软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(4):907−924 [doi: 10.13328/j.cnki.jos.005055]

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

一种随机化的软件模型生成方法*

何 啸[1,2,3], 李文峰[1], 张 天[3], 麻志毅[2], 邵维忠[2], 胡长军[1]

[1](北京科技大学 计算机与通信工程学院,北京 100083)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[3](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

通讯作者: 何啸, E-mail: hexiao@ustb.edu.cn

摘 要: 模型转换是模型驱动开发的核心技术.当要把模型转换用于工业生产时,其性能成为影响这一技术成败的关键因素之一.为了测试模型转换程序的性能,需要能够快速地生成一组具有较大规模的模型数据用于作为测试的输入数据.提出一种随机化的模型生成方法,该方法能够根据元模型的定义以及用户输入的约束条件随机且正确地生成模型文件.实验结果表明:该方法与其他方法相比,具有更好的生成效率,从而更适合支持模型转换的性能测试.

关键词: 模型生成;模型转换;性能测试;随机测试;模型驱动工程

中图法分类号: TP311

中文引用格式: 何啸,李文峰,张天,麻志毅,邵维忠,胡长军.一种随机化的软件模型生成方法.软件学报,2017,28(4):907−924. http://www.jos.org.cn/1000-9825/5055.htm

英文引用格式: He X, Li WF, Zhang T, Ma ZY, Shao WZ, Hu CJ. Randomized approach to software model generation. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):907−924 (in Chinese). http://www.jos.org.cn/1000-9825/5055.htm

Randomized Approach to Software Model Generation

HE Xiao[1,2,3], LI Wen-Feng[1], ZHANG Tian[3], MA Zhi-Yi[2], SHAO Wei-Zhong[2], HU Chang-Jun[1]

[1](School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

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

[3](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

Abstract: Model transformation is the key to model-based software engineering. When the model transformation is applied to industrial developments, its scalability becomes an important issue. To test the performance of model transformations, developers must be able to generate a set of models, i.e. the test inputs, efficiently. This paper proposes a randomized approach to generating large models. This approach can produce a model randomly and correctly based on the definition of metamodel and user-defined constraints. And the evaluation result also shows that the proposed approach is more efficient than other approaches, and therefore is more suitable for supporting performance testing of transformations.

Key words: model generation; model transformation; performance testing; random testing; model-driven engineering

模型驱动的软件工程是一种以模型为核心的软件开发方法,模型和模型转换是其核心成分.模型是对系统的一种抽象描述,而模型转换则是将一种模型变换成另一种模型的特殊程序,它推动了整个开发过程并实现了

* 基金项目: 国家重点基础研究发展计划(973)(2013CB329606); 国家自然科学基金(61300009)

Foundation item: National Basic Research Program of China (973) (2013CB329606); National Natural Science Foundation of China (61300009)

收稿时间: 2014-08-28; 修改时间: 2015-11-18; 采用时间: 2016-02-19; jos 在线出版时间: 2016-05-03

CNKI 网络优先出版: 2016-05-04 08:44:08, http://www.cnki.net/kcms/detail/11.2560.TP.20160504.0844.003.html

开发的自动化.

现有的工作大多都将研究重点放在了转换语言和算法上,如讨论如何实现模型重构[1,2]、实现代码生成[3,4]和模型同步[5,6]等.这些研究增强了模型转换的处理能力,使之能够实现许多复杂的操作.

当把模型转换技术运用到工业生产中时,可伸缩性成为左右其成败的关键因素之一.在被开发(或维护)的系统比较复杂的情况下,被转换的模型可能具有较大规模.如果转换程序不能快速地处理“大模型”,那么这个开发(或维护)过程就可能失败.此外,在大数据时代,高效地处理“大模型”的能力也变得越来越重要.

例如,在基于运行时模型的软件管理技术中,模型同步(即双向模型转换)常常被用于实现系统状态和抽象管理视图之间的连接[7].当系统状态发生改变时,通过正向转换将其映射到管理视图上;系统管理员在管理视图上进行系统调整,并利用逆向转换将系统变化传播到运行时系统上.在这一过程中,模型同步的执行效率至关重要.如果同步操作不能及时将系统状态转换成管理视图,那么管理员则可能无法观测到系统最新的运行情况,从而无法对系统进行正确调整.同理,如果同步操作不能将管理视图上的变化及时传播到运行时系统上,系统状态可能在转换过程中发生改变,从而导致系统调整的错误和失败.

因此,在应用模型转换之前,应该对其性能进行系统化的测试.一方面,可以明确转换程序的执行效率和极限,从而确定其应用范围;另一方面,还可以进一步确定其中的性能瓶颈,从而可以对其进行优化改进.为了进行性能测试,一个关键问题在于如何有效地生成较大规模的输入数据,即模型.模型生成需满足下面 4 个基本需求.

- 语法正确性:自动生成的模型需要符合语法约束;
- 高效性:生成模型的过程必须是高效的;
- 随机性:模型必须被随机生成,以便衡量转换程序的平均执行时间;
- 可配置性:模型生成的过程必须是可配置的,比如,应允许用户限制模型元素的数量等等.

目前主要存在两种类型的自动化模型生成方法——基于求解器的方法和基于算法的方法.在基于求解器的方法中,元模型和生成需求被编码成某种约束表达式,如 Object Constraint Language(OCL)[8]表达式.之后,这些约束表达式会被代入到一个约束求解器中进行求解.如果约束可解,求解器会返回一个符合条件的模型.基于求解器的模型生成方法能够处理复杂的约束条件,但其效率十分低下,无法快速生成较大规模的模型,因此不能很好地支持性能测试.基于算法的模型生成方法通常会使用某种算法生成模型,这类方法能够随机、快速地生成模型.但大部分算法往往忽略了生成过程的可配置性以及一些基本语法约束,这就导致它们生成的模型可能包含语法错误.

本文提出一种针对模型转换性能测试的模型生成算法,该方法具有随机性和可配置性,能够在合理的时间内生成正确的模型.在该方法中,所有的模型元素及关系都将被随机生成.为了生成正确的模型,该方法考虑了元模型中定义的全部语法约束和一部分语义约束.该方法还允许用户自定义约束来指导生成过程,以便具有更好的可配置性.最后,通过一组实验证明该方法能够正确、高效地生成较大规模的模型数据.本文的初步想法已经发表在 COMPSAC 2014 上[9],本文完善了方法描述,并对方法进行了更多的实验评估.

本文方法的主要目标是生成规模较大、语法正确的模型,但并不直接处理用 OCL 描述的语义约束.这样做有以下 3 个原因:(1) 不同于白盒测试,用于性能测试的输入模型对于 OCL 语义约束的要求较低;(2) 在生成过程中处理 OCL 约束会极大地影响模型生成效率;(3) 对于某些转换程序而言,如模型一致性修复程序,不符合 OCL 约束的模型反而是有效的输入.本文方法可以看作模型生成的基线方法,并可以结合其他技术构造满足复杂约束的模型:从理论上看,只要能够快速地生成足够数量的随机模型,一定可以从中选出符合 OCL 约束的结果;从实践上看,可以利用模型修复技术[10]对随机生成的模型进行后处理,使其符合 OCL 约束.

本文第 1 节介绍该算法支持的约束条件.第 2 节讨论该算法的技术细节.第 3 节对本文方法进行实验评估.第 4 节讨论相关的工作.最后是结论和未来工作.

1 模型生成的约束

在本文中,模型生成是一个根据语法约束(即元模型)和用户选项(配置参数)生成模型的过程.语法约束和用

户选项都可以看作是生成约束.然而,与基于求解器的方法不同,本方法并不会直接求解这些约束,而是利用这些约束来指导模型的生成过程.首先讨论本文方法所支持的约束.

1.1 元模型与模型

不失一般性,元模型$\mathcal{M}$可以被形式化地定义为

$$\mathcal{M}=(T,H,A,R,C,assoc,mult,\prec),$$

其中,

- T 是一个由类(class)构成的集合,其中,每一个元素代表一个类型;
- H 是一个抽象类的集合,且 $H\subset T$;
- A 是一个由属性构成的集合,其中,每一个属于 A 的属性 a 可以被定义为一个标签函数 $a:t_c\to d$,a 是属性的标识符,$t_c\in T$ 表示拥有该属性的类型,而 d 表示一个基本数据类型;
- R 是由引用组成的集合,其中,每一个引用(reference)都代表两种类型的元素之间的一种关联;
- C 表示聚合引用,且 $C\subseteq R$;
- $assoc$ 是一个函数,其定义是 $assoc:R\to T^2$.它将每个关联 $r\in R$ 映射到一个类型的有序对$\langle src,tar\rangle$上,指明了引用 r 的源和目标.如果 $assoc(r)=\langle src,tar\rangle$,我们定义符号 $r.source\equiv src$,$r.target\equiv tar$;
- $mult$ 也是一个函数,其定义是 $mult:R\to N_0^2\times N_0^2$.该函数用于刻画引用的多重性,其中,$N_0$ 是指所有的非负整数,包括+∞.对于每一个引用 $r\in R$,如果$(\langle ls,us\rangle,\langle lt,ut\rangle)=mult(r)$,则 ls 和 us 确定了源端多重性的下限和上限,而 lt 和 ut 则确定了目标端多重性的下限和上限;
- 最后,$\prec$表示类型之间的继承关系,它是一个定义在集合 T 上的偏序关系.如果 $c_1\prec c_2$,则表示 c_1 是 c_2 的一个子类.此外,本文还定义$\sqcap_c\equiv\{t|t\preccurlyeq c\}$,$\sqcup_c\equiv\{c|c\preccurlyeq t\}$.

