ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, Vol.20, No.8, August 2009, pp.2087−2101
doi: 10.3724/SP.J.1001.2009.03469

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

# 一种基于图转换的模型重构描述语言*

刘 辉[1,2,3+], 麻志毅[2,3], 邵维忠[2,3]

[1](北京理工大学 计算机学院,北京 100081)

[2](北京大学 信息科学技术学院 软件研究所,北京 100871)

[3](高可信软件技术教育部重点实验室,北京 100871)

## Graph Transformation Based Description Language for Model Refactorings

LIU Hui[1,2,3+], MA Zhi-Yi[2,3], SHAO Wei-Zhong[2,3]

[1](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

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

[3](Key Laboratory of High Confidence Software Technologies for the Ministry of Education, Beijing 100871, China)

+ Corresponding author: E-mail: liuhui2005@gmail.com, http://www.sei.pku.edu.cn

**Liu H, Ma ZY, Shao WZ. Graph transformation based description language for model refactorings. *Journal of Software,* 2009,20(8):2087−2101.** http://www.jos.org.cn/1000-9825/3469.htm

**Abstract**: This paper proposes a graph transformation based description language of model refactoring (GraTDeLMoR) to formalize model refactorings. It designs basic elements of the language according to the features of model refactorings, and proposes approaches to describe model refactorings with these basic elements. It also proposes steps of the application of refactorings formalized with the proposed language, and provides corresponding CASE support. The paper discusses the descriptive ability of the description language with some typical examples, and results suggest that the proposed language is expressive in formalizing model refactorings.

**Key words**: refactoring; model; graph transformation; description language

摘 要: 提出了一种基于图转换的模型重构描述语言.针对模型重构的特征,设计了模型重构描述语言的基本元素,并给出了如何通过这些基本元素描述模型重构及重构规则的方法.在此基础上,给出了根据形式化重构规则执行模型重构的具体步骤和策略,并提供了较为完整的模型重构 CASE 支撑工具.通过实例讨论了该模型重构描述语言的描述能力.结果表明,该语言具有较强的描述能力,能够比较简洁地描述复杂的模型重构规则.

关键词: 重构;模型;图转换;描述语言

中图法分类号: TP311 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant No.60773152 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2007AA010301-01, 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 National Key Technology R&D Program of China under Grant No.2006BAH02A02 (国家科技支撑计划); the Key Laboratory of High Confidence Software Technologies for the Ministry of Education of China under Grant No.HCST200802 (高可信软件技术教育部重点实验室开放研究基金)

Received 2008-04-02; Revised 2008-08-07; Accepted 2008-10-07; Published online 2009-04-10

重构是在不改变软件的外部行为特性的前提下,通过重新组织软件内部元素来提高软件质量的活动[1].软件重构作为一种提高软件质量的有效手段,已经成为软件开发与维护的关键活动之一[2−4].软件重构的对象可以是软件开发中的任何制品,包括需求文档、设计模型、源代码,甚至测试案例[5].在软件重构提出的最初 10 年左右(1992 年~2001 年),软件重构主要关注于源代码层[1,5].随着软件建模技术与建模语言[6,7]的日趋成熟以及模型驱动的体系结构(model driven architecture,简称 MDA)[8]的推广和普及,模型逐渐成为软件开发与维护的核心制品.相应地,模型重构也逐渐成为软件重构的一个研究热点.Astels[9]则提出可以利用 UML 模型辅助代码重构.Sunye 等人[10]首次提出模型重构的概念,并给出了 UML 类图和状态图的一些重构规则.Correa 和 Werner[11]进一步将重构引入带 OCL(object constraint language)约束[12]的 UML 模型.模型重构作为提高模型质量的一种有效手段,逐渐为人们所认可和采用.

模型重构的核心是模型重构规则.每条重构规则针对一类设计缺陷,并给出消除此类设计缺陷的方法.无论是手工执行模型重构,还是借助 CASE 工具自动执行模型重构,都必须根据专家定义的模型重构规则进行.但是,重构规则的执行者和重构规则的定义者通常不是同一个人.一般由专家定义重构规则,而执行重构是一般的软件设计人员或 CASE 工具.所以,为了便于重构执行人员准确理解重构规则,就必须将重构规则准确地描述出来.

现阶段,人们依赖自然语言来描述重构规则[1,13].虽然自然语言描述易于书写和阅读,但是自然语言容易产生二义性,进而威胁模型重构的正确性.此外,CASE 工具目前还无法理解自然语言,所以也就无法执行自然语言描述的模型重构规则.因此,人们需要某种形式语言来描述模型重构规则[5,14].

形式语言的使用不但可以避免自然语言的二义性,而且可以有效提高模型重构的效率和可靠性.首先,重构的形式化是实现重构自动或半自动化的基础,而自动或半自动化的软件重构技术以及重构工具的出现将有助于提高重构的执行效率,降低重构的成本[5,10,15,16];其次,重构的形式化有利于提高重构的可靠性.通过理论推导和形式证明等方式可以验证重构的特性,从而提高模型重构的可靠性[14].

重构形式化的关键是要设计一种合适的软件重构描述语言.形式化重构描述语言首先必须要有足够的表达能力,能够比较方便地描述常见的重构规则;其次,形式化重构语言还应该足够简单明了,以减小用户的学习成本以及重构工具的开发成本.

本文针对模型重构的特点提出了一种基于图转换的模型重构描述语言(graph transformation based description language of model refactoring,简称 GraTDeLMoR).首先,该描述语言只针对面向对象软件模型上的模型重构.这不仅有利于简化形式描述语言,而且有利于充分利用模型重构的特征以强化形式描述语言的描述能力.其次,该描述语言是基于经典图转换描述语言的.基于形式的图转换描述语言,可以充分利用现有的图转换理论对模型重构及重构规则的性质进行推导,提高模型重构的可靠性.而且,基于人们熟悉的图转换描述语言也有利于用户的学习和使用,有利于复用现有的图转换工具.

本文第 1 节介绍模型重构描述语言的基本元素.第 2 节介绍如何用模型重构描述语言的基本元素描述模型重构及重构规则.第 3 节通过具体的实例讨论本文提出的模型重构描述语言的描述能力.第 4 节给出相应的 CASE 支撑工具.第 5 节集中讨论在描述语言的表达能力、推导能力、CASE 工具性能与效率之间的权衡问题.第 6 节讨论相关工作.第 7 节总结本文的工作.

## 1 模型重构描述语言的基本元素

每条模型重构规则针对一类设计缺陷,给出用于替换该类设计缺陷的更合理的设计.因为重构规则是对一类重构的抽象,所以,无论是其所针对的设计缺陷还是用于替换该设计缺陷的更合理的设计,在重构规则的描述上都表现为设计模式.所以,模型重构描述语言首先必须具有足够的表达能力以描述常见的设计模式.

下面逐一介绍重构描述语言 GraTDeLMoR 用于描述设计模式的基本元素.

### 1.1 类型化的有向图(**typed directed graph**)

本文提出的模型重构描述语言是基于图转换的,所以图是该语言最基本的概念.

定义 **1**(图). 图 $G=(V,E,s,t)$包含节点 $V$、边 $E$ 以及边到节点的映射 $s$ 和 $t$.对任意一条边 $e\in E$,$v_s=s(e)\in V$ 为边

$e$ 的始点,$v_t=t(e)\in V$ 为边 $e$ 的终点.

因为大多数软件建模语言和编程语言都是类型化的,所以通常采用类型化的带标签的有向图作为重构描述语言的基础.

