Software Engineering and Applications 软件工程与应用, 2022, 11(6), 1297-1309
Published Online December 2022 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2022.116133

Hans汉斯

一种面向ATL模型转换程序的自动化缺陷修复方法

谭泽理，杜柯柯，江明月

浙江理工大学计算机科学与技术学院，浙江 杭州

收稿日期：2022年11月9日；录用日期：2022年12月6日；发布日期：2022年12月16日

摘　要

模型转换是模型驱动架构(Model Driven Architecture, MDA)的核心活动，它的质量直接影响到基于MDA的软件产品的质量。因此，基于模型转换开展的测试、错误定位以及缺陷修复对于基于MDA的软件开发来说至关重要。然而，由于模型转换程序自身的特点，应用于传统软件的一些方法不能直接应用。本文针对ATL (ATLAS Transformation Language)模型转换程序，提出一种基于随机搜索(random search)的缺陷修复方法，主要包括：1) 利用蜕变关系来描述模型转换程序相关源模型及其对应目标模型所应满足的属性，在此基础上应用蜕变测试技术来实现补丁的验证，以缓解ATL程序自动化缺陷修复中的Oracle问题；2) 针对ATL程序设计并实现9种变异操作(mutation operators)；3) 基于随机搜索来驱动补丁生成和验证。进一步基于该方法实现一个原型工具AMTRepair，并基于AMTRepair开展了初步的实验。实验结果表明针对目标程序，AMTRepair的缺陷修复率约为51%，其正确补丁生成率为30%。

关键词

缺陷修复，ATL模型转换，蜕变测试，变异操作，补丁生成

An Approach for Automatically Repairing ATL Model Transformation Programs

Zeli Tan, Keke Du, Mingyue Jiang

School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou Zhejiang

Received: Nov. 9th, 2022; accepted: Dec. 6th, 2022; published: Dec. 16th, 2022

Abstract

Model transformation program is the core activity of Model Driven Architecture (MDA), and its

文章引用: 谭泽理, 杜柯柯, 江明月. 一种面向 ATL 模型转换程序的自动化缺陷修复方法[J]. 软件工程与应用, 2022, 11(6): 1297-1309. DOI: 10.12677/sea.2022.116133

quality directly affects the quality of software products based on MDA. Therefore, testing, error location and defect repair based on model transformation are critical for MDA based software development. However, due to the characteristics of the model converter itself (such as Oracle problem), some methods applied to traditional software can not be directly applied. This paper proposes a random search based defect repair method for ATL (ATLAS Transformation Language) model transformation program, which mainly includes: 1) Use transformation testing and spectrum error location methods to obtain suspicious ATL rules to deal with the Oracle problem faced by ATL program testing; 2) Design and implement 9 mutation operators according to ATL program; 3) Based on a random search, patch generation and verification are driven. Furthermore, a prototype tool AMTRepair is implemented based on this method, and preliminary experiments are carried out based on AMTRepair. The experimental results show that the defect repair rate of AMTRepair is about 51% and the correct patch generation rate is 30% for the target program.

Keywords

Defect Repair, ATL Model Conversion, Metamorphic Testing, Mutation Operation, Patch Generation

1. 引言

模型转换(Model Transformation)是模型驱动软件架构(Model Driven Architecture, MDA)的核心与灵魂。作为一种新兴的软件开发模式，MDA [1]主张在整个软件开发生命周期围绕模型展开，以此为基础将系统的业务模型与具体的实现细节分离。相比传统的软件开发过程，MDA 能更好地适应多变的需求、不断增长的系统规模、以及日益复杂的系统环境，并能使得所开发的软件具备更好的可重用性和可移植性。这些优势使得基于 MDA 的软件开发模式在多个领域(如分布式系统，电信领域应用，网络应用，国防与航天应用，无线传感器网络等)得到广泛应用和推广[2] [3]。

基于 MDA 的软件开发过程如图 1 所示。MDA 利用不同层次的模型从不同角度来描述软件系统：计算无关模型(Computation Independent Model, CIM)关注系统的业务功能描述；平台无关模型(Platform Independent Model, PIM)从软件系统实现的角度来描述系统业务功能；平台相关模型(Platform Specific Model, PSM)描述利用特定的技术对系统的实现。基于 MDA 的软件开发过程实质上就是实现从 CIM 到 PIM，从 PIM 到 PSM，进而从 PSM 到代码的映射。而不同模型之间的映射则通过模型转换来自动实现。

在 MDA 中，不同模型之间的转换实质上是通过在模型转换平台上运行模型转换程序来自动实现的。也就是说，模型转换的过程就是执行模型转换程序的过程。显然，在 MDA 中要保障最终软件的质量，归根结底在于如何保障模型转换程序的质量。

由于模型转换自身的特点[4] [5]，应用于传统软件上的质量保障技术大部分都不能直接应用于模型转换，因此需要针对模型转换研究适合并有效的质量保障技术。研究者很早就意识到模型转换质量的重要性，并且持续有新的研究成果被发掘，但是目前大部分研究工作主要关注于缺陷检测[6] [7] [8] [9]以及缺陷定位[10] [11] [12] [13] [14]。针对模型转换程序的自动化缺陷修复，当前工作聚焦于典型的语义或者语法错误。但是，由于模型转换调试所面临的 Oracle 问题[9]，当前的方法主要采用人工定义规约或者已有的示例输入－输出来扮演 Oracle 的角色，在应用上有一定的局限性[15] [16] [17]。