需要指出的是,上述的定义是对 OCL 规范中对象模型定义的简化.不难验证,该定义兼容图语法的类型图和 Ecore(即 meta object facility(MOF)规范[11]的工业实现).

一个符合元模型$\mathcal{M}$语法约束的模型 M 可以被定义为

$$M=(N,E,type_N,type_E),$$

其中,N是模型元素集合;E表示关系集合;$type_N$是一个函数$type_N:N\to T$,它将模型元素映射到对应的类型上;$type_E$也是一个函数 $type_E:E\to R$,它将关系映射到引用上.

假设 $M=(N,E,type_N,type_E)$,则本文定义:

- 对于每个元素 e 和类型 t,$e\in_t M\Leftrightarrow e\in N\wedge type_N(e)=t$;
- 对于一个元素 e 和一个集合 G,$e\in_G M\Leftrightarrow e\in N\wedge type_N(e)\in G$;
- 对于每个元素 $e\in_t M$ 和属性 $a:t_c\to d$,符号 $a(e)$返回属性 a 对于 e 的取值;
- 对于任意两个元素 a 和 b 以及一个引用 r,$\langle a,b\rangle\in_r M\Leftrightarrow\langle a,b\rangle\in E\wedge type_E(\langle a,b\rangle)=r$.并且,我们称$\langle a,b\rangle$是一个“$r$-关系”.
- 对于任意两个元素 a 和 b 以及一个引用集合 G,$\langle a,b\rangle\in_G M\Leftrightarrow\langle a,b\rangle\in E\wedge type_E(\langle a,b\rangle)\in G$.

此外,在本文中,

$$r(x)\equiv\{\langle x,z\rangle|\langle x,z\rangle\in_r M\},r^{-1}(y)\equiv\{\langle z,y\rangle|\langle z,y\rangle\in_r M\}.$$

1.2 语法约束

最基本的约束就是从元模型中推导出的语法约束.生成的模型必须满足所有的语法约束.对于一个模型 M,它所对应的元模型$\mathcal{M}$规定了以下 3 种类型的语法约束.

- 元素语法约束:对于任意在模型 M 中的元素 e,它的类型必须是定义在$\mathcal{M}$上的一个非抽象类;
- 属性语法约束:对于任意元素 $e\in_{\sqcap_t} M$ 和它的属性 a(即 $a:t_c\to d$),$a(e)$的类型必须是 d;
- 关系语法约束:对于任意在模型 M 中的关系$\langle a,b\rangle$,它的类型必须是元模型中的一个关联,假设为 r.此

外,还必须满足多重性条件,即:对任意 $a\in_{\sqcap_{r.source}} M$ 和 $b\in_{\sqcap_{r.target}} M$, $lt\leqslant|r(a)|\leqslant ut\wedge ls\leqslant|r^{-1}(b)|\leqslant us$,其中,

$$(\langle ls,us\rangle,\langle lt,ut\rangle)=mult(r).$$

1.3 引用上的语义约束

除了语法约束之外,生成的模型还必须满足一些语义约束.语义约束多种多样,涉及到模型所刻画的具体领域.本文只关注定义在引用上的 4 种基本的语义约束,这 4 种基本的语义约束定义如下.

- 自反性:该约束决定了 r-关系的源端和目标端是否可以是同一个模型元素(在不考虑类型约束的情况下).更准确地说,如果 r 是非自反的,那么$\forall a(\langle a,a\rangle\notin_r M)$.在默认情况下,所有引用都是自反的;
- 有序性:该约束决定了引用 r 是否是一个偏序.当 r 是有序的,则对于任意$\langle e_n,e_0\rangle\notin_r M$,一定不存在 e_1, $e_2,\ldots,e_{n-1}$,使得 $\underset{0\leqslant i<n}{\wedge}\langle e_i,e_{i+1}\rangle\in_r M$.默认情况下,引用都是无序的;
- 必要性:该约束决定了一个元素是否必须参与到一个 r-关系中.它有两种具体的类型:源端必要性和目标端必要性.如果 r 是目标端必要的,则$\forall a(|r(a)|>0)$;如果 r 是源端必要的,则$\forall b(|r^{-1}(b)|>0)$.在默认情况下,引用既不是源端必要的,也不是目标端必要的;
- 唯一性:该约束决定了一个元素是否可以参与到多个 r-关系中.它也具有两种具体的类型:源端唯一性和目标端唯一性.如果 r 是目标端唯一的,则$\forall a(|r(a)|\leqslant 0)$;如果 r 是源端唯一的,则$\forall b(|r^{-1}(b)|\leqslant 1)$.默认情况下,引用既不是源端唯一的,也不是目标端唯一的.

上面 4 种约束可以扩展到引用集合 G 上.

- 自反性:如果 G 是自反的(或非自反的),其中的每个引用都是自反的(或非自反的);
- 有序性:如果 G 是有序的,则对于任意$\langle e_n,e_0\rangle\notin_G M$,一定不存在 $e_1,e_2,\ldots,e_{n-1}$,使得 $\underset{0\leqslant i<n}{\wedge}\langle e_i,e_{i+1}\rangle\in_G M$. 对任意两个元素 e_0 和 e_n, $e_0<_G e_n\Leftrightarrow\exists e_1,e_2,\ldots,e_{n-1}(\underset{0\leqslant i<n}{\wedge}\langle e_i,e_{i+1}\rangle\in_G M)$;
- 必要性:若 G 是目标端必要的,则有 $\forall a\left(\sum_{r\in G}|r(a)|>0\right)$; 若 G 是源端必要的,则有 $\forall b\left(\sum_{r\in G}|r^{-1}(b)|>0\right)$;
- 唯一性:若 G 是目标端唯一的,则有 $\forall a\left(\sum_{r\in G}|r(a)|\leqslant 1\right)$; 若 G 是源端唯一的,则有 $\forall b\left(\sum_{r\in G}|r^{-1}(b)|\leqslant 1\right)$.

例如:表示继承的引用必须是非自反的和有序的;所有的聚合引用所构成的集合是非自反、有序且源端唯一的;通常,聚合引用集合也必须是源端必要的,这是因为每个元素(根元素除外)都必须有一个容器.

1.4 范围约束

如前所述,本文提出的生成算法具有一定的可配置性.其可配置性体现在本文方法允许定义一些私有的范围约束来限制生成的模型,范围约束包括元素范围约束、关系范围约束和值范围约束.

对于在元模型中定义的一个类型,它的元素范围约束(ERC)规定了在最终生成的模型中可以存在多少个该类型的元素,每一个类型只能拥有一个 ERC.同时,我们还可以定义全局元素范围约束(GERC)用来限制模型中元素的总数量.ERC/GERC 都是一个整数类型的界(bound),界的定义将在下面介绍.

类似于 ERC,对于元模型中的每一个引用,我们可以为其定义一个关系范围约束(RRC).RRC 限制了模型中某种类型关系的数量,RRC 也是一个整数类型的界.

此外,对于每一个属性,我们还可以定义一个值范围约束(VRC),值范围约束用来限制该属性可能的取值.每个 VRC 都是某种基本数据类型(属性的类型)的界.

某种数据类型 d 的界 B_d 可以被定义为

$$B_d\subseteq V_d\times P \tag{1}$$

其中,V_d 是类型 d 所有的可能取值;P 是一个概率集合,P 中的每一个元素都是一个从 0 到 1 的实数.一个合法的界必须满足以下两点:

$$\sum_{(v,p)\in B_d} p=1\text{且}\forall b_1,b_2\in B_d(b_1\neq b_2\rightarrow v_1\neq v_2),$$

其中,$b_1=(v_1,p_1)$,$b_2=(v_2,p_2)$.每个属于 B_d 的$\langle v,p\rangle$的含义是"从$\{v|\langle v,p\rangle\in B_d\}$中选择 v 的概率为 p".

范围约束可以用来控制生成模型中各种类型的元素和关系的出现数量以及不同属性值的出现频率,从一个侧面保证生成模型的多样性和各种元模型要素的覆盖率.例如,要让模型中包含 30%的 A 元素和 40%的 B 元素,则可设定 A,B 元素的 ERC 分别为 0.3S 和 0.4S,其中,S 为模型中元素数量总和.如果希望 A 元素中某个布尔类型的属性 a 取值分布为"70%为 true,30%为 false",则可设置 a 的 VRC 为$\{\langle$true,0.7$\rangle$,$\langle$false,0.3$\rangle\}$.

2 生成算法

2.1 方法概览

本文的方法可以表示为图 1 所示的流程,其中包含 4 个主要步骤.