定义 2(类型化带标签的有向图(**typed labeled directed graph**)). 类型图(type graph)$TG=(V_1,E_1,s_1,t_1,Name_1)$.其中,$Name_1$为节点和边的名称赋值.图 $G=(V_2,E_2,s_2,t_2,TypeV_2,TypeE_2,Name_2)$是符合类型图 $TG$ 的类型化的带标签的有向图,则必须存在一个映射 $f=(f:E\to E,f:V\to V)$,并且有 $f_V\circ s_2=s_1\circ f_E$, $f_V\circ t_2=t_1\circ f_E$, $TypeV_2=Name_1\circ f_V$, $TypeE_2=Name_1\circ f_E$. $TypeV_2$ 为节点分配类型,$TypeE_2$ 为边分配类型.

类型化的有向图必须符合某个类型图的定义.类型图实质上也是一个有向图.它的每个节点(边)都定义了一个“类型”,而该类型的名称就是节点(边)的名称.符合该类型图的类型化有向图,其节点和边的类型必须来自类型图的节点和边所定义的类型.因此有 $TypeV_2=Name_1\circ f_V$ 和 $TypeE_2=Name_1\circ f_E$ 成立.

利用模型重构描述语言 GraTDeLMoR 可将软件模型描述为一个类型化的有向图,其中每个节点表示一个模型元素,而每条边表示元素之间的一个关系.相应的类型图来自建模语言的元模型.由软件模型转换得到的类型化有向图必须符合相应的类型图.也就是说,其节点和边的类型必须分别来自建模语言所定义的实体类型(比如 UML 的类)和关系类型(比如 UML 的继承关系).类型图、类型化有向图、模型以及元模型之间的关系如图 1 所示.

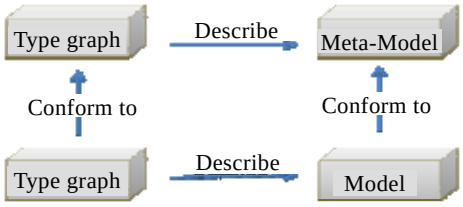



Fig.1 Relationships among type graphs, typed graphs, meta-models, and models

图 1 类型图、类型化有向图、元模型及模型之间的关系

图 2 简要展示了如何利用类型化有向图表示设计模型.节点和边的标签由两部分组成,如“Class:Person”.其中,“Person”为节点的名称,“Class”为节点的类型.在描述重构规则时,常常无须关注节点的名称,此时可以将节点名省略.图 2 中的边因为没有名字所以也只需标示类型即可.

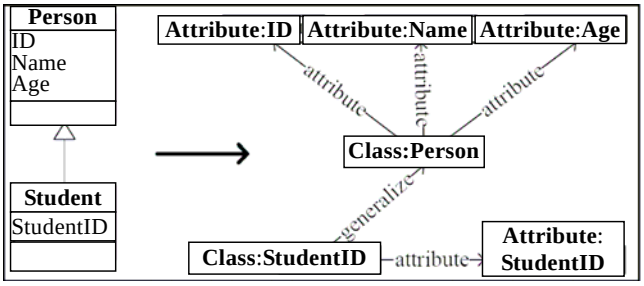



Fig.2 Design models as typed directed graph

图 2 以类型化有向图表示设计模型

### 1.2 自由变量

自由变量是设计模式常用的一个表达要素.模式之所以成为模式(而不是模式的一个实例),就在于其自由变量的存在.为模式的自由变量赋不同的值,则得到模式的不同匹配(实例).

图 3 描述的模式要求两个类拥有一个同名的属性.图 3 的节点“3:Attribute:$x$”和节点“4:Attribute:$x$”包含变量 $x$.在模式匹配过程中,变量 $x$ 可以被赋予任何有意义的值.但无论 $x$ 的值如何变化,都可以保证节点“3:Attribute:$x$”和“4:Attribute:$x$”的实例具有相同的名称.自由变量的作用域为变量所在的规则,作用域内同名的变量即视为同一变量.

图 3 的节点“1:Class”和“2:Class”都没有标明其节点名称,但这并不意味着节点的实例没有名称.没有标明名称的节点(比如“1:Class”)其实是省略了自由变量,其实例的名称可以任意取值.所以,节点“1:Class”的实例可以是任意名称的类.

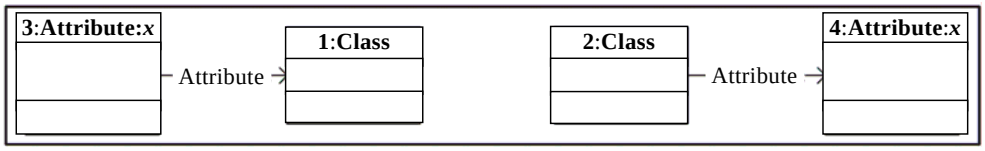



Fig.3 Free variables

图 3 自由变量

### 1.3 多重性

在基本的有向图中,节点并没有多重性.所以,如果采用基本有向图描述模式,则在模式匹配中每个节点都只有一个像点.但是在模型重构中,设计缺陷的匹配(实例)经常包含集合元素.

节点的多重性(cardinality)是描述集合的有效方式.无论是固定大小的集合还是可变大小的集合,集合的大小都可以用多重性来表示.多重性的表示如图 4 所示.多重性的表示与 UML 的多重性表示类似,采用 *m*…*n* 的形式来表示.为了将节点的多重性与其标签相区分,将节点的信息分成多个独立的信息栏表示(类似于 UML 类的表示法),并将节点标签写入第 1 栏而节点多重性写入第 2 栏(如图 4).此外,为了区分一般节点和集合节点(具有多重性),可以使用层叠矩形表示集合节点,而普通节点采用单个矩形表示.

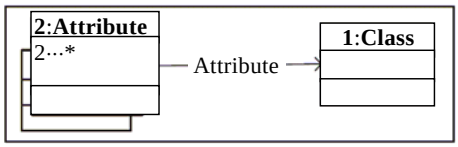



Fig.4 Cardinality

图 4 多重性

图 4 中的“2:Attribute”的多重性为 2…*,表示这个集合的大小必须大于等于 2(即类必须包含至少两个属性).节点标签的最前面加上数字标号可用于区分节点(类型甚至名称相同的节点).数字标号的另一个主要作用是用于标记模型重构规则的左右模式的映射关系.

对集合和多重性的表示,作如下限制:

(1) 整体性

每个集合必须作为一个整体操作.如果集合内的某个元素作了修改,则必须对集合内的所有元素作同样的修改.因为集合作为一个整体操作,所以在模型重构规则的右模式不必标注相应集合的多重性,其大小必须等于左模式中相应集合的大小.将集合作为整体操作不但有利于简化重构工具的实现,还有利于重构规则之间的冲突检测.重构规则的冲突检测主要靠关键对技术.而关键对的计算是通过枚举重构规则的左模式的所有重叠方式(overlap)来实现的.如果集合不是作为一个整体操作,两个集合之间的叠加方式的数量必将急剧增长.如果集合的上限是无穷大(“*”),则将有无穷多种叠加模式.而将集合节点作为一个整体则可以和普通节点一样处理.

(2) 统一处理与多重性默认值

单一的元素(普通的节点)也可以看作是多重性为 1 的集合.这样可以将所有节点都统一作为集合处理.如果某个节点没有明确标示多重性,则该节点的多重性默认为 1.将所有节点统一作为集合处理有利于简化描述语言及其执行引擎.

### 1.4 集合的嵌套

集合之间的关系必须等同地作用于集合内的所有元素,如图 5 所示的模式中有两个集合 *A* 和 *B*,其中 *A* 的大小为 2,*B* 的大小为 3.按照第 1.3 节中集合的解释,在该模式的任意一个匹配中,两个 *A* 元素都必须分别与 3 个 *B* 元素相关联(如图 5 左下方所示).此时,每个 *A* 元素都与相同的 3 个 *B* 元素相关联.但是有时可能需要表达另一种情况,即每个 *A* 元素都有 3 个不同的 *B* 与之关联,如图 5 右下方所示.有一些重构规则可能需要这种模式,比如组合零散小类.如果在软件设计中出现多个孤立的零散小类(属性和操作都很少的类),则应该将这些小类打包以利于软件的维护.此时每个小类都带有一些属性和操作,但这些属性和操作不必相同.为了描述类似的重构规则,本文引入嵌套机制.每个节点(包括表示集合的节点)都可以嵌套.图 6 的嵌套模式可以比较方便地描述图 5 中右下方所示的模式.上文提到的零散小类可以由图 7 所示的嵌套模式描述.在该模式中,应该包含 2~5 个类,每个类最多包含 3 个属性和 3 个操作,而且这些属性和操作不必相同.