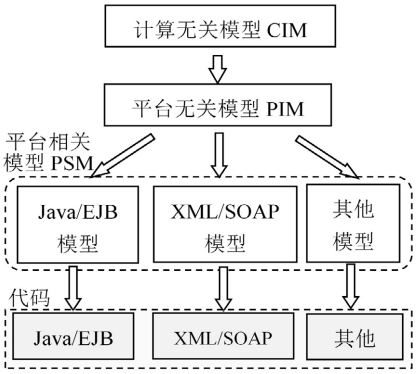


Figure 1. MDA based software development
图 1. 基于 MDA 的软件开发过程

鉴于这种情况，我们立足于一类常用的模型转换程序，ATL (ATLAS Transformation Language)模型转换[18]。针对其功能性缺陷，本文提出并实现一种自动化缺陷修复方法，旨在缓解其所面临的 Oracle 问题。本文的主要贡献如下：

1) 利用蜕变关系来描述模型转换程序相关源模型及其对应目标模型所应满足的属性，在此基础上应用蜕变测试技术来实现补丁的验证，以缓解 ATL 程序自动化缺陷修复中的 Oracle 问题；

2) 针对 ATL 程序设计并实现 9 种变异操作以支持自动化的补丁生成，并应用随机搜索算法来驱动自动化的缺陷修复过程；

3) 实现针对 ATL 程序的自动化缺陷修复工具 AMTRepair，并在公共数据集上开展实验分析。实验结果表明针对目标程序，AMTRepair 的缺陷修复率约为 51%，其正确补丁生成率为 30%。

2. 背景知识

2.1. ATL 模型转换程序

模型转换的目的实现不同模型之间的自动化变换。而模型转换程序是由具体的程序语言来实现的。目前，已经有多种模型转换语言包括 Henshin [19]，AGG [20]，Maude [21]，AToM3 [22]，QVT [23]和 ATL [24]等。其中，ATL (ATLAS Transformation Language)是一种基于文本规则的模型转换语言，它提供声明式和命令式的语言概念，提供元模型标准支持并能快速集成到开发平台。这些使得 ATL 在学术领域和工业界都得到广泛的应用。本文将聚焦于 ATL 模型转换程序，并研究其自动化的缺陷修复方法。

模型转换的过程如图 2 所示。一个 ATL 模型转换程序的输入为一个源模型，其输出为对应的目标模型。源(目标)模型遵循相应的源(目标)元模型的定义。一个 ATL 程序主要由一系列 rule 和 helper 组成。其中 rule 描述了如何从特定的源模型元素生成目标模型的对应元素，而 helper 实现了某些辅助功能。

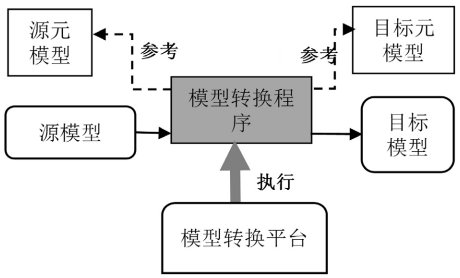


Figure 2. Procedure of model transformation
图 2. 模型转换基本过程

图 3 展示了一个 ATL 模型转换示例 Class2Relational，它实现了从 Class 模型(源元模型如图 3(a)所示)到 Relational 模型(目标元模型如图 3(b)所示)的转换。一个 Class 模型由一个或多个 Class 构成，每个 Class 包含一组 Attribute，Attributes 分为单值和多值。一个 Relational 模型由一个或多个 Table 构成，每个 Table 包含一组 Columns，并对其有键的引用；类 Column 有一个对 Type 的引用。

一个 ATL 程序中包含多个 rule。如图 3(c)所示，*Class2Relational* 程序片段中展示了 2 个 rule：*DataType2Type*，*SingleValuedDataTypeAttribute2Column*。一个 rule 由几个部分组成，包括：1) 源模型的被映射元素 inElement，由 from 关键字引导，声明被该 rule 匹配的源模型元素；2) 源模型的元素筛选条件 SM Filter condition，描述该 rule 所针对的源模型元素所需要满足的条件；3) 目标模型元素 outElement，声明所生成的目标元素；4) 与 outElment 相关的一系列 bingdings，描述目标模型元素值的获取方式。以图 3(c)中的 rule *SingleValuedDataTypeAttribute2Column* 为例(9~19 行)。其 inElement 为 Class!Attribute，即 Class 模型中的 Attribute 元素(11 行)，SM Filter condition 是单值属性(12 行)。该 rule 的 outElement 为 Relational 模型中的 Column 元素(15 行)，其相应的 bindings 声明了 Column 元素每个属性值的获取方法(16~17 行)。

随着模型转换技术在不同领域的应用和发展，其质量问题也得到广泛关注。然而，对其进行质量评估与维护面临诸多挑战，其中之一就是 Oracle 问题，即很难评判模型转换程序输出的正确性[9]。

2.2. 蜕变测试

蜕变测试(Metamorphic Testing, MT) [25]已被成功应用到大量的软件系统以有效地缓解 Oracle 问题。蜕变测试的核心概念是蜕变关系(Metamorphic Relations, MRs)，蜕变关系是对目标程序相关输入和对应的输出之间关系的一种描述，是程序必要属性的反映。蜕变测试的核心思想是借助蜕变关系从初始输入中构造衍生输入，进而通过判断相关初始输出和衍生输出之间是否符合蜕变关系来判断程序是错误的。通常，利用蜕变测试组(Metamorphic Test Group, MTG)来表示与给定蜕变关系相关的一组初始输入和衍生输入。