第 1 个阶段是配置生成参数.在这个过程中,需要建立一个配置模型,其中包含语义约束和用户自定义约束.用户自定义约束包括范围约束和额外约束.配置模型和元模型将被用于指导模型的生成;

第 2 个阶段是元素和属性的生成.基于语法和范围约束,本方法可以创建模型元素并设置该模型元素的属性值;

第 3 个阶段是生成所有的关系,它包含 3 个子步骤:(1) 实例化聚合引用(这些引用构成的集合是有序的、非自反的、源端必要且源端唯一的);(2) 根据配置模型定义的语义约束和范围约束,实例化被配置模型约束的引用;(3) 实例化普通类型的引用.尽管我们把关系生成过程分解为 3 个子步骤,但它们都使用了同样的算法实现,通过输入不同的参数来控制不同类型的关系生成;

最后一个阶段是验证阶段.所有的用户定义的额外约束都会在这个阶段被检查,因为我们在模型生成过程并没有求解这些约束.如果生成的模型满足了这些约束,本方法将返回该模型;否则将返回一个空模型,并报告错误.

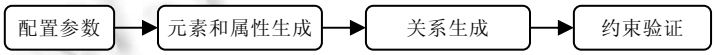

Fig.1 Approach overview

图 1 方法概览

2.2 配置生成过程

本节将讨论如何利用配置模型来配置模型生成过程.我们首先利用 MOF 来定义配置模型语法,如图 2 所示.

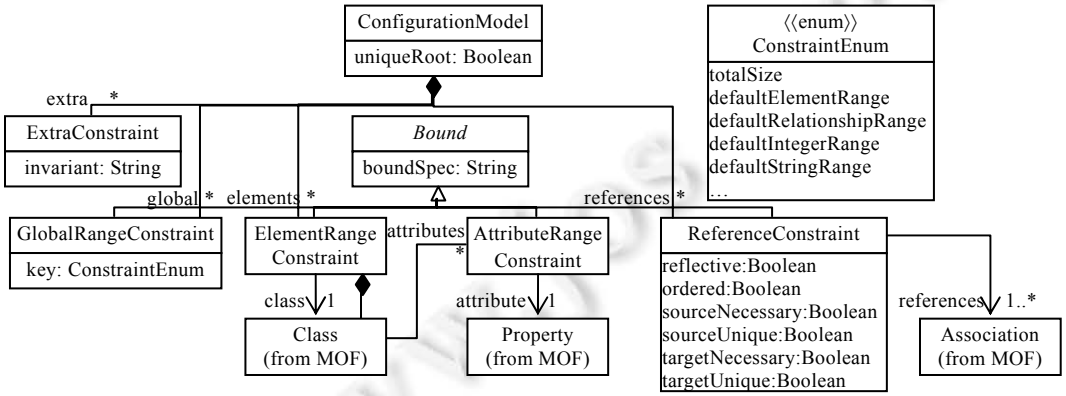

Fig.2 Definition of configuration model

图 2 配置模型

类 ConfigurationModel 表示一个配置,它指向一个 Class,表示了生成的模型中可以作为根元素的类型.它的属性 uniqueRoot 表示生成模型是否有唯一的根元素.一个配置模型 ConfigurationModel 可以包含 4 种类型的约

束:ElementRangeConstraint,RelationshipConstraint,GlobalRangeConstraint 和 ExtraConstraint.

一个 ElementRangeConstraint 对应一个 ERC.关系 class 表示该约束所限制的类型.需要注意的是:该约束限制了用指定类型创建的元素数量,而不包括用其子类创建的元素数量.每个 ElementRangeConstraint 还可能包含一组 AttributeRangeConstraint,一个 AttributeRangeConstraint 表示一个 VRC.关系 attribute 指明了被约束的属性.类 RelationshipConstraint 则被用于限制关系的生成过程,它结合了语义约束和关系范围约束.需要说明的是:聚合关系不能使用 RelationshipConstraint 进行显式约束,因为本文算法会自动处理聚合引用.

一个 GlobalRangeConstraint 可以用来规定范围的默认值,包括模型元素的总数(total size)、默认的元素范围(default element range)、默认的关系范围(default relationship range)以及默认的值范围(default value range).模型元素的总数(total size)刻画了生成的模型中元素的总个数.默认的元素范围(default element range)刻画了任意类型的实例数量.如果不存在针对某类型的 ElementRangeConstraint,这个默认值将被使用.默认的关系范围和默认的值范围与此类似.

ExtraConstraint 表示任何在本文中没有被讨论的约束,它们必须被写成 OCL 表达式的形式.出于生成效率的考虑,本文的方法不能求解这些约束,只是用它们验证生成的模型.

ElementRangeConstraint,AttributeRangeConstraint,RelationshipConstraint,GlobalRangeConstraint 都是 Bound 的子类,Bound 表示一个界,它可以表示成文字形式,其语法必须符合下面的定义.

bound::=bList
item::=bItem|pbItem
pbItem::=bItem ':'probability
bItem::=bList|bRange|literal
bList::='{'bList','item'}'
　　|'{'item'}'
bRange::='['literal'..'literal']'
literal 表示常量
probability 表示一个从 0 到 1 的实数

其中,bList 表示可能取值的列表,bRange 表示一个有上界和下界定义的范围.bItem 表示一个没有明确定义概率的项,可以是一个具体值、一个列表或一个范围,其概率可以由其他项推导出来.

用这种形式定义的界可以很容易地被转换成公式(1)所定义的逻辑形式.例如,{3,5:0.2,[7..9]:0.3}可以被转换成{⟨3,0.5⟩,⟨5,0.2⟩,⟨7,0.1⟩,⟨8,0.1⟩,⟨9,0.1⟩}.

2.3 生成元素和属性

第 2 阶段是生成模型中的元素和属性值.简单地说,对于元模型$\mathcal{M}$中的每个非抽象类 t,创建 s_t 个元素;其中,s_t 表示 t 元素的数量.基本的元素和属性生成算法可以写成算法 1 的形式.

算法 1. Generate elements and attribute values.

1: **for all** $t\in T-H$ **do** //t 为元模型中的非抽象类
2: $s_t\leftarrow$需要生成的 t-元素数量
3: **for** i=1 to s_t **do**
4: $e\leftarrow$创建一个 t-元素
5: 将 e 加入到模型中
6: **for all** $a:t\rightarrow t_a\in A$ **do**
7: $v\leftarrow$根据 a 的范围约束中规定的取值范围和概率所生成的值
8: $a(e)\leftarrow v$
9: **end for**

10: **end for**

11: **end for**

算法 1 的基本流程并不复杂,但困难的是为每个类 t 确定对应的 s_t.如果 s_t 没有被正确赋值,那么就可能无法生成合法模型.这是因为元模型和配置模型在生成的模型上强加了如下约束.

- 首先,令 $S_t=\sum_{o\in\sqcap_t} s_o$;
- 其次,对于每一个类,其实例数量由所有可能容器元素数量的取值决定.先定义引用之间的等价关系~,即 $r1\sim r2\Leftrightarrow\sqcap_{r1.target}\cap\sqcap_{r2.target}\neq\varnothing\vee\exists r3(r1\sim r3\wedge r3\sim r2)$.利用这一等价关系,我们可以将聚合引用集合 C 划分成若干等价类 $C_1,...,C_n$.那么对每个等价类 C_i,都需要满足下面的约束条件:

$$\sum_{r\in C_i}(lt_r\times S_{r.source})\leqslant\sum_{c\in CC_i}s_c\leqslant\sum_{r\in C_i}(ut_r\times S_{r.source}),$$

其中,对引用任意 r 假设有$(\langle ls_r,us_r\rangle,\langle lt_r,ut_r\rangle)=mult(r)$,且 $CC_i=\{c|r\in C_i\wedge c\in\sqcap_{r.target}\}$;

- 第三,任意两个类 s 和 t,S_s 和 S_t 同时还被非聚合引用 r 的多重性所约束,其中,$\langle s,t\rangle=assoc(r)$且$(\langle ls,us\rangle,\langle lt,ut\rangle)=mult(r)$,则有 $ut\times S_s\geqslant ls\times S_t\wedge us\times S_t\geqslant S_s$;
- 第四,对于每一个类 t 和 S_t 的取值必须属于 $range_t$,其中,$range_t$ 表示由 ElementRangeConstraint 或 default element range 所确定的界;
- 最后,如果定义了 total size 约束,则所有 S_t 的和必须符合该约束.

在获得这些约束后,我们可以利用一个约束求解器去求解这些范围约束,并获得一个随机解,以此来确定每一个类 t 所对应的 s_t 的取值.

2.4 生成关系

第 3 个阶段是生成模型 M 的关系,这个过程可以表述为算法 2.这种算法包含 3 个子过程:生成聚合关系(第 1 行)、生成受约束关系(第 2 行~第 4 行)、生成普通关系(第 5 行).所有这 3 个子过程都基于配置模型中定义的语义和范围约束.

算法 2. Entry of relationship generation.

1: 生成聚合关系

2: **for all** RelationshipConstraint *c* **do**

3: 生成受 *c* 约束的关系(即,类型属于 *c.references* 的关系)

4: **end for**

5: 产生其他关系

首先,我们讨论聚合关系的生成.如之前所述,所有的聚合关系都是非自反的、有序的、源端唯一的、源端必要的(除了根元素).所以,对于所有的聚合引用,我们都可以在算法 3 中进行处理.*parents* 是所有可能的容器元素集合,*children* 是所有子元素集合.注意,这两个集合可能相交.第 4 行收集所有可能的根元素(即 *possibleRoots*).第 5 行~第 7 行计算根元素的数量(即 *numberOfRoot*),这一步是基于 *uniqueRoot* 的值进行计算的.第 8 行~第 11 行从 *possibleRoots* 中随机选择 *numberOfRoot* 个元素作为根元素,并将其从 *children* 集合中移除,因为根元素不需要容器.实际的生成过程是通过调用 GenerateRelationships(第 13 行)实现的,其中,根据聚合关系的语义设置了参数.算法 GenerateRelationships 稍后再进行讨论.