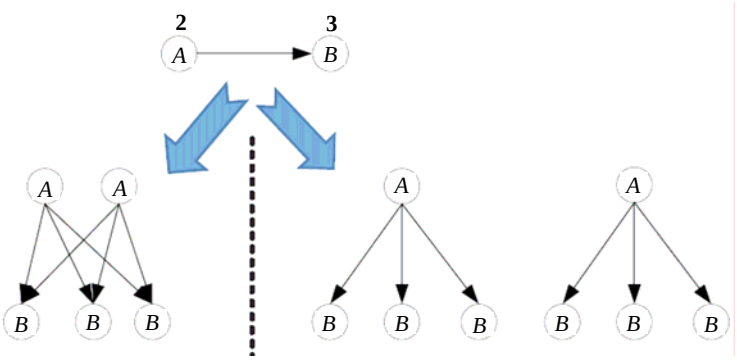



Fig.5 Relationships between sets

图 5 集合间的关联

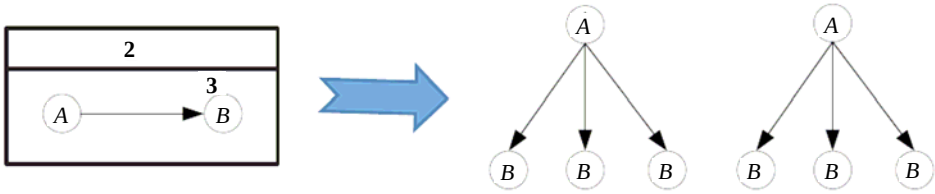



Fig.6 Nested patterns

图 6 嵌套模式

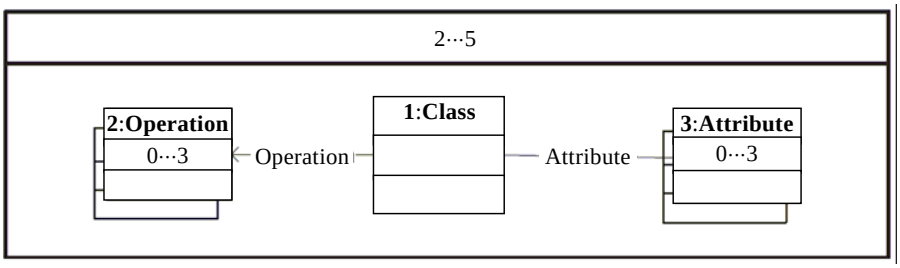



Fig.7 Small classes

图 7 零散小类

### 1.5 序列(**sequence**)

在 UML 设计模型中,不但存在集合也存在序列.比如消息序列在用况图(use case diagram)和顺序图(sequence diagram)中都有出现.如图 8 所示的 4 个消息依次相邻,在时间上有严格的时序关系.所以无法用集合(set)来表示这 4 个有偏序关系的元素.

本文采用如图 9 所示的序列来描述图 8 所示的消息序列.在这个序列中,一共包含 4 个元素(以序列的多重性来表示),元素之间顺次以时序关系(偏序关系)"—*follow*→"相连接.

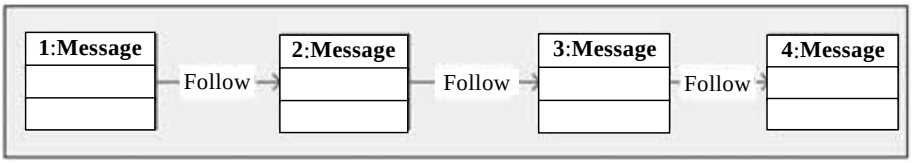



Fig.8 Message sequence

图 8 消息序列

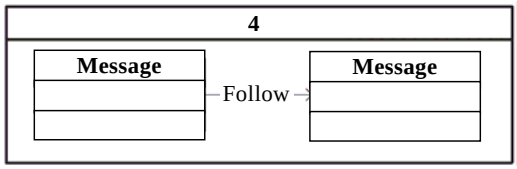



Fig.9 Sequence

图 9 序列

### 1.6 负面应用条件

引入负面应用条件(negative application conditions,简称 NAC)[17]可以用于说明规则所不允许出现的结构元素.有许多重构规则会用到 NAC,比如“删除空类”的重构规则.该规则要求被删除的类不能包含任何域或操作.可以使用如图 10 所示的负面应用条件来表示这个约束:NAC1 要求该类不能包含属性;NAC2 要求该类不能包含操作.一条重构规则可以带有多个 NAC,只有满足所有 NAC 约束之后才能启动该条重构规则.

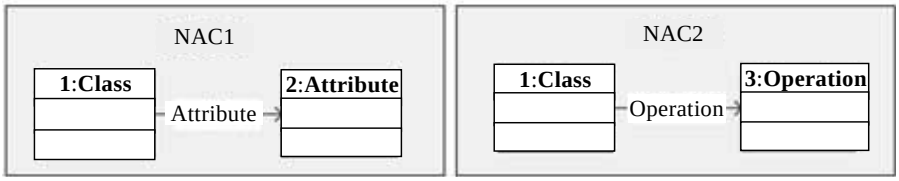



Fig.10 Negative application conditions

图 10 负面应用条件

### 1.7 不存在(non-exist)

重构规则“消除数据块”的前提条件是不存在某个只包含其中一个数据域而没有包含另一个数据域的类(或方法)[18].虽然 NAC 可以表示不存在某类元素的约束,但是因为 NAC 不允许嵌套,所以无法描述重构规则“消除数据块”的前提条件(NAC 中包含带“不存在”的条件).本文使用如图 11 所示的叉号“×”来表示不存在(NE).图 11 的 NAC 解读为:不存在某个包含“属性 3”而不包含“属性 2”的类.

NE 也可以直接出现在重构规则的左模式中以替代某些简单的 NAC.如图 10 所示的两个 NAC 都可以通过在重构规则的左模式中直接加入 NE 表达式来说明.但是,NAC 有利于将转换规则与转换条件分离从而提高可读性,所以一般情况下应该优先使用 NAC.只有在 NAC 无法描述的情况下,才应该考虑使用 NE.

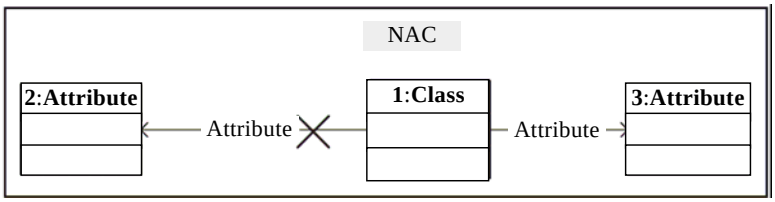



Fig.11 NAC with non-exist

图 11 带不存在关系的 NAC

### 1.8 所有(for all)

在模式的表述中,常常需要排他性的描述.比如描述一个完全公开的类,即其所有属性和操作都必须是公开的.该规则的关键是要覆盖所有的属性和操作.

为了较好地描述以上模式中的所有约束,本文引入“For All”用于标示边的多重性.在如图 12 所示的模式中,在类“1:Class”和属性:“2:Attribute”的关系(拥有关系)上,在属性“2:Attribute”一端加了一个多重性“ALL”.这就意味着,类的所有属性都必须包含在节点“2:Attribute”的实例中.

边的多重性可以和节点的多重性一起出现.在如图 12 所示的模式中,节点“2:Attribute”所代表的集合必须包含类“1:Class”的所有属性(边的多重性约束),同时要求集合的大小必须大于等于 2(节点的多重性约束).