以一个程序 P (实现了正弦函数 sin 的计算)为例。对于任取的一个 x 值，很难确定 $P(x)$的正确性，即测试 P 面临 Oracle 问题。从蜕变测试的角度，可以考虑 sin 函数所满足的一些特性，比如 $\sin(x)=\sin(\pi-x)$。借助此特性，可以构造的一个蜕变关系：$P(x)=P(\pi-x)$，进而对 P 进行测试。例如，给定一个初始输入 37，根据蜕变关系构建其衍生输入为 143，然后比较 $P(37)$与 $P(143)$，如果它们不相等，此蜕变关系被违反，即说明程序 P 是错误的。

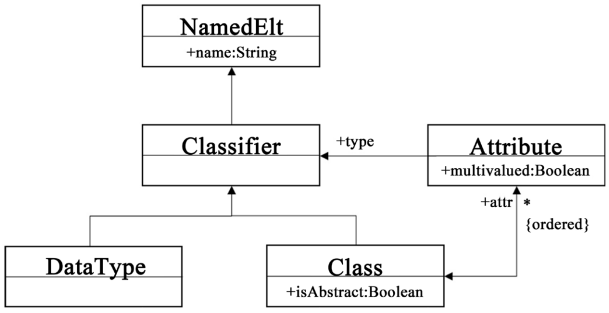
经过几十年的发展和研究，蜕变测试已经被广泛应用于众多领域中，包括编译器的测试、搜索引擎和机器学习等。

3. ATL 模型转换程序自动化缺陷修复

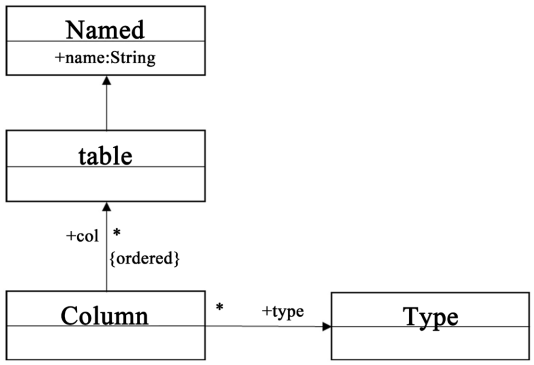
3.1. 方法概述

鉴于调试 ATL 模型转换程序所面临的 Oracle 问题，我们将结合蜕变测试技术来缓解修复 ATL 程序缺陷所面临的 Oracle 问题。本文提出一种基于蜕变测试技术的缺陷修复方法 AMTRepair，总体框架如图 4 所示。基于给定的错误 ATL 程序 P、一组蜕变关系 *MRs*，以及一些初始的测试用例 *tcs*。AMTRepair 首先基于 *MRs* 构造 MTG 集合 S。在本文中一个 MTG 为一个初始源模型和它对应的衍生源模型。然后，AMTRepair 应用 MT 以及 S 来对 P 进行测试并收集相关的覆盖信息。在此基础上，AMTRepair 应用基于频谱的错误定位(spectrum-based fault localization, SBFL)方法[26]来对 P 中的规则的可疑度进行计算，从而获得一个可疑的规则的列表。最后，AMTRepair 应用随机搜索算法，随机选择变异的操作对可疑的规则

中的代码进行修改生成候选补丁，进而采用 MT 对补丁进行验证，直至生成一个满足 *S* 中所有 MTGs 的补丁。



(a) Class 元模型



(b) Relational 元模型

```
1  rule DataType2Type {
2  from
3  dt : Class!DataType
4  to
5  out : Relational!Type (
6  name <- dt.name
7  )
8  }
9  rule SingleValuedDataTypeAttribute2Column {
10 from
11 a : Class!Attribute (
12 a.type.oclIsKindOf(Class!DataType) and not a.multiValued
13 )
14 to
15 out : Relational!Column (
16 name <- a.name,
17 type <- a.type
18 )
19 }
```

(c) 模型转换程序(片段)

Figure 3. Illustrative example: *Class2Relational*
图 3. 说明型示例：*Class2Relational*

3.2. 蜕变关系选取

蜕变关系是 MT 的核心部分，因此将 MT 应用到模型转换上的关键点在于 MR 的构造。我们前期的工作已经将 MT 应用到 ATL 程序测试以及缺陷定位中[27] [28]。以此为基础，本文使用以下三类蜕变关系。

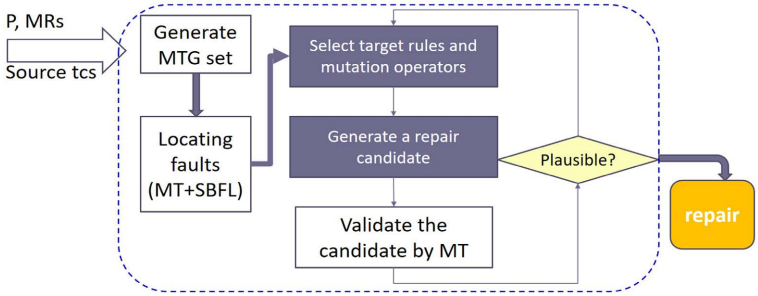


Figure 4. Overall framework of AMTRepair
图 4. AMTRepair 总体框架

1) MR-M：通过更改初始源模型的一些元素属性值来构造衍生源模型，则初始输出和衍生输出中相关元素值相应不同。

2) MR-A：通过向初始源模型中添加一个元素来构造衍生源模型，则衍生输出比初始输出包含多的元素。

3) MR-D：通过删除初始源模型中的一个元素来构建衍生源模型，则相比初始输出，衍生输出中包含相应多的元素。

针对一个目标 ATL 程序，AMTRepair 应用多个蜕变关系，旨在获得高质量的补丁。

3.3. 变异操作

AMTRepair 采用程序变异的方法来生成补丁。我们基于 ATL 程序的特点，设计了 3 类变异操作，分别作用于 ATL rule 的四个主要部分，inElement，OutElement，SM filter condition，以及 bindings。

