrow{f}A'$ 中,$A-A'$部分不一定要被删除,$A'-A$ 部分也不一定是新增的内容.比如在图 9 中的规则 2,顶点"Class:2"可能被删除,可能被保留,甚至可能就是"Class:4";而如果规则 2 描述的是一个模型转换规则,那么顶点"Class:2"肯定会被删除,而"Class:4"则一定会被添加.

5 特性保持约束的证明

本文第 3 节和第 4 节分别把模型转换规则以及特性保持约束描述为图转换规则.图转换规则可以用于推导,借助图转换理论可以自动检测图转换规则之间的冲突和依赖关系[17,18].通过图转换理论也可以推导出某个转换规则是否满足特性保持约束.

转换规则之间的冲突关系可以通过图转换中的关键对(critical pair)进行分析[17].给定转换 $t_1:G\xrightarrow{p_1,m_1}H_1$ 和 $t_2:G\xrightarrow{p_2,m_2}H_2$,如果转换 t_1 的执行会导致转换 t_2 无法执行,或者转换 t_2 的执行会导致转换 t_1 无法执行,那么我们就说转换 t_1 和 t_2 有冲突.假定 $p_1:L_1\xrightarrow{f_1}R_1$, $p_2:L_2\xrightarrow{f_2}R_2$,那么当且仅当 $m_1(L_1-R_1)\cap m_2(L_2)\neq\varnothing$时,$t_1$ 的执行会导致 t_2 无法执行.也就是说,当且仅当 t_1 删除了某些匹配 m_2 所需的元素,t_1 的执行才会导致 t_2 无法执行.我们无法列举所有导致转换规则 p_1 和 p_2 发生冲突的源图(source graph)G 以及匹配 m_1 和 m_2,因为这样的转换可能会有无穷多个.

定义 1(关键对). 关键对是一对有冲突的转换 $t_1:G\xrightarrow{p_1,m_1}H_1$ 和 $t_2:G\xrightarrow{p_2,m_2}H_2$,其中,匹配 m_1 和 m_2 为联合满射(jointly surjective morphisms).

关键对中的源图 G 是导致转换规则 p_1 和 p_2 发生冲突的最小源图,如果某个源图 G'导致规则 p_1 和 p_2 发生冲突,那么一定存在 G'的一个子图 $G\subseteq G'$(保留 $m_1(L_1)$和 $m_2(L_2)$部分,其他都删除),使得规则 p_1 和 p_2 在图 G 上的转换成为一个关键对.也就是说,只要转换规则 p_1 和 p_2 不存在任何关键对,那么,转换规则 p_1 和 p_2 就不可能发生冲突.因为在关键对中,匹配 m_1 和 m_2 是联合满射,所以,对于给定的两个转换规则,可能导致关键对的源图 G 的数目是有限的.可以通过遍历所有这样的源图 G 来检测两个转换规则是否有冲突[18].已有许多图转换工具支持关键对分析,比如 AGG[16,17].

给定转换规则 $r:L_r\xrightarrow{f_r}R_r$ 和转换 $t_r:G\xrightarrow{r,m_r}H$ (如图 13 左部所示),需要检测转换 t_r 是否满足特性保持约束 $c:L_c\xrightarrow{f_c}R_c$.如果不存在匹配 $m_c:L_c\to G$,即在源图 G 中不存在模式 L_c 的实例,则特性保持约束 c 对转换 t_r 无效(也就是说,转换 t_r 并不违反约束 c).如果存在匹配 $m_c:L_c\to G$,但在转换之后 $m_c(L_c)$既没有被删除也没有被修改(即 $m_c(L_c)\cap m_r(L_r-R_r)=\varnothing$),那么,转换 t_r 也没有违反约束 c.转换 t_r 违反约束 c 的唯一可能情况是:存在匹配 $m_c:L_c\to G, m_c(L_c)\cap m_r(L_r-R_r)\neq\varnothing$,而且转换得到的图 H 不存在子图 $H'=m_c^*(R_c)$,使得图 13 所示的左、右两个推出(pushout)成立.这种情况意味着转换 t_r 破坏了 L_c 在图 G 中的实例,却没有构造出相应的 R_c 实例.