算法 3. Generate containment relationships.

1: $N\leftarrow$模型 M 的元素集合

2: $parents\leftarrow N$

3: $RC\leftarrow$配置模型中指定的根元素类型

4: $possibleRoots\leftarrow\{e|e\in N\wedge\exists t(t\in RC\wedge type_E(e)\preccurlyeq t)\}$

5: **if** *config.uniqueRoot*=**true then** $numberOfRoot\leftarrow 1$

6: **else** *numberOfRoot*←1 到|*possibleRoots*|中的随机整数

7: **end if**

8: *roots*←从 *possibleRoots* 中随机选择的 *numberOfRoot* 个元素

9: *children*←*N*–*roots*

10: *size*←|*children*|

11: *GenerateRelationships*(*C*,*parents*,*children*,**true**,**true**,**false**,**false**,**false**,**true**,*size*)

接下来生成被 RelationshipConstraint 所约束的引用.由于 RelationshipConstraint 刻画了一组引用集合的语义和范围约束,简单地从中抽取相应的信息并将其作为 GenerateRelationships 的实际参数.这个过程如算法 4 所示.对于给定的一个 RelationshipConstraint,收集可能的源和目标元素(第 3 行和第 4 行),随机生成需要生成的关系数量(第 5 行).与算法 3 类似,调用 GenerateRelationships 使用提取的参数生成关系(第 6 行).

算法 4. *GenerateConstrainedRelationships*(*rc*).

Input: *rc* 是一个 RelationshipConstraint;

1: *N*←模型 *M* 的元素集合

2: *S*←*rc.references*

3: $src \leftarrow \{e | e \in N \wedge \exists r(r \in S \wedge type_E(e) \preccurlyeq r.source)\}$

4: $tar \leftarrow \{e | e \in N \wedge \exists r(r \in S \wedge type_E(e) \preccurlyeq r.target)\}$

5: *size*←根据 *rc* 中规定的关系范围约束随机选择的整数

6: *GenerateReferences*(*S*,*src*,*tar*,*rc.sourceUnique*,*rc.sourceNecessary*,*rc.targetUnique*,*rc.targetNecessary*,*rc.reflexive*,*rc.ordered*,*size*)

在第 3 个子阶段,根据默认的语义生成剩余的关系,如算法 5 所示.

算法 5. Generate reminder relationships.

1: *N*←模型 *M* 的元素集合

2: **for all** 未生成的引用 *r* **do**

3: $src \leftarrow \{e | e \in N \wedge type_E(e) \preccurlyeq r.source\}$

4: $tar \leftarrow \{e | e \in N \wedge type_E(e) \preccurlyeq r.target\}$

5: *size*←根据全局关系范围约束所选择的整数值(如果约束存在),或随机整数(如果约束不存在)

6: *GeneratedReferences*(*S*,*src*,*tar*,**false**,**false**,**false**,**false**,**true**,**false**,*size*)

7: **end for**

如前所述,核心的生成逻辑是由 GenerateRelationships 实现的.算法 GenerateRelationships 的基本思想可以表述为:(1) 随机选择一个引用 *r*;(2) 从集合 *N* 中随机选择一个源端元素 e_s 和一个目标端元素 e_t;(3) 建立一个 *r*-关系$\langle e_s,e_t\rangle$,并插入到模型 *M* 中.然而在这个基本过程中,我们必须处理如下两个问题:(1) 如何根据定义在 *r* 上的各种约束条件正确选择 e_s 和 e_t?(2) 如果无法选择到合适的 e_s 和 e_t,如何进行处理?

在继续之前,我们首先定义两个辅助函数 IsForbidden 和 IsCandidate.IsForbidden 用于检查两个元素 e_s 和 e_t,是否可以建立一个 *r*-关系而不违反自反性和有序性约束(见算法 6).IsCandidate 用于检查元素是否可以成为 *r*-关系的源端或目标端而不会违反多重性的上界以及必要性和唯一性约束.

算法 6. *IsForbidden*(*S*,e_s,e_t,*reflexive*,*ordered*).

Input:引用集合 *S*,两个模型元素 e_s 和 e_t,自反性和有序性约束 *reflexive* 及 *ordered*;

Output:当创建关系$\langle e_s,e_t\rangle$后,自反性和有序性约束 *reflexive* 及 *ordered* 是否会被违反.

1: **if** *reflexive*=**false and** e_s=e_t **then return true** //检查自反性

2: **else if** *ordered*=**true and** $e_t <_S e_s$ **and then return true** //检查有序性

3: **else return false end if**

算法 IsCandidate 有两个版本,即 IsSrcCandidate 和 IsTarCandidate.算法 7 展示了 IsSrcCandidate,它用来判断元素 e 是否可以用作 r-关系的源端.如果 e 满足了多重性的下界并且还有其他不满足下界的元素 e_t 存在,则 e 不能作为 r-关系的源端(第 3 行、第 5 行、第 6 行).这是因为算法赋予不满足下界的元素更高的优先级用来创建新关系,这是为了保证所有的元素都能满足下界.算法 IsTarCandidate 与之类似,只是它使用 $\sum_{r\in S}|r^{-1}(o)|$, $r^{-1}(o)$,ls_r 和 us_r 代替了 $\sum_{r\in S}|r(o)|$,$r(o)$,lt_r 和 ut_r 进行判断.

算法 7. *IsSrcCandidate*(*S*,*r*,*e*,*unique*,*necessary*).

Input:引用集合 S,引用 r,被检查的元素 e,必要性和唯一性约束 *necessary* 及 *unique*;

Output:元素 e 是否可以作为一个 r-relationship 的源点.

1: $(\langle ls_r,us_r\rangle,\langle lt_r,ut_r\rangle)\leftarrow mult(r)$

2: $U\leftarrow${所有候选元素} //每个候选元素都不违反上界和语义约束,并且都是有效的

3: $C_L\leftarrow\left\{o\mid o\in U\wedge\left(\left(unique\rightarrow\sum_{r\in S}|r(o)|=0\right)\wedge\left((r(o)<lt_r)\vee\left(necessary\wedge\sum_{r\in S}|r(o)|=0\right)\right)\right)\right\}$

4: $C_U\leftarrow\left\{o\mid o\in U\wedge\neg unique\wedge lt_r\leqslant\sum_{r\in S}|r(o)|\leqslant ut_r\wedge\neg\left(necessary\wedge\sum_{r\in S}|r(o)|=0\right)\right\}$

5: **if** $C_L\neq\varnothing$ **then return** C_L 中是否包含 e

6: **else if** $C_U\neq\varnothing$ **then return** C_U 中是否包含 e

7: **else return false end if**

现在讨论 GenerateRelationships 的细节,它可以描述为算法 8.

算法 8. *GenerateReferences*(*S*,*src*,*tar*,*srcUnique*,*srcNecessary*,*tarUnique*,*tarNecessary*,*reflexive*,*ordered*,*size*).

Input:引用集合 S,源端元素集合 *src*,目标端元素集合 *tar*,语义约束 *srcUnique*,*srcNecessary*,*tarUnique*,*tarNecessary*,*reflexive*,*ordered*,待生成的关系数量 *size*;

1: $G\leftarrow S$

2: $R\leftarrow$模型 M 的关系集合

3: **repeat**

4: **if** $G=\varnothing$ **then abort end if**

5: $r\leftarrow$从 G 随机选择的引用,且满足

$$\forall e(lt_r\leqslant|r(e)|\wedge ls_r\leqslant|r^{-1}(e)|)\rightarrow\neg\exists r'\exists e(|r'(e)|<lt_{r'}\vee|r'^{-1}(e)|<lt_{s'})$$

6: $e_s,e_t\leftarrow$从 *src* 和 *tar* 中随机选择的两个元素,且满足:*IsSrcCandidate*(*r*,e_s,*tarUnique*,*tarNecessary*)$\wedge$ *IsTarCandidate*(*r*,e_t,*srcUnique*,*srcNecessary*)$\wedge\neg$*IsForbidden*(*S*,e_s,e_t,*reflexive*,*ordered*)

7: **if** 这样的 e_s 和 e_t 存在 **then** $R\leftarrow R\cup\{\langle e_s,e_t\rangle\}$

8: **else if** 只有 e_s 存在 **then**

9: 寻找四元组$\langle r',x,y,z\rangle$,且满足:(1) $\langle x,y\rangle\in_{r'}M$;(2) 删除关系$\langle x,y\rangle$后,$\langle x,z\rangle$和$\langle e_s,y\rangle$都是合法的 r'-关系和 r-关系,即,*IsForbidden*(*S*,*x*,*z*,*reflexive*,*ordered*)和 *IsForbidden*(*S*,e_s,*y*,*reflexive*,*ordered*)都是 **false** 且 z 的多重性上界依然满足

10: **if** 如果四元组存在 **then** $R\leftarrow R-\{\langle x,y\rangle\}\cup\{\langle x,z\rangle,\langle e_s,y\rangle\}$

11: **else** 将 e_s 标记为 r 的无效源端候选元素

12: **end if**

13: **else if** 只有 e_t 存在 **then**

14: 寻找四元组$\langle r',x,y,z\rangle$,且满足:(1) $\langle x,y\rangle\in_{r'}M$;(2) 删除关系$\langle x,y\rangle$后,$\langle z,y\rangle$和$\langle x,e_t\rangle$都是合法的 r'-关系和 r-关系