1) 删除操作(deletion, D)：删除一条语句或者表达式。

2) 修改操作(Modification, M)：修改语句的部分表达式。对于 inElement 或者 outElement，修改主要在子元素(也就是“!”后的部分)上进行，而对于 bindings，修改操作用于赋值运算符“<-”右边的表达式。

3) 插入操作(Insertion, I)：插入一条语句或者表达式。

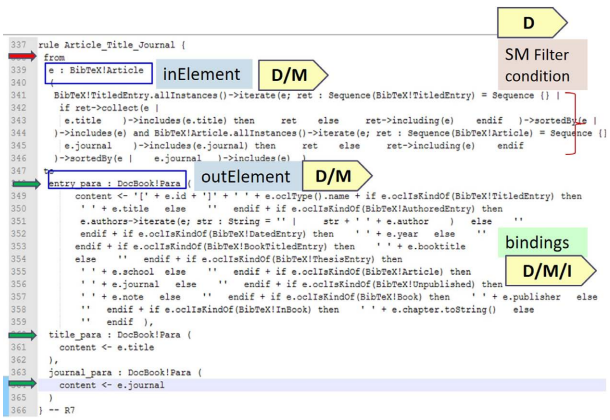


Figure 5. Mutation operations against ATL programs
图 5. 针对 ATL 程序的变异操作

图 5 为我们截取实验中被测程序的片段，其中我们标记了对程序进行的变异操作所对应的位置及变异方法。

3.4. 基于随机搜索的补丁生成

AMTRepair 应用随机搜索来驱动补丁生成和验证过程。随机搜索驱动的补丁生成算法如算法 1 所示。该算法的输入为目标 ATL 程序 P，MTG 集合 S 以及规则可疑度列表 L。主要思想为根据可疑规则列表对模型转换程序语句应用变异操作并生成候选补丁，进而利用 MT 对候选补丁进行验证。如果该补丁通过验证则返回该补丁，否则继续生成其他候选补丁。

Algorithm 1. Random search-based defect repair
算法 1. 基于随机搜索的缺陷修复

Input: faulty program *P*;
Input: MTG set *S*;
Input: suspicious rule list *L*;
Output: a plausible repair
begin
1 **for** *rule* **in** *L*
2 **do**
3 *m_candidates* = ***mutate*** (*P*, *rule*);
4 **for** *c* in *m_candidates* **do**
5 **If** ***validate*** (*c*, *S*) == true **then**
6 return c;
7 **end if**
8 **do**
9 for c1, c2 in *m_candidates* do
10 *c* = ***crossover*** (c1, c2)
11 *c_candidates* = *c_candidates* ∪ {*c*}
12 done
13 **for** *c* in *c_candidates* **do**
14 **If** ***validate*** (*c*, *S*) == true **then**
15 return c;
16 **end if**
17 **do**
18 **end while**
19 **end**

在一次随机搜索过程中，首先基于给定程序 *P* 和错误定位给出的可疑 rule 列表 *L* 来生成候选补丁。即选择被操作 rule (可疑度越高的 rule 有越高的机会被选择)以及相应的操作。一方面，变异操作将对 ATL 程序中对应位置进行修改，例如删除某个 rule 中的一条语句，在满足语法要求的情况下生成新的程序(第 3 行)。另一方面，crossover 操作将结合所生成的候选补丁进行重组，以生成新的补丁(第 10 行)。由于这两次选择的随机性，结果的候选补丁 candidate 中，可能包含一个或者多个修改。每一个候选补丁生成后，MTG 集 *S* 被用来对其进行评估以确定是否是合格的补丁(5 行和 14 行)。我们的评估标准是：满足 *S* 中所有的 MTG 的补丁为一个合格补丁。当一个合格补丁被生成，整个补丁生成过程结束。否则，以上过程一直重复直至所有的可能候选补丁都被生成并被验证。

3.5. 原型工具实现

我们实现一个原型工具 AMTRepair。AMTRepair 主要基于 Java 开发，融合了 ATL 模型转换的执行、追踪等机制。AMTRepair 主要包含以下部分。

1) 自动化蜕变测试框架。基于给定蜕变关系和目标程序，自动实施蜕变测试。一方面，实现 3.2 章所设计的蜕变关系，实现衍生模型的自动生成，以及初始、衍生目标模型的自动化检查。另一方面，基于 ATL engine 和 ATL core API 实现从 java 中直接运行 ATL 程序。

2) 基于频谱的错误定位框架。一方面，选取并实现 Tarantula 公式[29]来进行规则可疑度的计算。另一方面，应用 ATL 模型转换 Trace mechanism [30]收集 ATL 程序运行时的覆盖信息，从而实现错误定位。