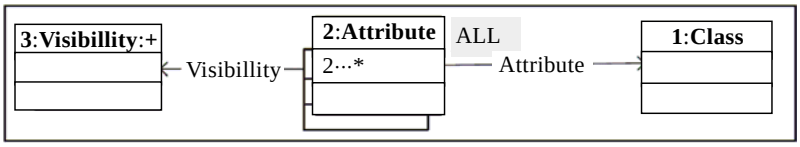



Fig.12 All attributes are public

图 12 对外公开所有属性

### 1.9 OCL约束

以上几个简单扩展可能无法描述某些复杂约束.假定图 7 所示的重构规则“零散小类”要求集合“3:Attribute”的大小 $Size_3$ 与集合“2:Operation”的大小 $Size_2$ 必须满足不等式:$Size_2+Size_3\leq 4$,即要求零散小类的属性和操作的数量之和不大于 4.因为该条约束涉及两个集合的大小关系,无法直接使用集合的多重性来表示该约束.

为此,本文引入对象约束语言(object constraint language)[19]来描述复杂约束关系.OCL 是 OMG 为描述 UML 模型的约束而引入的一种形式化面向对象描述语言.OCL 虽然是形式化语言,但比一般的形式化语言更易于阅读[19].其语法与面向对象编程语言类似,有利于软件设计人员学习和使用.在 OCL 中,$Size_2+Size_3\leq 4$ 是一条标准的 OCL 不等式约束.

为了便于约束检查,也为了便于关键对检测,本文对 OCL 约束进行如下限制:

(1) OCL 约束不能涉及 *NACs*、左模式以及右模式之外的其他模型元素.在检查 OCL 约束时,只需要 *NACs*、左模式以及右模式中的元素就够了.如果 OCL 约束涉及其他模型元素,则该重构规则能否实施还取决于重构规则之外的其他元素.这不但加大了检查 OCL 约束的难度,也大大增加了关键对检测(关键对检测技术是冲突检测的基础)的难度.关键对检测需要从左模式与右模式的所有叠加方式中找出所有可能导致冲突的情况.如果 OCL 约束还涉及左模式及右模式之外的其他模型元素,则关键对检测必须枚举相关元素的所有可能取值,从而加大了关键对检测的计算量.

(2) OCL 对集合元素的约束必须均等地作用于集合内的每个元素.不允许对集合内的个别元素进行约束.这符合将集合作为整体处理的方式(参见第 1.3 节),有利于简化模型重构工具的实现以及重构规则之间的冲突检测.

对OCL的两条约束是权衡语言的表达能力和推理能力的一个折衷方案.一方面,可以充分利用OCL以增强重构描述语言的表达能力;同时,对 OCL 进行一定的限制以便于实现重构的推理和验证.如果没有以上两条限制,模型重构的约束检查以及关键对检测都将因为 OCL 的引入而变得异常困难.引入这两条约束后,现有的约束检查和关键对检查算法略作调整即可处理带 OCL 约束的重构描述.

## 2 重构规则描述

上节简要介绍了模型重构描述语言的基本元素.本节介绍如何利用这些基本元素描述模型重构.

### 2.1 简单模式及模式匹配

定义 **3**(简单模式). 简单模式 $SM=(V,E,s,t,TypeV,TypeE,X,Name)$,其中 $V,E$ 分别表示模式的节点和边;关系 $s,t$ 分别表示边的始点和终点;$TypeV,TypeE$ 分别为模式的节点和边分配类型,$Name$ 为节点和边的名字赋值,$X$ 为模式中包含的自由变量.

如果简单模式的节点或边没有标明其名称,则默认为省略了自由变量.在进行模式匹配时需要自动添加省略的自由变量.在简单模式中,节点和边都不包含多重性,其实质为一个类型化的有向图.

定义 **4**(简单模式匹配). 简单模式 $SM=(V_1,E_1,s_1,t_1,TypeV_1,TypeE_1,X,Name_1)$,模型 $G=(V_2,E_2,s_2,t_2,TypeV_2,TypeE_2,Name_2)$,变量 $X$ 的值域为 $D$,如果存在满射 $m=(m_E:E_1\rightarrow E_2,m_V:V_1\rightarrow V_2)$及映射 $Assign:X\rightarrow D$,使得

$m_V \circ s_1 = s_2 \circ m_E$, $m_V \circ t_1 = t_2 \circ m_E$, $TypeV_1 = TypeV_2 \circ m_V$, $TypeE_1 = TypeE_2 \circ m_E$, $Assign \circ Name_1 = Name_2 \circ m$, 则称模型 $G$ 为模式 $SM$ 的一个匹配,记为 $Match(SM,m)=G$.

模式匹配要求映射为满射,即匹配 $Match(SM,m)$的所有元素都必须在模式 $SM$ 上有对应的源像点.匹配还要求每条边的始点和终点在映射之后也必须是该边映射之后的始点和终点,所以有 $m_V \circ s_1 = s_2 \circ m_E$ 和 $m_V \circ m_1 = t_2 \circ m_E$ 成立.同时要求映射前后的元素(节点或边)的类型必须保持一致.模式匹配的过程中需要为模式的自由变量赋值,使得映射前后的元素具有相同的名称.

### 2.2 复杂模式及模式匹配

如前文所述,简单模式实质上是一个类型化的有向图,其表达能力有限.这也是本文基于图转换描述语言提出模型重构描述语言的主要原因.下面介绍的复杂模式可以包含本文提出的所有元素,这使得复杂模式具有更强的表达能力.

定义 **5**(复杂模式). 复杂模式 $M=(V,V_{set},V_{seq},V_{nest},E)$.其中,$V_{set}\subseteq V$ 表示模式中的集合节点(没有标记多重性的普通节点可以作为多重性为 1 的集合节点),$V_{seq}\subseteq V$ 表示模式中的序列,$V_{nest}$ 表示嵌套节点,且 $V_{set}\cup V_{seq}\cup V_{nest}=V$.$E$ 为模式中各个节点之间的边.每个集合节点 $v_{set}=(v_{simple},c)$由一个普通节点 $v_{simple}$ 和多重性 $c$ 组成.每个序列节点 $v_{seq}=(m,e,c)$由子模式 $m$、边 $e$ 和多重性 $c$ 组成.每个嵌套节点 $v_{nest}=(m,c)$由子模式 $m$ 和多重性 $c$ 组成.边 $e=(Type,name,v_1,v_2,Isforall_1,Isforall_2)$包括类型 $Type$、名称 $name$、两个端点 $v_1,v_2$ 以及分别对应于两个端点的多重性 $Isforall_1,Isforall_2$.

定义 **6**(复杂模式匹配). 模型 $O_1=(V_1,E_1)$为模式 $M=(V,V_{set},V_{seq},E)$的一个匹配(实例),当且仅当:

(1) 对模式中的任意一个集合节点$v=(v_{simple},c)\in V_{set}$,在模型$O_1$中必须存在简单节点$v_{simple}$的$c$个实例(匹配)$(match_1,match_2,\ldots,match_c)$.

(2) 对模式中的任意一个序列节点$v=(m,c)\in v_{seq}$,在模型$O_1$中必须存在子模式$m$的$c$个实例(匹配)$(match_1,match_2,\ldots,match_c)$,并且对任意$i\in[1,c]$,必须存在边$oe=(match_i,match_{i+1})\in E_1\wedge Type(oe)=Type(e)$.

(3) 对模式中的任意一个嵌套节点$v=(m,c)\in V_{nest}$,在模型$O_1$中必须存在子模式$m$的$c$个实例(匹配)$(match_1,match_2,\ldots,match_c)$.