15: **if** 如果四元组存在 **then** $R\leftarrow R-\{\langle x,y\rangle\}\cup\{\langle z,y\rangle,\langle x,e_t\rangle\}$

16: **else** 将 e_t 标记为 r 的无效目标端候选元素

17: **end if**

18: **else** $G \leftarrow G-\{r\}$

19: **end if**

20: **until** *src* 和 *tar* 中的所有元素都满足多重性下界 **and** (所有元素都达到多重性上界 **or** 已经产生了 *size* 个关系)

算法会不断生成新的关系,直到终止条件满足.在生成一个新关系时,它首先尝试选择两个模型元素来建立一个合法的 *r*-关系(第 6 行、第 7 行).在这个过程中,它会调用算法 6 和算法 7 来选择恰当的元素.这种选择策略能够保证多重性下限的约束被满足.例如,图 3(a)展示了一个包含两个类 *A* 和 *B* 以及一个从 *A* 到 *B* 的聚合关系 *r* 的元模型.该元模型要求每个 *A* 元素至少关联一个 *B* 元素.如图 3(b)所示,这是一个包含两个 *A* 和两个 *B* 的中间结果.模型中已经存在了一个 *r*-关系$\langle e_1,e_3\rangle$,现在我们尝试去创建第 2 个关系.按照算法第 6 行,只有 e_2 和 e_4 能够被选择成为新关系的源和目标.否则,如果选择 e_1 和 e_4 来创建关系,将得到一个错误的结果,因为没有其他的 *B* 元素可以与 e_2 关联,其多重性约束就会被破坏.

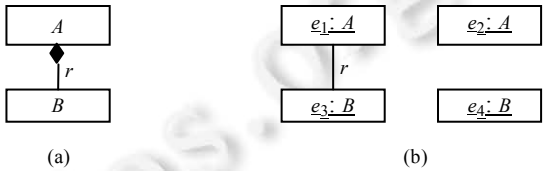

Fig.3 Example of element selection

图 3 元素选择的例子

如果只能找到源端元素 e_s(即,此时不存在任何 e_t 能够满足第 6 行的条件),则算法会尝试"修补"模型.修补模型是通过删除一个已经存在的关系并加入两个新关系来实现的,整个过程不会违反任何多重性、范围约束和语义约束(第 9 行、第 10 行).如果模型无法通过修补过程来增加新的关系,则将 e_s 标记为 *r* 的无效源端(第 11 行),以后再创建 *r*-关系时,e_s 不会再被选择.如果只能找到目标端元素 e_t,处理方式与之类似(第 14 行~第 17 行).修复过程是希望通过调整现有模型中的一个关系来向模型增加新的关系.例如,如图 4(a)所示的元模型,其中定义了 3 个类和 2 个聚合关系.元模型要求 *A* 元素必须关联一个 *B* 元素和一个 *C* 元素.图 4(b)展示了一个中间结果,其中包含 3 个元素和 1 个 *r*-关系$\langle e_1,e_3\rangle$.现在我们尝试添加一个 *r*′-关系.然而,由于类型的限制,e_2 不能参与到这样一个关系中(用虚线表示).那么根据修复逻辑,我们可以删除$\langle e_1,e_3\rangle$,然后向模型中添加新的 *r*-关系$\langle e_1,e_2\rangle$和一个新的 *r*′-关系$\langle e_1,e_3\rangle$,最终结果如图 4(c)所示.

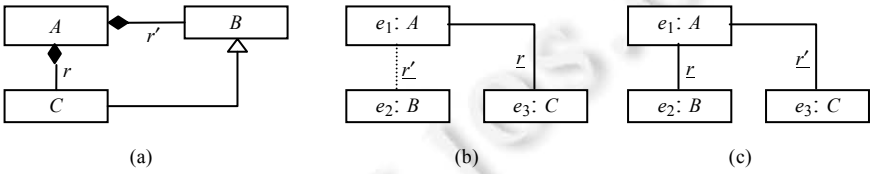

Fig.4 Example of model fixing

图 4 模型修复的例子

如果不能为 *r* 找到任何源端和目标端,就将 *r* 从集合 *G* 中删除(第 18 行).

这个算法的终止条件是下面两个条件之一:(1) *G* 是空集;(2) 在所有元素都达到多重性下界的情况下,所有元素都达到了多重性的上界(不能再生成任何关系了)或者已经生成了 *size* 个关系.该算法一定会终止,因为在每一个循环中,我们至少创建了一个新关系,删除了一个候选元素,或者删除了一个候选引用.因此,终止条件一定会被满足.

2.5 约束验证

最后一个步骤是对生成的模型进行约束验证,此阶段会对所有的语法、语义和用户定义的约束进行检查.如果生成的模型违反了任何约束,生成的模型将会被拒绝,并且报告给用户一个错误的信息.由于我们的方法能够保证语法和语义约束不被破坏(我们会在后续进行讨论),最后可能被破坏的约束就是用户定义的约束了,特别是复杂的额外约束.但是由于我们的方法主要用于实现黑盒测试和性能测试,这种情况并不多见.所以,绝大多数情况下,我们的方法都能生成一个有效的输出.

2.6 讨 论

1) 复杂性

该算法的复杂度是多项式的.假设基于元模型$\mathcal{M}=(T,H,A,R,C,assoc,mult,\prec)$生成一个模型 $M=(N,E,type_N,type_E)$.容易得出:生成元素和属性(算法 1)的复杂度是 $O(|N|\times|A|)$,生成关系的复杂度是 $O(|E|+|N|\times|R|)\times(O(|E|)\times O(|N|^3))$.这是因为:(1) 算法 8 的循环体最多被执行的次数是 $O(|E|+|N|\times|R|)$;(2) 算法 6 的复杂度是 $O(|N|^2)$;(3) 选择两个候选元素(算法 8 的第 6 行)的复杂度是 $O(|N|\times|N|^2)$;(4) 寻找$\langle r',x,y,z\rangle$(算法 8 的第 10 行和第 17 行)的复杂度是 $O(|E|\times|N|\times|N|^2)=O(|E|)\times O(|N|^3)$.在实现中,采用了缓存机制来减小时间开销.

2) 正确性

当不存在冲突的时候,我们的方法能够生成一个符合语法约束、语义约束和范围约束的模型.严格地证明这一点是很复杂的,我们只能定性地讨论一下.

- 首先,我们的方法基于元模型生成元素、属性和关系,因此,生成的模型满足语法约束(多重性约束在后面进行讨论);
- 其次,当生成关系时,我们的方法依然把语义约束考虑在内.在算法 8 的第 6 行,被选择的用于生成新关系的元素必须满足 4 种语义约束.并不难证明,修补过程(算法 8 中第 8 行~第 12 行及第 13 行~第 17 行)也不会违反任何语义约束.这是因为在追加一个新的关系之前,我们调用了 IsForbidden 和 IsCandidate 函数检查语义约束;
- 最后,算法 1 和算法 8 只有当所有的范围约束都满足时才会终止.

本文的方法能够保证生成所需数量的元素,但并不保证一定会创建用户要求数量的关系.如果用户规定的数量过小或者过大,或者其他约束之间存在冲突时,算法所生成的关系数量都可能不符合用户的要求.在后续工作中,我们将研究如何消解生成过程中发现的约束冲突.

值得注意的是,在执行修补的过程中将不会违反任何范围约束.假设我们发现了$\langle r',x,y,z\rangle$,如果 y 的度(即$|r'^{-1}(y)|$)在$\langle x,y\rangle$被移除后违反了它的下界,y 在后面就需要与其他的x'关联.这等同于在没有任何删减的情况下直接关联 e_s 和 y.

3) 随机性和元模型覆盖率

本文提出的算法是随机算法,因为其中所有选择操作都是随机的,比如在创建关系时,选择何种类型的关系进行创建、在哪两个元素之间进行创建都是随机决定的,因此算法所生成的模型也是随机的.这最大限度地减小了利用相同配置所生成的多个模型之间的相似性,保证了数据的多样性.

此外,由于支持范围约束,本文方法还能保证不同类型的元素和关系以及不同属性值的出现数量和频率.

- 一方面,可以利用元素范围约束和关系范围约束规定每种类型的元素和关系出现的上下界.算法能够根据这个上下界,在模型中生成特定数量的元素和关系(在没有冲突的情况下);
- 另一方面,利用值范围约束可以规定某个属性的不同取值的出现频率.

这都可以保证元模型中关键要素的覆盖率.

4) OCL 约束

本文算法生成的模型能够符合元模型所规定的语法约束和一部分基本语义约束,但对于复杂的语义约束不能直接处理.如前所述,这样做的原因有 3 个:(1) 不同于白盒测试,性能测试中对于 OCL 约束的使用相对较少,

输入数据的规模更加重要;(2) 在生成模型时考虑 OCL 约束将极大地影响生成效率,不利于构造大模型;(3) 对于某些模型转换,违反 OCL 约束的模型可能才是有效的输入.正因如此,虽然忽略了 OCL 约束,但本文方法依然能够在一定程度上满足模型转换性能测试的需要.