3) ATL 语句自动化变异操作。利用 parser 实现 ATL 程序文件到 ATL 模型的转换，在次基础上实现各类变异操作。

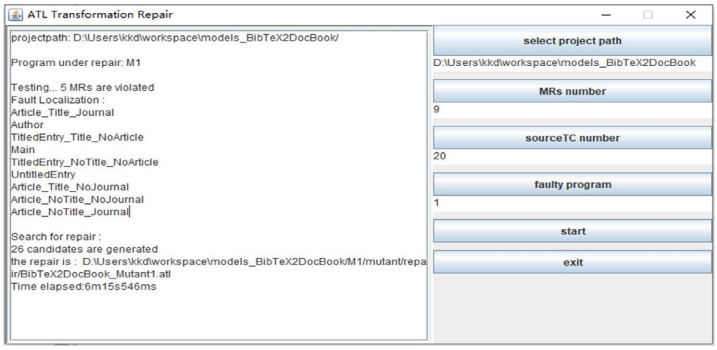


Figure 6. Mutation operations against ATL programs
图 6. 针对 ATL 程序的变异操作

AMTRepair 的操作界面如图 6 所示。左边是显示缺陷修复过程中的一些重要信息，右边包含进行缺陷修复所需要的一些配置信息。用户可以在界面右侧输入错误程序的路径位置，设置的 MR 的数量，使用的测试用例的数量以及修复的程序的编号这些信息。点击 start 后该工具开始运行并在左侧展示信息。首先会展示此时正在修复的程序的位置、该程序违反的 MR 的数量以及错误定位所确定的该程序中规则的可疑度列表等基本信息。此外，紧跟着会展示当前进行变异操作的位置，即针对哪些规则进行变异操作，以及生成的候选补丁的数量及被验证通过的补丁的保存位置等详细信息，方便用户实时了解修复过程中的每个阶段及相关信息。

4. 实验结果分析

我们开展实验来评估 AMTRepair 修复错误的能力。本节报告了我们的实验结果，并展示了基于实验结果进行进一步的分析。

4.1. 实验设计与配置

4.1.1. 目标程序

实验选择 3 个公开的 ATL 项目作为研究对象：Class2Relational [9]，Class2Table [31]，BibTex2DocBook [14]。Class2Relational 和 Class2Table 均实现了从 Class 模型到 Relational Table 模型的转换。BibTex2DocBook 实现了从 BibTeXML 模型到 DocBook 组成文档的转换。其中，BibTeXML 是 BibTeX 书目工具的一种基于 xml 的格式，而 DocBook 是一种基于 xml 的文档组合格式。3 个项目的具体信息如表 1 所示。基于 3 个项目，我们一共针对 70 个错误 ATL 程序进行缺陷修复。

4.1.2. ATMRepair 配置

ATMRepair 的输入包含被测错误程序、可疑度列表以及 MTG。而 MTG 的构造则使用了 3.3 节中的蜕变关系来生成。ATMRepair 利用变异操作来生成候选补丁。在 3.4 节中我们介绍了三大类变异操作，我们使用了这三大类变异操作中共 9 种变异操作，具体如表 2 所示。对于每一个目标程序，我们通过随机测试模型生成方法为其准备 100 个初始源模型，并一个用对应的蜕变关系来构造相应的衍生源模型。

Table 1. Information of target programs
表 1. 目标程序信息

object programs	mutants	helpers	rules
BibTex2DocBook	40	3	9
Class2Relational	15	1	6
Class2Table	15	2	8
Total	70	6	23

Table 2. Information of mutation operator
表 2. 变异操作信息

类别	变异操作
Deletion	Binding deletion out-pattern element deletion filter deletion
Addition	binding addition out-pattern element addition in-pattern element addition
Modification	binding value change in-pattern element class change out-pattern element class change

4.1.3. 补丁评价方法

为了评价 AMTRepair 的修复效率，我们参照相关工作统计它所生成的补丁数量及正确补丁数量[16] [17]。在验证补丁时我们设定的是生成的补丁能否通过所有的测试用例，如果通过，则认为是成功生成补丁。然而在这些补丁应用的程序中，存在通过所有测试用例但其实并未完全正确修复的情况。因此为了客观评价程序实验的结果，我们会对程序进行进一步的查验，人工地去分析判断最后程序是否被成功修复，以此分析 AMTRepair 的有效性。

4.2. 实验结果

我们利用 AMTRepair 对每一个错误版本进行修复，修复结果统计在表 3 中。我们在进行数据统计时将修复程序的个数分为被修复的个数和被正确修复的个数两个部分。其中，被修复的个数指的是生成的补丁能够通过所有的测试用例所修复的程序个数。而通过人工检查，在被修复的程序中被正确修复的程序个数，我们统计为被正确修复的个数。从表 3 可以看出，对于目标程序 BibTeX2DocBook，修复率(被修复个数/错误版本数)为 50%，正确修复率(被正确修复的个数/错误版本数)为 20%，其中正确修复个数占修复个数的 40%；程序 Class2Relational 修复率和正确修复率约为 60%和 47%，其中正确修复个数约占修复个数的 78%；Class2Table 中修复个数约占总数的 47%，正确修复率为 40%，正确修复数大约占修复数的 86%。总体来看，AMTRepair 可以修复大约 51%的错误版本，其中正确修复的补丁占总体的 30%，占修复个数约 58%。

图 7 则展示了一个错误转换程序的修复案例，其中图 7(a)为错误程序，我们可以看到该程序的错误发生在“"name <-a.name"”语句，它的正确版本应该是“name <- a.name + 'Id'”。而经过我们的工具修复，成功生成了修复补丁，如图 7(b)所示。该补丁在该语句所在的 rule 中添加了语句“name <- a.name + 'Id'”，而这也成功通过了所有的测试用例，返回为修复版本。正如 4.1 中最后讨论的那样，存在一些被修复但并未正确修复的情况，这一般是因为此次修复恰好通过了所有的测试用例，但实际它修复的位置可能不正确，甚至并未进行任何修复，因此我们需要对修复结果进行人工验证。