(4) 对任意边$e=\langle v_1,v_2\rangle\in E$,则在节点$v_1$的任意实例$match_i$和节点$v_2$的任意实例$match_j$之间必须存在$e$的实例:$oe=(match_i,match_j)\in E_1\wedge Type(oe)=Type(e)$.如果 $e\cdot Isforall_1=1$,则不存在节点$v_1$的实例$match_i\not\subset O_1$.如果 $e\cdot Isforall_2=1$,则不存在节点$v_2$的实例$match_j\not\subset O_1$.

复杂模式匹配(定义 6)采用递归的方式定义.嵌套节点可以嵌套子模式.复杂模式经逐层分解之后,最后必然递归到简单模式,采用定义 4 的简单模式匹配判定其所对应的匹配是否存在.然后逐层返回判定每个嵌套层次的复杂模式匹配是否存在,最后判定模型 $O$ 是否为复杂模式 $M$ 的匹配.

### 2.3 模型重构及重构规则

无论是简单模式还是复杂模式,在模型重构描述语言中,模式的描述都是为描述模型重构或重构规则服务的.下面介绍如何描述模型重构及重构规则.

定义 **7**(模型重构规则). 模型重构规则 $p:NACs,Constraints\mapsto L\xrightarrow{f}R$.其中 $L$ 和 $R$ 分别称为左模式和右模式,映射 $f$ 则用于指明 $L$ 和 $R$ 之间的关系.$NACs$ 是负面应用条件,$Constraints$ 是 OCL 约束.

重构规则的左模式 $L$ 用于指明本规则所针对的设计缺陷(bad smells).而重构规则的右模式 $R$ 则给出了与左模式“等效”但更合理的设计.这里的等效主要是指用右模式的实例(匹配)替换左模式的相应实例(匹配)可以保持软件的外部行为特性不变.而替换的好处则是模型质量的改善.映射 $f$ 则用于描述替换过程,详细说明该删除、修改或添加哪些元素.$NACs$ 和 $Constraints$ 是重构规则实施的前提条件.前提条件可能是对设计缺陷的进一步限定,也可能是对规则应用环境的限制.只有在同时满足 $NACs$ 和 $Constraints$ 约束的前提下才能启动重构规则.重构规则 $p:NACs,Constraints\mapsto L\xrightarrow{f}R$ 的执行语义由定义 8 详细说明.

定义 **8**(模型重构). 给定模型 $G$、模型重构规则 $p:NACs,Constraints\mapsto L\xrightarrow{f}R$,以及模式匹配 $m:L\to G$,重

构规则 $P$ 通过匹配 $m$ 实施于模型 $G$ 上的一次重构可以表示为 $t: G \xrightarrow{p,m} H$ .其计算过程可以表示为如图 13 所示的推出(pushout).

$$\begin{array}{ccc} L & \xrightarrow{f} & R \\ \downarrow m & & \downarrow m^* \\ G & \longrightarrow & H \end{array}$$

$$\forall c: Constraint \in Constraints \cdot c(G,m) = \text{True}$$
$$\forall nac: NAC \in NACs \cdot nac(G,m) = \text{True}$$

Fig.13 Model refactoring based on graph transformation

图 13 基于图转换的模型重构

模型重构 $t: G \xrightarrow{p,m} H$ 是根据模型重构规则 $p: NACs, Constraints \mapsto L \xrightarrow{f} R$ 对模型 $G$ 实施改进.重构的结果是将模型 $G$ 变成 $H$.重构的具体步骤如下:

(1) 在待重构的模型$G$上找出重构规则$p$的左模式$L$的匹配$Match(L,m)$.如果匹配不存在,则重构活动结束,否则进入下一步.

(2) 根据映射$m$和模型$G$判定重构规则的负面应用条件约束$NACs$是否满足.如果不满足约束$NACs$,则重构活动结束,否则进入下一步.

(3) 根据映射$m$和模型$G$判定重构规则的OCL约束$Constraints$是否满足.如果不满足约束$Constraints$,则重构活动结束,否则进入下一步.

(4) 根据映射$f$确定并创建需要创建的新元素(如果$R$中某个元素在$L$中没有对应的源像点,则需要创建该元素).

(5) 根据映射$m$为新创建元素的变量赋值.

(6) 根据映射$f$确定并删除需要创建的元素(如果$L$中某个元素在$R$中没有对应的像点,则需要删除该元素).

同一条规则可能在某个模型上实施多次.每次重构都需要定位左模式的一个实例(即需要确定匹配映射 $m$).

## 3 实 例

本节通过两个典型的模型重构规则来讨论本文提出的模型重构描述语言的描述能力,并将本文提出的模型重构描述语言与现有描述语言相比较.

首先介绍两条模型重构规则:“提升公共域”和“消除重复域”.提升公共域是指将子类所共享的公共域提升到它们的公共父类中[2],这样有利于复用,也有利于简化系统设计以便于理解和维护.

定义 **9**(提升公共域(**pull up field**)). 如果某个父类 $C_s$ 的所有子类都包含某个共同的域 $f$,则应将公共域 $f$ 从子类提升到父类 $C_s$.

提升公共域要求父类的“所有”子类都必须包含该公共域.如果只是部分子类包含了公共域,就只能插入一个新的父类作为共享这些公共域的子类的直接父类,然后将公共域提升到这个新类中.这就是下面的“消除重复块”重构规则.

定义 **10**(消除重复域(**remove duplicate filed**)). 如果某个父类 $C_s$ 的部分(至少大于等于两个)子类都包含某个共同的域 $f$,则应添加空类 $C_{NEW}$ 作为 $C_s$ 的子类,包含公共域 $f$ 的子类则不再作为 $C_s$ 的直接子类而是作为 $C_{NEW}$ 的直接子类.同时,将公共域 $f$ 从子类提升到 $C_{NEW}$.

重构规则“提升公共域”和“消除重复块”的主要不同在于包含公共域的是“所有”子类还是“部分”子类.因为针对不同的情况必须采取不同的重构措施,所以必须在重构规则的描述中明确它们之间的区别.但是,经典图转换理论中没有相应的概念用于描述“所有”或者“部分”,所以也就无法区分以上两个规则.在经典图转换中,每个节点的多重性都是 1,所以无法描述节点个数不固定的情况.其他形式化重构描述语言也没有提供这两个概念,所以也无法描述和区分这两条典型的重构规则.

本文提出的模型重构描述语言则可以比较方便地描述以上两个重构规则.具体描述如图 14、图 15 所示.在“提升公共域”的左模式中,首先通过给子类到父类的边标示多重性“ALL”来表示全部,即所有和父类存在关系“Generalize”的类都必须包含在这个集合中.这就意味着节点“2:Class”表示父类“1:Class”的所有子类.此外,图 14 所示的左模式还通过给嵌套节点标示多重性“2…*”来表示拥有公共属性的子类必须大于等于两个.如果父类只有 1 个子类,则本规则并不适用.每个子类拥有各自的属性,但是这些属性必须具有相同的类型“5:Type”和名称“*x*”.

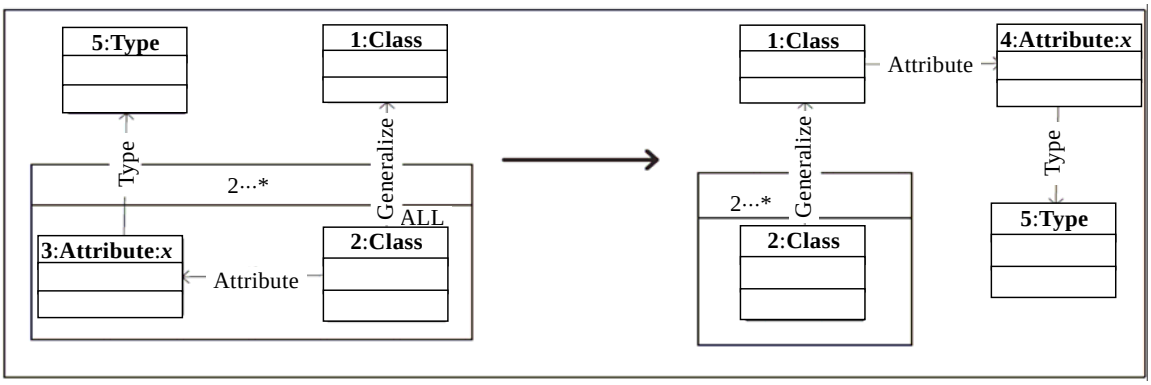



Fig.14 Pull up common fields

图 14 提升公共域

重构规则“消除重复域”如图 15 所示.在规则的左模式中,父类“1:Class”的子类包括“4:Class”和嵌套节点内的“2:Class”两部分.虽然“2:Class”包含属性“3:Attribute:*x*”,但子类“4:Class”则不包含与属性“3:Attribute:*x*”同名且同类型的属性.这就保证了父类的部分(至少 1 个)子类没有包含公共属性.