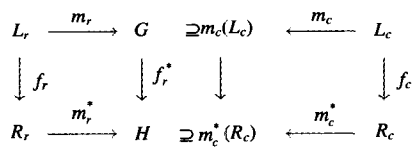

Fig.13 Transformation and property preservation constraint

图 13 转换与特性保持约束

总结上文讨论可以看到,转换 t_1 导致转换 t_2 无法执行的充要条件是 t_1 的执行会删除转换 t_2 所需的元素,即

$$m_1(L_1-R_1)\cap m_2(L_2)=\varnothing \tag{1}$$

而转换 t_r 违反特性保持约束 c 的必要条件是转换 t_r 破坏了约束 c 的左模式 L_c 的匹配,即:

$$m_r(L_r-R_r)\cap m_c(L_c)\neq\varnothing \tag{2}$$

条件(1)和条件(2)不但具有相似的语意,还具有相同的表现形式.如果把约束 c 看成是一个转换规则,把 $t_c:m_c(L_c)\xrightarrow{c,m_c}m_c^*(R_c)$ 看成是转换规则 c 的执行,那么,转换 t_r 违反约束 c 的前提之一就是转换 t_r 的执行必须

导致 t_c 无法执行(转换冲突).所以,检测转换规则是否冲突的手段(关键对技术)也可以用于检测转换规则与转换约束之间的关系.检测算法如下所示:

1:输入:转换规则 $r:L_r \xrightarrow{f_r} R_r$,特性保持约束 $c:L_c \xrightarrow{f_c} R_c$
2:**If** $(L_r-R_r)=\varnothing$
3: Return False //无冲突
4:**End If**
5:**For Each** Critical Pair $\langle t_r:G \xrightarrow{r,m_r} H_1, t_c:G \xrightarrow{c,m_c} H_2 \rangle$
6: **If** $m_r(L_r-R_r)\cap m_c(L_c)=\varnothing$ //L_c 的实例 $m_c(L_c)$不会被破坏
7: Continue
8: **End If**
9: **If** $\exists m_c^*:R_c \longrightarrow H_1 \bullet t_r(m_c(L_c \cap R_c))=m_c^*(L_c \cap R_c)$
10: //L_c 的实例 $m_c(L_c)$由 R_c 的实例 $m_c^*(R_c)$ 取代
11: Continue
12: **End If**
13: Return True //冲突
14:**End For**
15:Return False //无冲突

下面以多媒体手机的例子作为案例进行分析验证.在多媒体手机的例子中,有 3 条转换规则和 4 条特性保持约束.下面依次检测每条转换规则是否满足这 4 条特性保持约束.

(1) 转换规则 1(消除多继承):L_r-R_r={Generalization,Gen}(其中,Gen 是通过图 12 的预处理自动添加的),而特性保持约束规则 1、约束规则 3 和约束规则 4 都没有用到 Generalization 或 Gen 边,所以,对任何关键对均有 $m_r(L_r-R_r)\cap m_c(L_c)\neq\varnothing$成立.所以,算法执行到第 3 行结束并返回 False.也就说,转换规则 1 不可能和特性保持规则 1、约束规则 3 或者约束规则 4 发生冲突.因为约束 2 的第 1 条规则也没有用到 Generalization 或 Gen 边,所以转换规则 1 也不会和约束 2 的第 1 条规则发生冲突.约束 2 的第 2 条规则用到一条 Gen 边,所以,唯一可能导致冲突的源图 G 如图 14 所示,此时,$m_r(L_r-R_r)\cap m_c(L_c)=Gen$,得到的目标图 H 如图 15 所示.虽然存在匹配 $m_c^*:R_c \longrightarrow H$,但是 $t_r(m_c(L_c \cap R_c)) \neq m_c^*(L_c \cap R_c)$,即算法执行到第 9 行判断分支条件为假,并进入第 13 行,判定转换规则 1 不满足约束 2 的第 2 条规则.

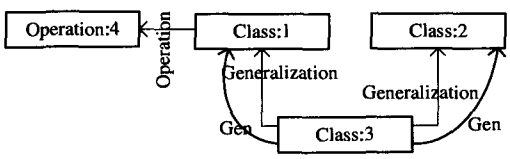

Fig.14 Source diagram G of rule 1
图 14 规则 1 的源图 G

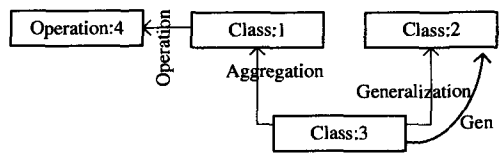

Fig.15 Target diagram H of rule 1
图 15 规则 1 的目标图 H

(2) 转换规则 2(公开操作):L_r-R_r={(Visibility,–)},而特性保持约束规则 1~约束规则 4 都没有用到(Visibility,–)的顶点,所以,对任何关键对均有 $m_r(L_r-R_r)\cap m_c(L_c)=\varnothing$成立.所以,转换规则 1 不会和特性保持规则 1~约束规则 4 发生冲突.