实际上,本文方法可以用作生成大规模输入模型的基础方法——在此基础上进行扩展,可以构造出符合复杂 OCL 约束的模型.理论上,只要用本文方法构造出足够数量的随机模型,再利用 OCL 约束进行筛选,则可以从中选择出满足条件的模型.实践中,可以设计出相应的约束修复程序,在生成的模型中寻找并修复不符合约束的地方,从而将随机模型修正为符合 OCL 约束的输入模型.这些扩展技术将是本文下一步的工作.

3 评 估

本节首先用一个例子展示如何利用本文的方法生成模型,并对模型转换的性能进行测试.之后,通过一个实验评估本文方法的随机性,并利用两个实验检验本文方法的效率.

为了检验本文方法的效率,我们进行了两个实验.在第 1 个实验中,我们比较了 EMFtoCSP[12],Alloy[13]和本文方法的执行效率.EMFtoCSP 和 Alloy 都是目前最好的模型约束求解器,有很多研究工作都是基于它们来实现测试输入生成[14,15].在第 2 个实验中,我们检验了本文方法生成大模型时的效率.两个实验的执行环境都是:一台配有 Intel i7 4770 CPU,16GB DDR3 内存,Windows 7 的电脑.本文并没有与其他基于算法的方法进行比较,这些方法的局限性已在文献[16]中被充分地讨论过,本文也会在第 4 节展开,进一步加以讨论.

3.1 应用举例

在之前发表的会议论文[9]中,展示了一个用本文方法对 ATL 转换 JavaSource2Table 进行性能测试的例子.

本文从 ATL 转换数据库(http://www.eclipse.org/atl/atlTransformations/)选择了另一个开源转换程序 MySQL2KM3,用本文方法对其进行性能测试.MySQL2KM3 能够将基于 MySQL 的关系数据库模型转换成 KM3 模型.MySQL 元模型如图 5 所示.

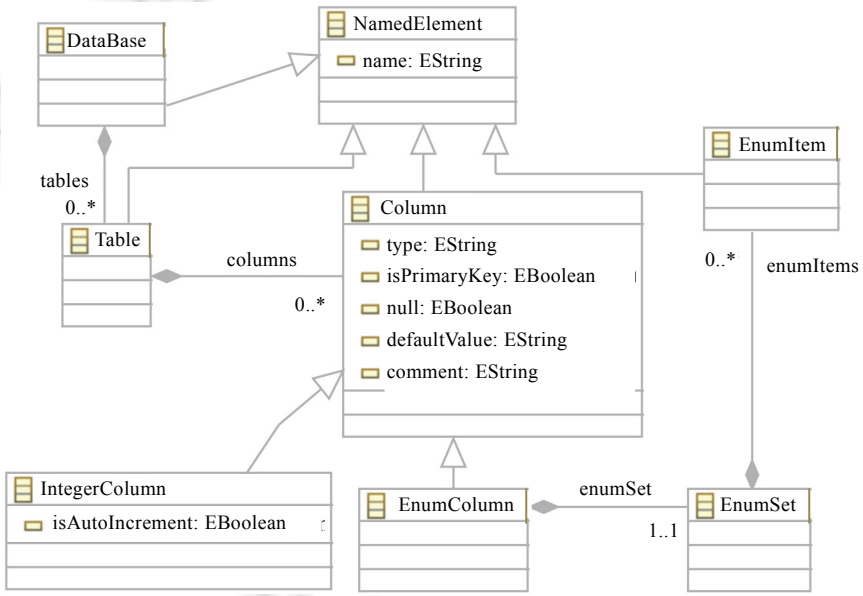

Fig.5 MySQL metamodel used in MySQL2KM3

图 5 MySQL2KM3 中使用的 MySQL 元模型

为了产生输入模型,首先根据第 2.2 节定义的配置元模型建立模型生成的配置模型,定义每种类型的元素和关系的生成数量以及每种属性的取值范围.为了避免产生过多的主键,将 Column 元素 isPrimaryKey 属性的取

值设定为 10%的概率为 true,90%的概率为 false.又通过对 MySQL2KM3 的源码分析发现:Column 元素的 type 属性如果取值是 tinyint,int,float,double,varchar,enum,则可以触发相应的转换规则;反之,则不能触发转换规则.为了保证对应规则都被触发,同时模拟无效输入,在配置模型中规定,type 属性只能从上述字面值和 blob(代表无效输入)中随机选取.同时,EnumColumn 代表枚举类型,其 type 属性显然必须取值为 enum.因此在配置模型中,规定 EnumColumn 的 type 属性必须取值为 enum,即,覆盖了上一条针对 type 属性的范围约束.最后,由于转换中没有一条规则处理 IntegerColumn,因此将其数量设置为 0.

根据配置模型,生成了 10 组输入模型,每组输入模型包含 5 个模型.第 1 组模型包含 500 个模型元素,之后,每组模型中元素数量等比例地增加,直到第 10 组模型包括 5 000 个模型元素.整个生成过程在 10 分钟内完成.依次将这 50 个模型作为转换 MySQL2KM3 的输入,执行转换获得执行时间,并针对每组输入求平均值,最终获得的结果见表 1.可以看出,本文方法能够很好地支持对模型转换进行性能测试.

Table 1 Execution time of MySQL2KM3

表 1 MySQL2KM3 的执行时间

Input size (# elements)	500	1 000	1 500	2 000	2 500	3 000	3 500	4 000	4 500	5 000
Time cost (s)	0.17	0.61	1.58	2.50	3.82	5.20	7.27	9.35	11.80	14.42

我们在进行性能测试的过程中还发现,MySQL2KM3 并不是一个全函数.针对某些满足特定条件的 Table 元素无法进行转换,从而导致执行过程中产生警告信息(但并不影响执行时间和结果).这是因为,在 MySQL2KM3 中存在 3 条规则用于转换 Table 元素,每条规则规定了一种前置条件.当 Table 元素不满足这 3 个前置条件但又被要求进行转换时,ATL 引擎就会发出警告,因为没有规则能够处理这样的 Table 元素.设计良好的转换程序应该能够避免这种警告的产生,例如,在要求转换 Table 元素之前先判断是否能够进行转换:如果不能转换,则应该使用默认值(如 null);反之,则调用转换规则并使用转换结果.这说明,本文方法还能被应用于黑盒测试的场景,并发现转换程序中存在的缺陷.如何利用本文方法支持对模型转换的正确性进行黑盒测试,也是本文未来的工作之一.

3.2 实验1:随机性

本文方法是一种随机方法,同样的配置下能够产生一组具有多样性的模型,从而可以用于计算转换程序的平均执行时间.为了评估本文方法产生模型的随机性,利用第 3.1 节中产生的 50 个模型计算它们之间的相似性.将每一组的 5 个模型进行两两比较,并获得每组模型的平均相似度.相似度的计算公式如下:

$$\frac{\text{匹配的元素数}}{\text{全部的元素数}}.$$

本文利用模型比较工具 EMF Compare(https://www.eclipse.org/emf/compare/)计算两个模型中的匹配元素.为了避免任何主观偏向,本文采用默认配置进行模型比较,最后统计匹配元素数量并计算相似度.对于每组模型的平均相似度见表 2.

Table 2 Model similarity

表 2 模型相似度

Model size (# elements)	500	1 000	1 500	2 000	2 500	3 000	3 500	4 000	4 500	5 000
Similarity (%)	18.20	19.93	21.71	22.77	23.13	23.80	24.17	24.54	25.34	25.55

从表 2 中可以看出:针对 MySQL 元模型,本文方法产生的每组模型的平均相似度可以控制在 26%以下.即:按照 EMF Compare 进行计算,任意两个同组模型之间至多有 26%的元素是相似的.通过检查这些相似元素可以发现,主要是 Column 元素和 EnumColumn 元素之间存在较多的匹配元素.由于 EMF Compare 判断两个元素是否相似的默认策略是计算元素属性之间的编辑距离,而 Column/EnumColumn 元素包含较多的属性且存在多个布尔类型属性,这就导致两个 Column 元素之间存在较大的相似机率.模型元素数量越多,匹配的 Column 元素也就越多,因此相似度在不断增加.根据函数拟合的结果,当模型元素达到 10^6 时,模型相似度达到 50%(即,大约为

Column 元素所占比例).需要指出的是:这种类型的相似元素是不可避免的,但对于计算转换程序的平均执行开销并无负面影响.

3.3 实验2:执行效率比较

首先,我们从 ATL 转换数据库中任选了两个元模型 JavaSource 和 PetriNet 用于测试.其中,JavaSource 是一个简单 Java 源文件元模型,PetriNet 则是一个 Petri 网元模型.

然后,被测试的 3 种方法都要被要求根据 JavaSource 元模型和相同的参数生成同样大小的模型.生成过程没有设置额外的约束条件.我们用每种方法生成 10 个模型,第 1 个模型有 150 个模型元素,第 2 个模型具有 300 个元素等(按此规律线性增长).每个模型生成 5 次,以便获取平均执行时间.实验结果见表 3.

Table 3 Time costs of generating JavaSource models

表 3 实例化 JavaSource 的时间开销

	1	2	3	4	5	6	7	8	9	10
Ours	0.06	0.12	0.14	0.18	0.18	0.9	0.20	0.23	0.24	0.26
Allo	0.12	0.77	2.33	5.23	9.68	17.38	29.60	45.04	61.60	73.33
EMFC	72.79	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

表 3 中的每个单元都展示了一种方法(行)在生成一个模型(列)时所用的平均时间(单位:s).在实验中,我们的方法能够在 0.06s~0.26s 之间返回结果;Alloy 则能在 0.12s~45.04s 之间生成结果;而对于 EMFtoCSP,它在生成第 1 个模型时使用了 72.79s,之后都没有在有效时间内(20min)生成输出.我们的方法和 Alloy 方法的性能曲线如图 6 所示.