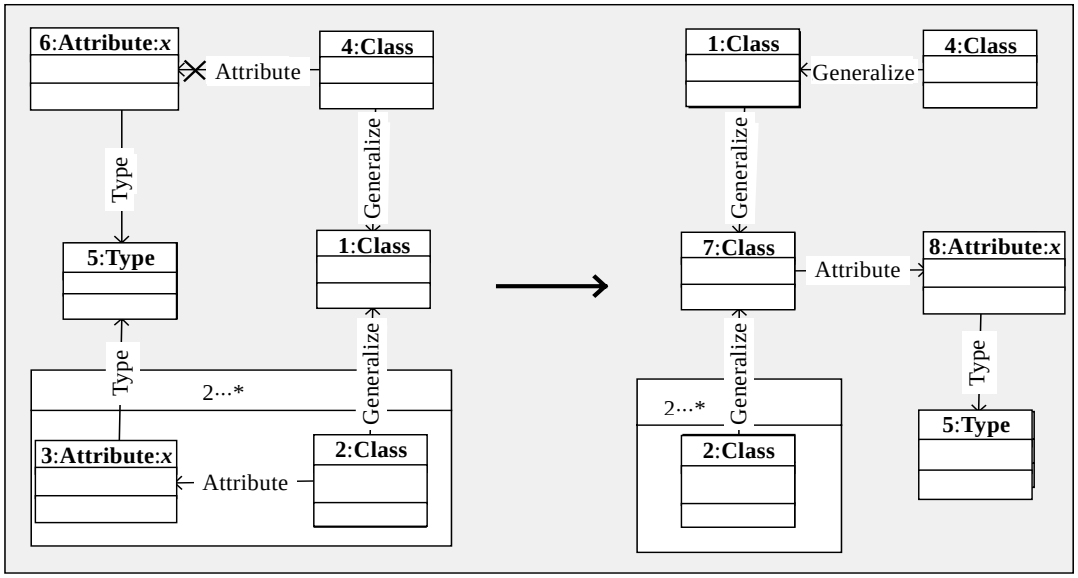



Fig.15 Remove duplication

图 15 消除重复域

## 4 支撑工具

课题组在软件建模和模型重构方面具有较好的研究基础,不但在模型重构的多个方面展开研究,而且还提供了较为完整的 CASE 支撑工具.CASE 工具的体系结构如图 16 所示.

CASE 工具基于北京大学软件工程研究所开发的建模工具 JBOO5.JBOO5 基于开放的 Eclipse 平台,为重构插件的实现和集成提供了便利.模型重构 CASE 工具本身也是一个插件集,包含重构描述语言编辑器、重构描

述语言解析器、缺陷检测、冲突检测、重构调度和执行引擎这 6 个.

重构描述语言编辑器提供创建、编辑模型重构规则和模型重构约束的平台,如图 17 所示.编辑器的左侧是元模型,用于提供节点和边的类型.编辑器的中心区域是规则编辑器,箭头的左侧为重构规则的左模式,箭头右侧为重构规则的右模式.左、右模式的映射关系由节点和边的编号标明.可以从画图面板创建节点,也可以从编辑器左侧的元模型中拖入相应的元模型元素创建相应类型的节点.

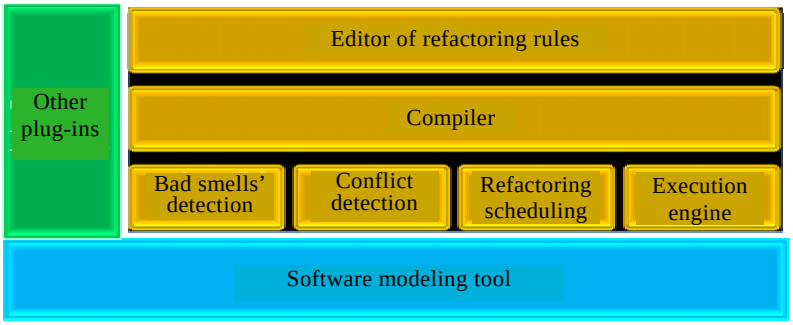



Fig.16 Tools for model refactoring

图 16 模型重构工具集

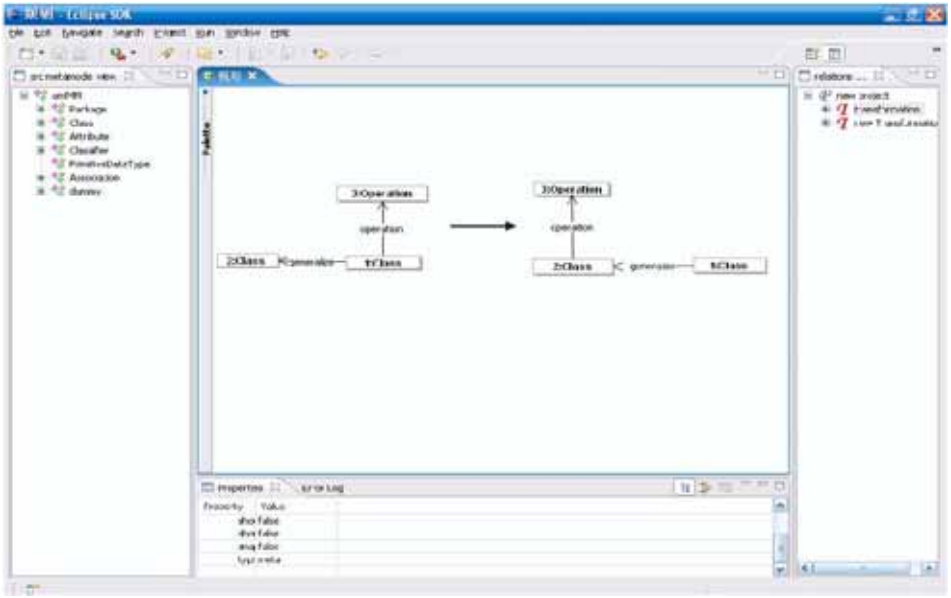


Fig.17 Editor of model refactoring description language

图 17 模型重构描述语言编辑器

CASE 工具的其他插件则基于模型重构描述语言编辑器的输入结果.模型重构描述语言还可用于描述模型重构约束[20],从而将模型重构规则与重构约束统一为一种格式.首先由解析器解析并检查输入的重构规则和重构约束.接着由冲突检测插件检测重构规则和重构约束之间的冲突关系.如果检测到冲突,则通过编辑器提示用户.这可能意味着需要修改重构规则或者重构约束.之后,开始执行模型重构:由缺陷检测插件找出重构规则左模式的所有实例,交给冲突检测插件检测各个可用重构之间的冲突关系,再提交给重构调度插件安排重构的执行顺序[21],最后由执行引擎按顺序执行重构.

在软件重构的主页[22]上列出了 94 条重构规则(对软件重构的研究依然在继续,而这个列表也在不断增长),其中可以作用在模型上的共计 38 条.我们利用 CASE 工具已经将这 38 条重构规则全部形式描述为基于图转换的模型重构规则.这也在一个侧面验证了本文提出的模型重构描述语言的描述能力.