Table 3. Experimental result
表 3. 实验结果

目标程序	被修复的个数	被正确修复的个数
BibTex2DocBook	20	8
Class2Relational	9	7
Class2Table	7	6

在后续对实验结果进行进一步分析时，我们发现除了那些正确修复缺陷程序的补丁外，那些通过指定的测试用例但并未完全修复错误程序的候选补丁并非是无用的，相反他们可以为开发人员在修复程序缺陷时提供一定的帮助。同时我们还发现，程序缺陷修复的正确率同时受到程序本身错误类别的影响，一方面是因为变异操作还不能完全覆盖程序所有的错误类别。因此无法生成正确的修复补丁；另一方面则是由于程序本身的结构或者错误点的特殊性导致测试用例无法发现该错误或修复无法达到该位置。此外实验结果显示，错误定位的准确性也影响着程序修复的效率。

```
rule ClassAttribute2Column {
        from
                a : Class!Attribute (
                        a.type.oclIsKindOf(Class!Class) and not a.multiValued
                )
        to
                foreignKey : Relational!Column (
                        name <- a.name, --mutation: bing value change+ 'Id',
                        type <- thisModule.objectIdType
                )
}
```

(a)

```
rule ClassAttribute2Column {
 from
  a : Class!Attribute
  (
   a.type.oclIsKindOf(Class!Class) and not a.multiValued
  )
 to
  foreignKey : Relational!Column (
   name <- a.name,
   --mutation: bing value change+ 'Id',
   type <- thisModule.objectIdType,
   name <- a.name + 'Id'
  )
}
```

(b)

Figure 7. Patch repair in Class2Table
图 7. Class2Table 补丁修复

5. 相关工作

在软件开发过程中，诸如需求上的理解错误、开发者本身的操作问题或者经验不足等原因，使得软件缺陷难以避免。这些缺陷可能会导致正在运行的软件发生错误甚至崩溃，从而带来严重的损失。因此，为了保证开发的软件质量，研发人员希望在没有人为干预的情况下自动修复缺陷，并通过设计高质量的测试用例来尽可能多的发现程序中的缺陷，这便是自动缺陷修复或自动补丁生成技术。此类技术的最终目的就是自动生成正确的补丁，以达到修复软件中的错误。

目前，程序自动修复大致分为两类：基于测试用例的程序错误修复和其他类型的程序错误修复[32]。本文基于测试用例的程序错误修复。该修复技术的输入是待修复的程序和测试用例，基本过程为利用错误定位技术定位程序错误的位置，然后根据对应错位位置生成候选补丁，最后对候选补丁进行验证。由此可见，自动缺陷修复离不开错误定位技术的支持。往往错误定位技术的高准确率能够很大程度的提高

自动缺陷修复的效率。

然而，针对模型转换程序的缺陷修复研究仍然处于雏形阶段。2015 年，基于对模型转换程序实现过程，特殊性的观察和理解，Sánchez 等发现大多数的模型转换程序实现过程中都需要手动的输入相关模型的特征(如属性等)，而这一过程很容易引起语法错误。由此，他们洞差到自动化修复模型转换程序对于 MDE 领域的重要性，并提出一种快速修复 ATL 模型转换语法错误的方法[33]。为了生成补丁，该方法对检测到的语法错误进行分类，并为每一类的错误提供预定义的修复模板。考虑到同一种错误可能可以通过不同的方法来修复，Sánchez 等进一步扩大了错误分类，并引入可预测性分析方法来对可能的补丁进行排序[15]。同样针对 ATL 模型转换程序的语法错误，VaraminyBahnemiry 等提出将补丁看作是一系列编辑操作的序列，而缺陷修复需要满足两个目标，一是尽可能减少错误的数量，另外一个是保留模型转换的预期行为[16]。基于这个思想，他们将缺陷修复转化为多目标优化问题，通过应用变异操作符来构造不同的候选补丁，并应用进化算法探索并选择满足目标的候选补丁。最后，通过定义四种不同的启发式规则来改善候选补丁以增加它们与修复目标的匹配度。

2021 年，以自动化修复 ATL 模型转换的语义错误为目标，VaraminyBahnemiry 等[17]进一步基于进化算法提出了解决方案。他们利用示例输入－输出模型来代表模型转换的预期行为规约，并旨在为给定模型转换程序生成满足预期行为规约的补丁。针对 9 类语义错误，他们相应的提出了 11 种修改操作。整个缺陷修复过程基于进化算法，在每次迭代中通过利用预定义的遗传算子来生成下新的下一代候选补丁，每一个候选补丁通过一个 fitness 函数来计算一个质量评分。鉴于需求改变及模型转换进化带来的问题，Rodriguez-Echeverria 等提出利用基于 contracts 的测试方法来检测和修复该类问题，进而为开发者提供一系列候选的修复操作[34]。此方法需要开发者手工的按照 contracts 的方式来说明模型转换的预期行为，进而自动化的从 contracts 中抽取出 Oracles，最后通过比较分析模型转换执行轨迹和测试结果来推荐可行的修复操作。

现有的研究工作为自动修复模型转换程序奠定了良好的基础。然而，不管是从目前所覆盖的错误范围，亦或是从技术应用推广的角度，针对模型转换的缺陷修复工作仍存在诸多局限。一方面，面向模型转换的缺陷修复技术有待进一步发展和精进。目前针对模型转换的缺陷修复技术仍然处于发展的初期，当前工作所考虑的补丁生成方法不够全面。此外，当前的方法主要采用人工定义规约或者已有的示例输入－输出来扮演 Oracle 的角色，在应用上有一定的局限性。从修复效率、性能及应用的角度出发，需要进一步考虑对缺陷修复每个阶段的优化策略，以及融合适合的方法来提升修复效率和修复性能。