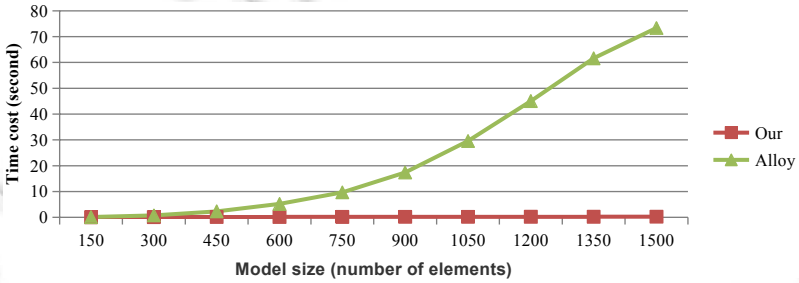

Fig.6 Performance curve of generating JavaSource models

图 6 实例化 JavaSource 模型的性能曲线

为了进一步比较 Alloy 方法和本文的方法,我们又利用 PetriNet 元模型进行了 2 次实验.同样,我们利用这两种方法生成 10 个模型,每个模型生成 5 次以便获得平均值.其中,第 1 个模型具有 600 个元素,第 2 个模型具有 1 200 个元素等.实现结果见表 4.其中,我们的方法可以在 0.06s~1.76s 之间返回结果;Alloy 方法在生成第 1 个模型时用时 226.87s,在生成第 2 个模型时已经无法返回结果了.

Table 4 Time costs of generating PetriNet models

表 4 实例化 PetriNet 的时间开销

	1	2	3	4	5	6	7	8	9	10
Ours	0.06	0.11	0.21	0.32	0.47	0.65	0.88	1.15	1.41	1.76
Allo	226.87	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

这个实验能够证明本文方法比其他两种方法的效率更高.此外,在实验中我们还注意到:本文的方法能够随机生成模型,而 Alloy 和 EMFtoCSP 则不能.这也就是说:在相同的配置下,本文的方法每次生成的模型都能和之前生成的不同;而对于 Alloy 和 EMFtoCSP,如果我们不改变约束条件,它们总会生成同一个模型.这是因为,它们所使用的约束求解器会按照确定的方法遍历状态空间(随机搜索的效率会更低).

3.4 实验3:可伸缩性

在第 3 个实验中,我们利用本文的方法分别为 5 个元模型生成一组模型(不包括其他约束).对于每一个元模型,我们会生成 5 个模型,其文件大小从 1MB 到 5MB 发生变化.对于每一个模型,我们都生成 5 次以便获取平均执行时间(单位:s).这 5 个元模型分别是 JavaSource,extlibrary,BibTex,PetriNet 和 TextualPathExp.其中,extLibrary 是 EMF 工具中的标准案例项目 Extended Library Model Example 中使用的元模型,JavaSource 和 PetriNet 与实验 1 中使用的元模型相同,它们与 BibTex,TextualPathExp 一起都来自 ATL 转换数据库.

对于每一个模型,为了控制模型的大小,其元素和关系数量都是固定值.对于每个元模型下相应的 5 个模型,元素范围约束和关系范围约束的变化比例是一致的.例如,假设生成第 1 个模型 M_1 时,我们使用了两个范围约束 ra_1 和 rb_1.在生成第 2 个模型 M_2 时,如果 $k{\cdot}ra_1{=}ra_2$,那么 $rb_2{=}k{\cdot}rb_1$.实验 3 的结果见表 5.

Table 5 Result of experiment 2 (s)

表 5 实验 2 的结果 (秒)

Size	1MB	2MB	3MB	4MB	5MB
JavaSource	3.94	13.84	34.49	63.68	98.05
extlibrary	1.94	7.26	16.36	29.12	39.86
BibTex	5.74	27.50	74.42	141.48	214.44
PetriNet	6.56	18.28	41.63	75.31	119.4
TextualPathExp	71.86	320	812.09	1 776.68	2 800.35

对于 JavaSource 模型,消耗的时间在 3.94s~98.05s 之间.对于 extlibrary 模型,消耗的时间在 1.94s~39.86s 之间.对于 BibTex 模型,消耗的时间在 5.74s~214.44s 之间.对于 PetriNet 模型,消耗的时间在 6.56s~119.4s 之间.对于 TextualPathExp 模型,消耗的时间在 71.56s~2800.35s 之间.

实验 2 的性能曲线如图 7 所示.在图 7 中,我们还列出了每条曲线拟合的函数.对于 JavaSource 模型,其性能曲线服从函数 $y{=}5.789x^{2.2792}$;对于 extlibrary 模型,性能曲线服从函数 $y{=}3.7502x^{2.0197}$;对于 BibTex 模型,性能曲线服从函数 $y{=}1.9556x^{1.9096}$;对于 PetriNet 模型,性能曲线服从函数 $y{=}5.9807x^{1.8124}$;对于 TextualPathExp 模型,性能曲线服从函数 $y{=}68.936x^{2.297}$.

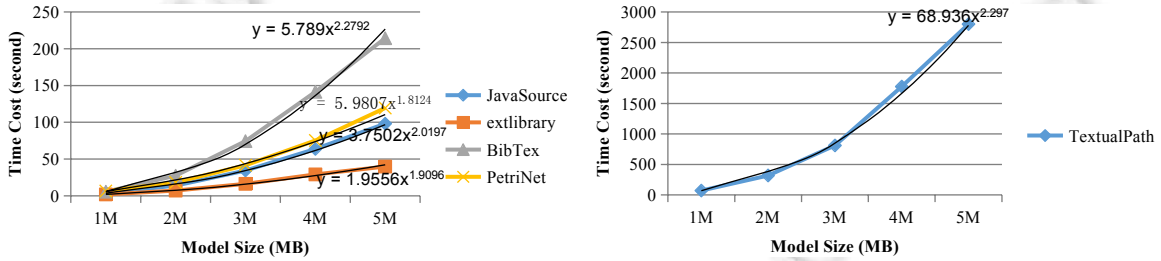

Fig.7 Performance curves of experiment 2

图 7 实验 2 的性能曲线

从这个实验中,我们可以得到下面 3 个结论.

1) 当模型的大小呈线性增长时,消耗的时间服从幂函数 $y{=}k{\times}x^p$;

2) 指数 p 的变化范围在 1.8124~2.297 之间,其平均近似值为 2.06.对于不同元模型,指数变化不大;

3) 对于不同的元模型,系数 k 剧烈变化,这就意味着,系数 k 受元模型复杂性影响较大.

理论上讲,本文方法的性能也会受到要生成的模型复杂性的影响.在今后的工作中,将研究本文方法性能和元模型的复杂性以及模型之间的关系.

4 相关工作

目前存在一些关于模型生成(或元模型实例生成)的研究工作,可以分为 3 种技术路线:基于求解的方法、基

于算法的方法和基于图形语法的方法.前两种方法在本文开始部分已经提到.本节将比较已有的研究工作.

基于求解器方法的基本思想如下:(1) 将元模型和约束转换成一组不变式,这些不变式可以被模型检查器、约束求解器或者 SAT/SMT 求解器所接受;(2) 使用上述求解器求解不变式,并获得一个解;(3) 把得到的解转换成模型.

Alloy[13]是一个成熟的针对关系模型证明器.Anastasakis 等人讨论了如何把一个 UML 类图转换成 Alloy 语言[17],他们提出一些转换规则,能够把 UML 类图和部分 OCL 约束映射到 Alloy 中.Sen 等人提出了一个基于 Alloy 的工具[14],该工具能够生成可以测试模型转换的输入模型.McQuillan 等人提出一种用于度量面向对象系统的元模型[18].用户可以使用该元模型定义一些度量指标(即一个度量模型),之后这些度量指标可以被转换成 Alloy 语言,并检测度量指标的有效性(如果有效,Alloy 可以找到一个符合该度量模型的实例).

Apt[19]等人提出了一种模型约束求解器——ECL^{i}PS^s.Cabot 等人[20]提出了一种将 OCL 约束下的 UML 模型转换成 ECL^{i}PS^s 的方法.他们的方法都基于 EMFtoCSP[12],并用于生成模型转换的测试输入[15].

Soeken 等人[21,22]也提出一种使用 SAT 求解器验证 OCL 约束下的 UML 模型的方法,他们使用 bit-vector(位向量)来编码元模型和约束.

不难看出,基于求解器的方法比本文的方法更加灵活,因为它们可以处理更多的约束条件.它们适用于模型验证和白盒测试.然而在第 3 节中我们已经通过实验证明:与本文的方法相比,基于求解器的方法不能高效地生成较大规模的模型.这使得它们无法很好地支持模型转换的性能测试.

除此之外,还有很多基于算法的方法.