(3) 转换规则 3(上移操作):L_r-R_r={Operation},而特性保持约束规则 1、约束规则 3、约束规则 4 都没有用到 Operation 边,所以对任何关键对均有 $m_r(L_r-R_r)\cap m_c(L_c)=\varnothing$成立.所以,转换规则 1 不会和规则 1、约束规则 3、约束规则 4 发生冲突.用到 Operation 边的约束只有约束 2.首先检测转换规则是否满足约束 2 的第 1 条规则.因为 L_r-R_r={Operation},而约束 2 的第 1 条规则的 L_c 中只有一条边为 Operation 类型,所以会导致 $m_r(L_r-R_r)\cap m_c(L_c)\neq\varnothing$的关键对源图只有如图 16 左图所示的源图 G.源图 G 正好是 L_r,所以转换之后得到的图 $H=R_r$(如图 16 右图所示).因为存在匹配 $m_c^*:R_c \longrightarrow H$,而且 $t_r(m_c(L_c \cap R_c))=m_c^*(L_c \cap R_c)$,所以转换规则 3 和约束 2 的第 1 条约束规则没有发生冲突.同理,可以证

明转换规则 3 也满足约束 2 的第 2 条约束规则.

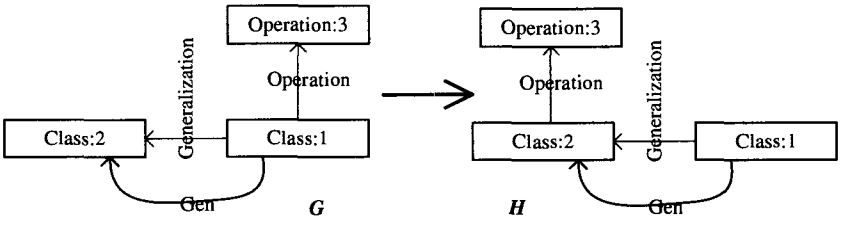

Fig.16 Source diagram G and target diagram H of the critical pair

图 16 关键对的源图 G 和目标图 H

6 讨 论

第 4 节将转换中的特性保持约束描述为图转换规则,而且本文第 5 节给出了验证某个转换规则是否满足特性保持约束的算法.所以对于特定的应用,只要可以给出特性保持约束,并将这些规则描述为图转换规则,那么,计算机就可以自动地证明某个转换规则(以图转换的形式描述)是否满足这些特性保持约束.

虽然图转换规则可以描述模型演化与重构中的大多数特性保持约束,但是还有一些比较特殊的特性保持约束难以描述为图转换规则.从第 4 节的讨论可知,特性保持约束必须满足以下两个条件才可以描述为图转换规则:

(1) 符合描述模式:"如果旧版本模型具有...(*A*)...,则要求演化(重构)之后的版本应该具有...(*A'*)...".

(2) *A* 和 *A'*图形化之后分别是旧版本模型(图形化之后)和新版本模型(图形化之后)的子图.

这就要求特性保持约束中体现该"特性"的实体(*A* 和 *A'*)必须直接出现于软件描述中.对于大多数特性保持约束,其所保持的"特性"都满足这个要求,比如在本文第 1 节所提到的对象保持、行为继承约束、调用保持、访问保持、更新保持以及接口保持等等.但是也有一些约束需要保持某些导出特性,比如软件的时间和空间复杂度(性能保持).本文提出的描述和检测算法暂时还无法处理这类导出特性.

7 结论和进一步的工作

模型的演化和重构都需要遵循一定的约束以保持模型的某些特性,这些约束的描述和证明是当前的研究热点之一.本文采用图转换的形式描述模型转换中的特性保持约束,同时给出了静态地证明某个转换规则是否满足特性保持约束的算法.证明过程可以由计算机自动完成.本文还以多媒体手机的设计类图作为例子,解释和验证了本文提出的描述机制以及证明方法.

进一步的研究将把本文提出的方法应用于 UML 模型重构.首先为 UML 模型(包括类图、交互图、用况图、状态图、构件图等等)重构确定重构中必须遵循的特性保持约束,并确定哪些约束可以描述为图转换规则.约束规则的发现和确立需要对产业中模型重构项目进行深入和广泛的调查.确立约束规则之后,需要对常见的 UML 模型重构规则进行验证,以确定特性保持约束的正确性和合理性.挑选出所有合理的约束规则组成约束集,该集合可用于对新的模型重构规则进行验证,如果验证失败,则应提醒重构规则的制订者重新考虑重构的合理性.最后,我们将把这一功能集成到 UML 建模工具 JBOO[19]中,以辅助设计人员设计模型转换规则.