6. 总结

如今软件应用发展日渐复杂，其中基于模型驱动的软件开发因其能很好地适应多变的需求和复杂的环境而逐渐受到更多的应用。而正如我们之前所讨论，决定基于模型驱动的软件质量的关键是模型转换的正确性，而模型转换的过程实质就是模型转换程序在模型转换平台上的实现过程。因此，保证软件质量的根本就是保证模型转换程序的质量。模型转换程序存在测试 Oracle 问题，本文基于已有的针对普通程序和模型转换程序的相关工作，对此类问题进行分析，提出结合蜕变测试的方法来解决 Oracle 问题。本文提出了一种程序缺陷修复工具 ATMRepair，该工具利用错误定位方法生成的可疑度列表生成候选补丁，并基于随机搜索的思想对补丁进行测试，直到找到正确修复版本。最后我们通过实验分析说明了该工具在模型转换程序上的可行性和有效性。

虽然 AMTRepair 在一定程度上可以修复模型转换中存在的缺陷，但是在实验过程中我们也发现该工具存在着一定的局限性，例如其修复的概率和种类受到错误程序的类型的影响。因此在后续工作中，我们需要对该方法作进一步的分析和改进，对实验结果进行更深入的研究。此外我们还将进一步细化针对

ATL 的变异操作，以使得 AMTRepair 可以应对更多类型的错误；另一方面，我们将进一步开展大量的实验来展示 AMT-Repair 的修复能力和效率。

基金项目

国家自然科学基金项目(No.61802349)；浙江省自然科学基金项目(LY20F020021)。

参考文献

[1] Brambilla, M., Cabot, J. and Wimmer, M. (2012) Model-Driven Software Engineering in Practice, Series Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, San Rafael. https://doi.org/10.1007/978-3-031-02546-4

[2] Mohagheghi, P. and Dehlen, V. (2008) Where Is the Proof?—A Review of Experiences from Applying MDE in Industry. *Proceedings of the European Conference on Model Driven Architecture-Foundations and Applications*, Berlin, 9-13 June 2008, 432-443. https://doi.org/10.1007/978-3-540-69100-6_31

[3] Asadi, M. and Ramsin, R. (2008) MDA-Based Methodologies: An Analytical Survey. *Proceedings of the European Conference on Model Driven Architecture-Foundations and Applications*, Berlin, 9-13 June 2008, 419-431. https://doi.org/10.1007/978-3-540-69100-6_30

[4] Lúcio, L., Amrani, M., Dingel, J., *et al*. (2016) Model Transformation Intents and Their Properties. *Software and Systems Modeling*, **15**, 647-684. https://doi.org/10.1007/s10270-014-0429-x

[5] Baudry, B., Ghosh, S., Fleurey, F., *et al*. (2010) Barriers to Systematic Model Transformation Testing. *Communication of ACM*, **53**, 139-143. https://doi.org/10.1145/1743546.1743583

[6] Sen, S., Baudry, B. and Mottu, J.-M. (2008) On Combining Multi-Formalism Knowledge to Select Models for Model Transformation Testing. *Proceedings of the* 1*st International Conference on Software Testing, Verification, and Validation*, Lillehammer, 9-11 April 2008, 328-337. https://doi.org/10.1109/ICST.2008.62

[7] Wieber, M., Anjorin, A. and Schürr, A. (2014) On the Usage of TGGs for Automated Model Transformation Testing. *Proceedings of the International Conference on Theory and Practice of Model Transformations*, York, 21-22 July 2014, 1-16. https://doi.org/10.1007/978-3-319-08789-4_1

[8] Lano, K. and Clark, D. (2008) Model Transformation Specification and Verification. *Proceedings of the Eighth International Conference on Quality Software*, Oxford, 12-13 August 2008, 45-54. https://doi.org/10.1109/QSIC.2008.38

[9] Jiang, M.Y., Chen, T.Y., Kuo, F.-C., *et al*. (2014) Testing Model Transformation Programs using Metamorphic Testing, *The* 26*th IEEE International Conference on Software Engineering and Knowledge Engineering* (*SEKE*'14), Vancouver, 1-3 July 2014, 94-99.

[10] Hibberd, M., Lawley, M. and Raymond, K. (2007) Forensic Debugging of Model Transformations. *Proceedings of the* 10*th International Conference on Model Driven Engineering Languages Systems*, Nashville, 30 September-5 October 2007, 589-604. https://doi.org/10.1007/978-3-540-75209-7_40

[11] Wimmer, M., Kappel, G., Kusel, A., *et al*. (2009) A Petri Net Based Debugging Environment for QVT Relations. *Proceedings of the* 24*th IEEE/ACM International Conference on Automated Software Engineering*, Washington DC, 16-20 November 2009, 3-14. https://doi.org/10.1109/ASE.2009.99

[12] Aranega, V., Mottu, J.-M., Etien, A. and Dekeyser, J.-L. (2009) Traceability Mechanism for Error Localization in Model Transformation. *Proceedings of the International Conference on Software and Data Technology*, Sofia, 26-29 July 2009, 66-73.