CASE 工具运用于北京大学软件研究所的元建模工具 PKU-Meta-Modeler 的重构中[23],取得了预期的效果.

不但降低了软件重构的成本,还提高了软件重构的效果和可靠性.具体参见文献[23].

## 5 讨 论

描述语言的表达能力和复杂程度将影响 CASE 工具的性能和效率.比如引入的 OCL 约束虽然增强了描述语言的表达能力,但也增加了约束检查以及关键对检测的计算量.因此,需要综合考虑语言的表达能力和 CASE 工具的性能及效率.本文提出的基于图转换的模型重构描述语言针对模型重构进行扩展,提高了语言的表达能力.同时,为了降低 CASE 工具的开发难度,提高 CASE 工具的执行性能,本文对扩展实行严格限制.本文对 OCL 约束的限制就是一个典型的例子.得益于这些限制,CASE 工具的开发可以复用现有的图转换执行引擎,而且基于图转换的约束检查以及关键对检测也可以大量复用现有的算法.CASE 工具在图转换、约束检查、关键对检测等方面的性能都接近于未作扩展的图转换工具 AGG[24,25](约为 AGG 的 85%~95%).

模型重构描述语言具有良好的可扩展性.为了权衡语言的表达能力和 CASE 工具的性能与效率,模型重构描述语言只对经典图转换描述语言进行少量的关键扩展.模型重构描述语言是基于经典图转换描述语言的一般扩展,现存的基于经典图转换描述语言上的扩展都可以集成到重构描述语言中.在不同的使用环境下,可以根据需要在此基础上实施进一步的扩展.也可以根据具体应用领域的特性,关闭某些扩展以提高重构的执行效率.扩展后的描述语言需要重新定义模式匹配算法.在 CASE 工具的设计上,也相应地采用了开放平台的设计,以支持描述语言的进一步扩展.

重构描述语言的复杂程度对模型重构性质的推导也有较大的影响.现有的性质推导方法通常依赖于经典图转换描述语言的相关特性,对扩展元素比较敏感.比如模型重构的冲突检测[26]是基于经典图转换的关键对检测技术,对描述语言的元素及其性质具有依赖性.所以,对经典图转换描述语言的扩展必须权衡语言的表达能力和性质推导能力.在经典图转换描述语言上的扩展增加了模型重构性质推导的难度.本文对扩展元素进行了严格限制,努力降低扩展元素对性质推导带来的不利影响.初步实验结果表明,在 CASE 工具上实现的模型重构冲突检测算法在效率上接近于基于经典图转换的冲突检测算法(约为后者的 87%).

## 6 相关工作

软件重构作为软件开发的关键活动之一受到越来越多的关注.重构规则的描述是其中的一个重要方面.Opdyke[1]和 Tichelaar[13]使用自然语言来描述软件重构规则.Opdyke[1]还通过重构规则的前置条件来保证软件重构满足某些重构约束.虽然重构前置条件的描述采用了形式约束语言,但是作为主体的重构操作步骤却使用自然语言描述.虽然自然语言描述易于书写和阅读,但是不利于机器的理解和执行,同时也容易产生二义性.为了消除自然语言描述的二义性,人们开始使用形式语言来描述软件重构[5,14].

因为图转换具有坚实的数学理论基础[27]以及较为成熟的支撑工具[28−30],基于图转换的软件重构描述语言逐渐成为主流的软件重构描述语言[5,14,31,32].因为模型,特别是建立在以 UML 为代表的可视化建模语言之上的软件模型,与图(graph)有着天然的联系,使得图转换更成为模型重构描述语言的首选之一[33].Mens 等人[14]探讨了如何利用图转换来描述软件重构规则.Mens 等人在文献[14]中的主要任务是探讨使用图转换描述语言作为软件重构描述语言的可行性,他们没有探讨经典图转换语言作为软件重构描述语言的表达能力问题.Paolo 和 Francesco[31]则利用分布式图转换(distributed graph transformation)描述集成重构(integrated refactorings).van Eetvelde 和 Janssens[32]则意识到经典图转换描述语言在描述软件重构中的一些不足,他们对经典图转换进行了一定的扩展.主要扩展包括两点:图提炼(refinement of graphs)[34]和重复块(duplications).这两个扩展都是为了能够用一个单一的图转换规则描述复杂的软件重构规则.但是这种扩展比较晦涩难懂,一般的软件开发人员如果没有一定的数学理论基础是很难理解的.此外,虽然重复块的作用与集合类似,但其效果总是相当于多重性为 1…*n*.本文引入的集合表示以及多重性表示法则显得更加灵活,表达能力也更强.同时,本文还在其他方面进行了更为广泛的扩展,以提高语言的描述能力.Agrawa 等人[35]基于图转换设计了一门模型转换描述语言,其中的模式描述语言部分与本文提出的重构描述语言类似.但是文献[35]的扩展限于多重性与嵌套的表示和语义.而

本文则针对模型重构的特征提出了更多的扩展元素,使描述语言的描述能力得到进一步的提高.另一方面,因为Agrawa等人[35]的扩展是针对一般的模型转换的,其扩展并不针对模型重构,也没有考虑扩展对模型重构性质推导带来的影响.UML模型的查询/视图/转换标准语言MOF QVT(meta object facility 2.0 query/view/transformation specification)[36]也是基于图转换的描述语言.其主要特点在于集合了图转换与OCL约束描述语言.Thomas[37]也试图将图转换语言与OCL相结合以增强语言的表达能力,但其目的是用于描述软件合约而不是模型重构规则.在MOF QVT[36]和文献[37]中,OCL的使用几乎不受任何限制.本文提出的重构描述语言对OCL约束进行了限制,其目的在于简化设计和实现,同时便于重构规则的性质推导.QVT和文献[35]的目标在于提供一门模型转换描述语言,而不是模型重构描述语言.他们没有针对模型重构的特征设计语言的元素,也没有考虑重构规则的性质推导和证明.

并不是所有的重构语言都基于图转换描述语言.Ivan Porres[38,39]提出了一种基于脚本语言Python[40]的模型重构描述语言SMW[41].Whittle[42]使用MAUDE作为模型重构描述语言.

重构约束的描述只是模型重构的一部分.基于图转换理论,人们开始探寻如何推导和证明软件重构的某些性质.Mens等人[26]利用经典的图转换理论所提供的关键对分析技术[43]来检测重构规则之间的冲突.重构规则之间的冲突检测是重构调度的基础和前提[26].Mens[14]基于图表达式描述软件重构的行为特性保持约束,进而利用图转换理论证明某个重构规则是否满足行为特性保持约束.刘辉等人[20]将模型重构约束描述为图转换规则,利用图转换之间的冲突检测机制来验证重构规则是否满足特性保持约束.

模型重构的其他活动包括设计缺陷检测[44,45]、重构调度与执行[21]、模型质量度量等.详细相关工作参见文献[5].

## 7 结论和进一步的工作

随着模型重构逐渐成为软件开发的关键活动之一,人们需要准确、无二义地描述模型重构,并需要CASE工具的辅助,以实现自动或半自动化的模型重构.自然语言描述具有二义性不够精确,所以人们需要某种更为形式化的描述语言来实现上述目标.为此,本文提出了一种基于图转换的模型重构描述语言.本文针对模型重构的特征有针对性地设计重构描述语言的9个基本元素,然后给出了利用这些基本元素描述模型重构及重构规则的方法和策略.本文通过两个典型的模型重构规则作为示例讨论了本文提出的模型重构描述语言的描述能力,并与现有描述语言相比较.课题组开发的CASE工具为基于图转换的模型重构提供了较为完整的支持.最后还讨论了本文工作与相关工作之间的区别与联系.