References:

[1] Miller J, Mukerji J, eds. MDA guide version 1.0. OMG document: omg/2003-05-01. 2003. http://www.omg.org/mda/mda_files/MDA_Guide_Version0.pdf

[2] van Der Straeten R, Jonckers V, Mens T. Supporting model refactorings through behaviour inheritance consistencies. In: Baar T, *et al.*, eds. Proc. of the Unified Modeling Language (UML) 2004. LNCS 3273, Berlin, Heidelberg: Springer-Verlag, 2004. 305-319.

[3] Mens T. A survey of software refactoring. IEEE Trans. on Software Engineering, 2004,30(2):126-139.

[4] Opdyke WF. Refactoring object-oriented frameworks [Ph.D. Thesis]. Urbana-Champaign: University of Illinois, 1992.

[5] Fowler M, Beck K, Brant J, Opdyke W, Roberts D. Refactoring: Improving the Design of Existing Code. California: Addison Wesley Longman, Inc., 1999.

[6] Proietti M, Pettorossi A. Semantics preserving transformation rules for prolog. ACM SIGPLAN Notices, 1991,26(9):274–284.

[7] Gheyi R, Massoni T, Borba P. A rigorous approach for proving model refactorings. In: Redmiles DF, Ellman T, Zisman A, eds. Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2005). Long Beach, 2005. 372–375. http://portal.acm.org/citation.cfm?id=1101973

[8] Gheyi R, Massoni T. Formal refactorings for object models. In: Proc. of the 20th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications. New York: ACM Press, 2005. 208–209. http://portal.acm.org/citation.cfm?id=1094938

[9] Bergstein PL. Object-Preserving class transformation. In: Proc. of the ACM SIGPLAN Conf. on Object-oriented Programming, Systems, Languages, and Applications. New York: ACM Press, 1991. 299–311. http://portal.acm.org/citation.cfm?id=117977&coll=portal&dl=ACM

[10] Mens T, van Eetvelde N, Demeyer S, Janssens D. Formalizing refactorings with graph transformations. Journal of Software Maintenance and Evolution: Research and Practice, 2005,17(4):247–276.

[11] Grunske L, Geiger L, Lawley M. A graphical specification of model transformations with triple graph grammars. In: Hartman A, Kreische D, eds. Proc. of the European Conf. on Model Driven Architecture—Foundations and Applications (ECMDA-FA 2005). LNCS 3748, Berlin, Heidelberg: Springer-Verlag, 2005. 284–298.

[12] Ehrig H, Engels G, Kreowski HJ, Rozenberg G. Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1. Singapore: World Scientific Publishing Co., Inc., 1997.

[13] Czarnecki K, Helsen S. Classification of model transformation approaches. In: Proc. of the OOPSLA 2003 Workshop on Generative Techniques in the Context of Model-Driven Architecture. 2003. http://www.swen.uwaterloo.ca/~kczarnec/ECE750T7/czarnecki_helsen.pdf

[14] Shao WZ, Yang FQ. Object-Oriented System Design. Beijing: Tsinghua University Press, 2003. 39–43 (in Chinese).

[15] Object Management Group. MOF QVT final adopted specification. OMG Adopted Specification ptc/05-11-01. 2005.

[16] 2006. http://tfs.cs.tu-berlin.de/agg/

[17] Taentzer TG, Runge O. Detecting structural refactoring conflicts using critical pair analysis. Electronic Notes in Theoretical Computer Science, 2005,127(3):113–128.

[18] Lambers L, Ehrig H, Orejas F. Efficient detection of conflicts in graph-based model transformation. Electronic Notes in Theoretical Computer Science, 2006,152(2):97–109.

[19] Ma ZY, Zhao JF, Meng XW, Zhang WJ. Research and implementation of jade bird object-oriented software modeling tool. Journal of Software, 2003,14(1):97–102 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/97.htm

附中文参考文献:

[14] 邵维忠,杨芙清.面向对象的系统设计.北京:清华大学出版社,2003.39–43.

[19] 麻志毅,赵俊峰,孟祥文,张文娟.青鸟面向对象软件建模工具的研究与实现.软件学报,2003,14(1):97~102. http://www.jos.org.cn/1000-9825/14/97.htm

刘辉(1978—),男,福建长汀人,博士生,主要研究领域为面向对象建模,软件重构,元建模,MDA 以及形式化软件工程.

邵维忠(1946—),男,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件工程环境,面向对象技术,软件复用,构件技术.

麻志毅(1963—),男,博士,副教授,CCF 高级会员,主要研究领域为软件工程,软件工程支撑环境,面向对象技术,构件技术.