Mougenot 等人[23]提出了一种基于 Boltzmann 算法[24]的大模型生成器.首先,他们将元模型转换成树的规约,其中,类是节点,聚合引用是边.该方法还支持聚合引用的多重性约束.之后,他们调用 Boltzmann 方法生成一个符合规范的有效树.该方法执行效率很高,但却没有讨论如何生成非聚合关系.此外,该方法也没有考虑语义约束.因此,他们的方法可能生成一个无效的模型,例如一个包含循环继承的类图.

Brottier 等人[25]提出了一种用于测试模型转换的模型生成算法.该算法能够依照一定的覆盖准则和组合策略,将一组给定模型片段复合成完整的模型.但他们没有讨论如何生成初始的模型片段.相比于该工作,本文的方法更具有可配置性,考虑到了更多的约束条件,从而确保了生成模型的正确性.

Ehrig 等人介绍了一种基于图语法的模型生成方法[26].该方法将元模型和约束编码成一组图转换规则,并通过执行这些规则生成一个模型.然而,在图转换中最常进行的操作是图模式匹配,这是一个 NP 完全问题,也是图转换的性能瓶颈.因此,这种基于图语法的模型生成方法也不可能高效地生成大规模的模型.

5 总结与展望

本文主要工作包括:(1) 提出了一种随机、高效、正确、可配置的模型生成算法,在合理的时间内能够生成符合语法、语义和范围约束的模型;(2) 进行了两个实验,验证了本方法的高效性.

在未来的工作中,我们计划从以下两个方面改进本文的方法:(1) 尝试处理更多种类的约束(尤其是语义约束);(2) 探索并行模型生成算法,以便更好地利用多核计算机.此外,我们还会进行更加系统的实验评估,以便分析本文方法的性能与元模型的复杂度以及将要生成的模型之间的精确关系.

此外,在本文方法的基础上,我们还将研究如何基于随机模型生成技术构造更加复杂但高效的模型生成方法,以便支持复杂的 OCL 约束.同时,还将探索如何利用本文方法实现对模型转换的随机测试.

References:

[1] Zhang J, Lin Y, Gray J. Generic and domain-specific model refactoring using a model transformation engine. In: Beydeda S, Book M, Gruhn V, eds. Proc. of the Model-Driven Software Development. Berlin: Springer-Verlag, 2005. 199−217. [doi: 10.1007/3-540-28554-7_9]

[2] Straeten R, Jonckers V, Mens T. A formal approach to model refactoring and model refinement. Software & Systems Modeling, 2007,6(2):139−162. [doi: 10.1007/s10270-006-0025-9]

[3] Hemel Z, Kats LC, Visser E. Code generation by model transformation. In: Vallecillo A, Gray J, Pierantonio A, eds. Proc. of the Theory and Practice of Model Transformations. Berlin: Springer-Verlag, 2008. 183−198. [doi: 10.1007/978-3-540-69927-9_13]

[4] Prout A, Atlee JM, Day NA, Shaker P. Semantically configurable code generation. In: Czarnecki K, Ober I, Bruel JM, Uhl A, Völter M, eds. Proc. of the Model Driven Engineering Languages and Systems. Berlin: Springer-Verlag, 2008. 705−720. [doi: 10.1007/978-3-540-87875-9_49]

[5] Xiong Y, Liu D, Hu Z, Zhao H, Takeichi M, Mei H. Towards automatic model synchronization from model transformations. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2007. 164−173. [doi: 10.1145/1321631.1321657]

[6] Hermann F, Ehrig H, Orejas F, Czarnecki K, Diskin Z, Xiong Y. Correctness of model synchronization based on triple graph grammars. In: Whittle J, Clark T, Kühne T, eds. Proc. of the Model Driven Engineering Languages and Systems. Berlin: Springer-Verlag, 2011. 668−682. [doi: 10.1007/978-3-642-24485-8_49]

[7] Song H, Huang G, Chauvel F, Xiong Y, Hu Z, Sun Y, Mei H. Supporting runtime software architecture: A bidirectional-transformation-based approach. Journal of Systems and Software, 2011,84(5):711−723. [doi: 10.1016/j.jss.2010.12.009]

[8] Object Management Group. OMG object constraint language (OCL) specification, Version 2.3.1. 2012. http://www.omg.org/spec/OCL/2.3.1

[9] He X, Zhang T, Ma ZY, Shao WZ. Randomized model generation for performance testing of model transformations. In: Chang CK, Gao Y, Hurson A, Matskin M, McMillin B, Okabe Y, Seceleanu C, Yoshida K, eds. Proc. of the Annual Int'l Computer, Software & Applications Conf. IEEE Computer Society, 2014. 11−20. [doi: 10.1109/COMPSAC.2014.103]

[10] Xiong Y, Hu Z, Zhao H, Song H, Takeichi M, Mei H. Supporting automatic model inconsistency fixing. In: Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. New York: ACM Press, 2009. 315−324. [doi: 10.1145/1595696.1595757]

[11] Object Management Group. OMG meta object facility (MOF) core specification, Version 2.4.1. 2011. http://www.omg.org/spec/MOF/2.4.1

[12] Pérez CAG, Buettner F, Clarisó R, Cabot J. EMFtoCSP: A tool for the lightweight verification of EMF models. In: Proc. of the Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA). New York: IEEE, 2012. 44−50. [doi: 10.1109/FormSERA.2012.6229788]

[13] Jackson D, Schechter I, Shlyakhter I. Alcoa: The Alloy constraint analyzer. In: Proc. of the Int'l Conf. on Software Engineering. New York: ACM Press, 2000. 730−733. [doi: 10.1109/ICSE.2000.870482]

[14] Sen S, Baudry B, Mottu JM. On combining multi-formalism knowledge to select models for model transformation testing. In: Proc. of the Int'l Conf. on Software Testing, Verification, and Validation. New York: IEEE, 2008. 328−337. [doi: 10.1109/ICST.2008.62]

[15] González CA, Cabot J. Atltest: A white-box test generation approach for ATL transformations. In: France RB, Kazmeier J, Breu R, Atkinson C, eds. Proc. of the Model Driven Engineering Languages and Systems. Berlin: Springer-Verlag, 2012. 449−464. [doi: 10.1007/978-3-642-33666-9_29]

[16] Wu H, Monahan R, Power JF. Metamodel instance generation: A systematic literature review. Technical Report, arXiv: 1211.6322, 2012. http://arxiv.org/abs/1211.6322v2

[17] Anastasakis K, Bordbar B, Georg G, Ray I. On challenges of model transformation from UML to Alloy. Software & Systems Modeling, 2010,9(1):69−86. [doi: 10.1007/s10270-008-0110-3]

[18] McQuillan JA, Power JF. A metamodel for the measurement of object-oriented systems: An analysis using Alloy. In: Proc. of the Int'l Conf. on Software Testing, Verification, and Validation. New York: IEEE, 2008. 288−297. [doi: 10.1109/ICST.2008.58]

[19] Apt KR, Wallace M. Constraint Logic Programming Using ECLiPSe. New York: Cambridge University Press, 2007.

[20] Cabot J, Clarisó R, Riera D. Verification of UML/OCL class diagrams using constraint programming. In: Proc. of the Int'l Conf. on Software Testing Verification and Validation Workshop. New York: IEEE, 2008. 73−80. [doi: 10.1109/ICSTW.2008.54]

[21] Soeken M, Wille R, Kuhlmann M, Gogolla M, Drechsler R. Verifying uml/ocl models using boolean satisfiability. In: Preas K, ed. Proc. of the Design, Automation & Test in Europe Conf. & Exhibition (DATE). New York: IEEE, 2010. 1341−1344. [doi: 10.1109/DATE.2010.5457017]

[22] Soeken M, Wille R, Drechsler R. Encoding OCL data types for SAT-based verification of UML/OCL models. In: Gogolla M, Wolff B, eds. Proc. of the Tests and Proofs. Berlin: Springer-Verlag, 2011. 152−170. [doi: 10.1007/978-3-642-21768-5_12]

[23] Mougenot A, Darrasse A, Blanc X, Soria M. Uniform random generation of huge metamodel instances. In: Paige RF, Hartman A, Rensink A, eds. Proc. of the Model Driven Architecture-Foundations and Applications. Berlin: Springer-Verlag, 2009. 130−145. [doi: 10.1007/978-3-642-02674-4_10]

[24] Duchon P, Flajolet P, Louchard G, Schaeffer G. Boltzmann samplers for the random generation of combinatorial structures. In: Proc. of the Combinatorics, Probability and Computing. New York: Cambridge University Press, 2004. 577−625. [doi: 10.1017/S0963548304006315]

[25] Brottier E, Fleurey F, Steel J, Baudry B, Le Traon Y. Metamodel-Based test generation for model transformations: An algorithm and a tool. In: Proc. of the Int'l Symp. on Software Reliability Engineering. New York: IEEE, 2006. 85−94. [doi: 10.1109/ISSRE.2006.27]

[26] Ehrig E, Küster JM, Taentzer G. Generating instance models from Meta models. Software & Systems Modeling, 2009,8(4): 479−500. [doi: 10.1007/s10270-008-0095-y]

何啸(1983－),男,北京人,博士,讲师,CCF 专业会员,主要研究领域为模型驱动工程,模型转换,元建模,代码生成.

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与支撑环境,软件建模技术,面向对象技术.

李文峰(1996－),男,学士,主要研究领域为软件工程.

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环境,面向对象方法,软件复用,软件构件技术.

张天(1978－),男,博士,副教授,主要研究领域为软件工程.

胡长军(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算,数据工程,领域软件工程.