模型重构的描述只是整个重构过程的第一步.还需要根据重构规则找出需要重构的模型实例、实施重构、验证重构的效果是否符合预期,重构是否满足重构约束等等.作者及其所在的研究团队在模型重构的这些方面都进行了一定的研究和探讨[20,21,44,45].研究的目标是提高模型重构的效率和可靠性.通过自动或半自动化地检测需要重构的模型块并借助模型重构工具实现自动或半自动化的模型重构.为提高重构的效果,还需要对模型重构实施合理调度[21].此外,为了提高模型重构的可靠性,还需要描述和验证模型重构约束[20].本文所提出的模型重构描述语言充分考虑了模型重构中后续活动对描述语言的要求.随着后续研究的深入,模型重构描述语言也可能需要进一步的调整.

**References**:

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

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

[3] Pipka JU. Refactoring in a “test first”-world. In: Marchesi M, Succi G, eds. Proc. of the 3rd Int’l Conf. on eXtreme Programming and Flexible Processes in Software Engineering. 2002. 178−181.

[4] van Deursen A, Moonen L. The video store revisited—Thoughts on refactorings and testing. In: Marchesi M, Succi G, eds. Proc. of the 3rd Int’l Conf. on eXtreme Programming and Flexible Processes in Software Engineering (XP2002). 2002. 71−76.

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

[6] Object Management Group. UML 2.0 Infrastructure Specification. OMG Adopted Specification ptc/03-09-15, 2003.

[7] Object Management Group. UML2 Superstructure Final Adopted Specification. 2003.

[8] Object Management Group. MDA Guide Version 1.0.1. OMG/03-06-01, 2003.

[9] Astels D. Refactoring with UML. In: Proc. of the 3rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering. 2002. 67−70.

[10] Sunye G, Pollet D, Traon YL, Jezequel JM. Refactoring UML models. In: Proc. of the 4th Int'l Conf. on the Unified Modeling Language. 2001. 134−148.

[11] Correa A, Werner C. Applying refactoring techniques to UML/OCL models. In: Proc. of the 7th Int'l Conf. on the Unified Modeling Language (UML 2004). 2004. 173−187.

[12] Object Management Group. UML 2.0 OCL specification. Technical Report, ptc/03-10-14, 2003.

[13] Tichelaar S. Modeling object-oriented software for reverse engineering and refactoring [Ph.D. Thesis]. University of Bern, 2001.

[14] 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.

[15] Roberts D, Brant J, Johnson R. A refactoring tool for smalltalk. Theory and Practice of Object Systems, 1997,3(4):253−263.

[16] Boger M, Sturm T, Fragemann P. Refactoring browser for UML. In: Proc. of the 3rd Int'l Conf. on eXtreme Programming and Flexible Processes in Software Engineering. 2002. 77−81.

[17] Lambers L, Ehrig H, Orejas F. Conflict detection for graph transformation with negative application conditions. In: Corradini A, Ehrig H, Montanari U, Ribeiro L, Rozenberg G, eds. Proc. of the Graph Transformations, the 3rd Int'l Conf. (ICGT 2006). LNCS 4178, 2006. 61−76.

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

[19] Object Management Group. UML 2.0 OCL specification. Technical Report, ptc/03-10-14, 2003.

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

[21] Liu H, Li G, Ma ZY, Shao WZ. Scheduling of conflicting refactorings to promote quality improvement. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. 2007. 489−492.

[22] http://refactoring.com/catalog/index.html

[23] Liu H. Graph transformation based model refactoring [Ph.D. Thesis]. Beijing: Peking University, 2008 (in Chinese with English abstract).

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

[25] http://tfs.cs.tu-berlin.de/agg/AGG-ShortManual/node36.html

[26] Mens T, Taentzer G, Runge O. Detecting structural refactoring conflicts using critical pair analysis. Electronic Notes in Theoretical Computer Science, 2005,127(3):113−128.

[27] 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.

[28] Ermel C, Rudolf M, Taentzer G. The AGG approach: Language and environment. In: Ehrig H, Engels G, Kreowski HJ, Rozenberg G, eds. Handbook of Graph Grammars and Computing by Graph Transformation, Vol.2: Applications, Languages, and Tools. River Edge: World Scientific Publishing Co., Inc., 1999.

[29] Nickel U, Niere J, Zündorf A. The FUJABA environment. In: Proc. of the 22nd Int'l Conf. on Software Engineering. 2000. 742−745.

[30] Schürr A, Winter AJ, Zündorf A. The PROGRES approach: Language and environment. In: Ehrig H, Engels G, Kreowski HJ, Rozenberg G, eds. Handbook of Graph Grammars and Computing by Graph Transformation, Vol.2: Applications, Languages, and Tools. World Scientific, 1999. 487−550.

[31] Bottoni P, Presicce FP, Taentzer G. Specifying integrated refactoring with distributed graph transformations. In: Proc. of the Applications of Graph Transformations with Industrial Relevance. 2003. 227−242.

[32] van Eetvelde N, Janssens D. Extending graph transformation for refactoring. In: Proc. of the 2nd Int'l Conf. on Graph Transformation ICGT 2004. LNCS 3256, 2004. 399–415.

[33] 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 Eurupean Conf. on Model Driven Architecture—Foundations and Applications (ECMAD-FA 2005). LNCS 3748, 2005. 284–298.

[34] Drewes F, Hoffmann B, Plump D. Hierarchical graph transformation. Journal of Computer and System Sciences, 2002,(64): 249–283.

[35] Agrawal A, Karsai G, Kalmar Z, Neema S, Shi F, Vizhanyo A. The design of a language for model transformations. Journal of Software and System Modeling, 2006,5(3):261–288.

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

[37] Baar T. OCL and graph-transformations—A symbiotic alliance to alleviate the frame problem. In: Proc. of the Workshop on Tool Support for OCL on ACM/IEEE the 8th Int'l Conf. on Model Driven Engineering Languages and Systems. LNCS 3844, 2006. 20–31.

[38] Porres I. Model refactorings as rule-based update transformations. Technical Report, TUCS Technical Report No.525, Turku Centre for Computer Science, 2003.

[39] Porres I. Model refactorings as rule-based update transformations. In: Proc. of the UML 2003—The Unified Modeling Language, Modeling Language and Applications. 2003. 159–174.

[40] van Rossum G. Python reference manual. Release 2.5.2, 2008. http://docs.python.org/ref/ref.html

[41] Porres I. A toolkit for manipulating UML models. Technical Report, TUCS Technical Report No.441, TUCS Turku Centre for Computer Science, 2002.

[42] Whittle J. Transformations and software modeling languages: Automating transformations. In: Jezequel JM, Hussmann H, Cook S, eds. Proc. of the UML 2002—The Unified Modeling Language, Model Engineering, Languages, Concepts, and Tools, the 5th Int'l Conf. LNCS 2460, Dresden: Springer-Verlag, 2002. 227–242.

[43] 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.

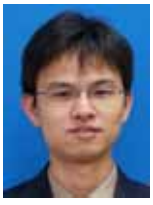
[44] Liu H, Shao WZ, Zhang L, Ma ZY. Detecting overlapping use cases. IET Software (Formerly IEE Proc.—Software), 2007,1(1): 29–36.

[45] Liu, H, Ma ZY, Zhang L, Shao WZ. Detecting duplications in sequence diagrams based on suffix trees. In: Proc. of the XIII Asia Pacific Software Engineering Conf. Los Alamitos: IEEE Computer Society, 2006. 269–276.

附中文参考文献:

[20] 刘辉,麻志毅,邵维忠.模型转换中的特性保持的描述与验证.软件学报,2007,18(10):2369–2379. http://www.jos.org.cn/1000-9825/18/2369.htm

[23] 刘辉.基于图转换的模型重构技术研究[博士学位论文].北京:北京大学,2008.



刘辉(1978－),男,福建长汀人,博士,讲师,主要研究领域为面向对象建模,软件重构,元建模,MDA.

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,主要研究领域为软件工程,软件工程环境,面向对象技术.