[13] Burgueño, L., Troya, J., Wimmer, M. and Vallecillo, A. (2015) Static Fault Localization in Model Transformations. *IEEE Transactions on Software Engineering*, **41**, 490-506. https://doi.org/10.1109/TSE.2014.2375201

[14] Troya, J., Segura, S., Parejo, J.A. and Ruiz-Cortés, A. (2018) Spectrum-Based Fault Localization in Model Transformations. *ACM Transactions on Software Engineering and Methodology*, **27**, 1-50. https://doi.org/10.1145/3241744

[15] Cuadrado, J.S., Guerra, E. and de Lara, J. (2018) Quick Fixing ATL Transformations with Speculative Analysis. *Software and Systems Modeling*, **17**, 779-813. https://doi.org/10.1007/s10270-016-0541-1

[16] Varaminy Bahnemiry, Z., Galasso, J. and Sahraoui, H. (2020) Fixing Multiple Type Errors in Model Transformations with Alternative Oracles to Test Cases. *Journal of Object Technology*, **20**, 1-14. https://doi.org/10.5381/jot.2021.20.3.a9

[17] Varaminy Bahnemiry, Z., Galasso, J., Belharbi, K. and Sahraoui, H. (2021) Automated Patch Generation for Fixing Semantic Errors in ATL Transformation Rules. *Proceedings of the ACM/IEEE* 24*th International Conference on Model Driven Engineering Languages and Systems* (*MODELS*'21), Fukuoka, 10-15 October 2021, 13-23.

https://doi.org/10.1109/MODELS50736.2021.00011

[18] Jouault, F., Allilaire, F., Bézivin, J., *et al*. (2008) ATL: A Model Transformation Tool. *Science of Computer Programming*, **72**, 31-39. https://doi.org/10.1016/j.scico.2007.08.002

[19] Arendt, T., Biermann, E., Jurack, S., *et al*. (2010) Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In: *International Conference on Model Driven Engineering Languages and Systems*, Springer, Berlin, 121-135. https://doi.org/10.1007/978-3-642-16145-2_9

[20] Taentzer, G. (2003) AGG: A Graph Transformation Environment for Modeling and Validation of Software. In: *International Workshop on Applications of Graph Transformations with Industrial Relevance*, Springer, Berlin, 446-453. https://doi.org/10.1007/978-3-540-25959-6_35

[21] Clavel, M., Durán, F., Eker, S., *et al*. (2007) All about Maude—A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic. Springer, Berlin.

[22] De Lara, J. and Vangheluwe, H. (2002) AToM3: A Tool for Multi-Formalism and Meta-Modelling. In: *International Conference on Fundamental Approaches to Software Engineering*, Springer, Berlin, 174-188. https://doi.org/10.1007/3-540-45923-5_12

[23] Greenyer, J. and Kindler, E. (2010) Comparing Relational Model Transformation Technologies: Implementing Query/View/Transformation with Triple Graph Grammars. *Software & Systems Modeling*, **9**, Article No. 21. https://doi.org/10.1007/s10270-009-0121-8

[24] Jouault, F., Allilaire, F., Bézivin, J., *et al*. (2006) ATL: A QVT-Like Transformation Language. *Companion to the* 21*st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, Portland, 22-26 October 2006, 719-720.

[25] Chen, T.Y., Cheung, S.C. and Yiu, S.M. (1998) Metamorphic Testing: A New Approach for Generating Next Test Cases. Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01.

[26] Du, K., Jiang, M., Ding, Z., *et al*. (2019) Metamorphic Testing in Fault Localization of Model Transformations. In: *International Workshop on Structured Object-Oriented Formal Language and Method*, Springer, Cham, 299-314. https://doi.org/10.1007/978-3-030-41418-4_20

[27] Jiang, M.Y., Chen, T.Y., Kuo, F.C., *et al*. (2014) Testing Model Transformation Programs Using Metamorphic Testing. *International Conference on Software Engineering & Knowledge Engineering*, Vancouver, 1-3 July 2014, 94-99.

[28] Xie, X., Wong, W.E., Chen, T.Y., *et al*. (2013) Metamorphic Slice: An Application in Spectrum-Based Fault Localization. *Information and Software Technology*, **55**, 866-879. https://doi.org/10.1016/j.infsof.2012.08.008

[29] Janssen, T., Abreu, R. and Van Gemund, A.J.C. (2009) Zoltar: A Spectrum-Based Fault Localization Tool. *Proceedings of the* 2009 *ESEC/FSE workshop on Software Integration and Evolution@ Runtime*, New York, August 2009, 23-30.

[30] Jouault, F. (2005) Loosely Coupled Traceability for ATL. *Proceedings of the European Conference on Model Driven Architecture Workshop on Traceability*, Nuremberg, May 2005, 29-37.

[31] ATL Transformations in Eclipse (2005). https://www.eclipse.org/atl/atlTransformations/Class2Relational/ExampleClass2Relational%5bv00.01%5d.pdf

[32] 王赞, 郜健, 陈翔, 等. 自动程序修复方法研究述评[J]. 计算机学报, 2018, 41(3): 588-610.

[33] Cuadrado, J.S., Guerra, E. and de Lara, J. (2015) Quick Fixing ATL Model Transformations. *Proceedings of the* 2015 *ACM/IEEE* 18*th International Conference on Model Driven Engineering Languages and Systems* (*MODELS*'15), Ottawa, 27 September-2 October 2015. 146-155.

[34] Rodriguez-Echeverria, R., Macías, F., Rutle, A. and Conejero, J.M. (2021) Suggesting Model Transformation Repairs for Rule-Based Languages Using a Contract-Based Testing Approach. *Software and Systems Modeling*, **21**, 81-112. https://doi.org/10.1007/s10270-021-00891-0