文章编号:1007-5321(2019)04-0070-06 DOI:10.13190/j.jbupt.2018-283

基于图论和 FSM 的 UML 模型与代码一致性检测

王 雷
(中国矿业大学(北京) 机电与信息工程学院, 北京 100083)

摘要: 提出了一种基于图论和有限状态机(FSM)的统一建模语言(UML)模型与代码一致性检测方法. 给出了该方法的基本思路;分别讨论了 UML 模型与代码静态一致性检测和动态一致性检测算法;实现了该检测方法的支撑工具,并使用该工具对 C++项目 UMLChecker 1.0 进行了检测. 实验结果表明,所提方法可对 UML 模型与代码的一致性进行检测,且具有较高的检测精度. 通过对静态行为和动态行为的一致性检测可知,检测精度明显提升.

关 键 词: 一致性检测; 检测精度; 图论; 有限状态机; 软件验证
中图分类号: TP311.5 **文献标志码**: A

Consistency Checking Between UML Models and Code Based on Graph Theory and FSM

WANG Lei
(School of Mechanical Electronic and Information Engineering, China University of Mining and Technology, Beijing 100083, China)

Abstract: A unified modeling language (UML) model and code consistency checking method based on graph theory and finite state machine (FSM) was proposed. The basic idea of this method was given; the static consistency checking and the dynamic consistency checking algorithms between UML models and code were discussed respectively; a support tool for this method was implemented, and C++ project UMLChecker 1.0 was checked by using this tool. The experimental results show that this method can check consistency between UML models and code, and has a high checking accuracy. For consistency checking in behavioral aspects, checking accuracy rate is improved obviously.

Key words: consistency checking; checking accuracy; graph theory; finite state machine; software verification

统一建模语言(UML, unified modeling language)是对象管理组织(OMG, object management group)采纳的一种标准建模语言,已经成为事实上面向对象建模的语言标准. 在软件开发过程中保持 UML 模型与源代码之间的一致性,对于软件项目,尤其是大型软件项目后期的理解、维护和重构很有帮助. 近年来软件系统的规模越来越大,UML 模型与源代码一致性检测的研究变得非常有意义. 相关学者已经提出很多不同 UML 图之间的一致性检测(水平一致性检测)[1-2]或 UML 模型不同版本之间的一致性检测(垂直一致性检测)方法[3]. 这些方法主要是针对 UML 模型本身的一致性检测,并不能检测 UML 模型与源代码之间的一致性. 还有学者提出了一些由 UML 模型自动生成一致性代码的方法[4-5],这些方法仍然无法对已有的源代码和 UML 模型进行一致性检测. 近年来有学者提出少量 UML

收稿日期: 2018-11-16
基金项目: 国家自然科学基金项目(60873093); 国家科技重大专项项目(2017ZX05018-005); 延安大学科研计划项目(YDY2019-16)
作者简介: 王 雷(1988—), 男, 讲师, E-mail:wanglei0823@foxmail.com.

模型代码之间的一致性检测方法[6-8]，然而，这些方法都不执行源代码来检测运行期间实际发生的方法调用与 UML 模型的一致性，在分析行为方面是有限且不精确的.

针对以上问题，提出一种基于图论和有限状态机（FSM，finite state machine）的 UML 模型（鉴于 UML 类图和顺序图是最常用的 2 种 UML 图，目前只考虑 UML 类图和顺序图）与源代码一致性检测方法，并实现了该方法的支撑工具. 该方法借助图的深度优先遍历和图同构判定算法对 UML 模型与源代码进行一致性静态检测，并将运行期间实际发生的方法调用与 UML 顺序图转换得到的 FSM 进行匹配来检测动态行为的一致性. 实验结果表明，该方法可以对 UML 模型与代码的一致性进行检测，且具有较高的检测精度. 对静态行为和动态行为的一致性检测效果尤其明显.

1　基本思路

采用二级检测提高 UML 模型与源代码一致性检测精度，即首先借助图的深度优先遍历和图同构判定算法对 UML 模型与源代码进行一致性静态检测. 在进行一致静态检测后，借助 FSM 的验证技术检测 UML 模型与源代码的动态行为一致性. 需要说明的是，这里不对 UML 模型进行验证，认为 UML 模型是一致且正确的，即当检测出不一致时，以 UML 模型为准修改源代码.

2　UML 模型与代码一致性静态检测

2.1　基本流程

UML 模型与代码一致性静态检测的基本流程是：首先对源代码进行静态信息预处理，获取源代码的静态结构信息和静态行为信息；然后检测源代码和 UML 类图的静态结构一致性，若存在不一致，则修改源代码，直到源代码的静态结构与 UML 类图完全一致；最后借助图的深度优先遍历和图同构判定算法对源代码转换得到的有向图和 UML 顺序图转换得到的有向图进行一致性检测，若存在不一致，则修改源代码，直到源代码的静态行为与 UML 序列图完全一致，如图 1 所示.

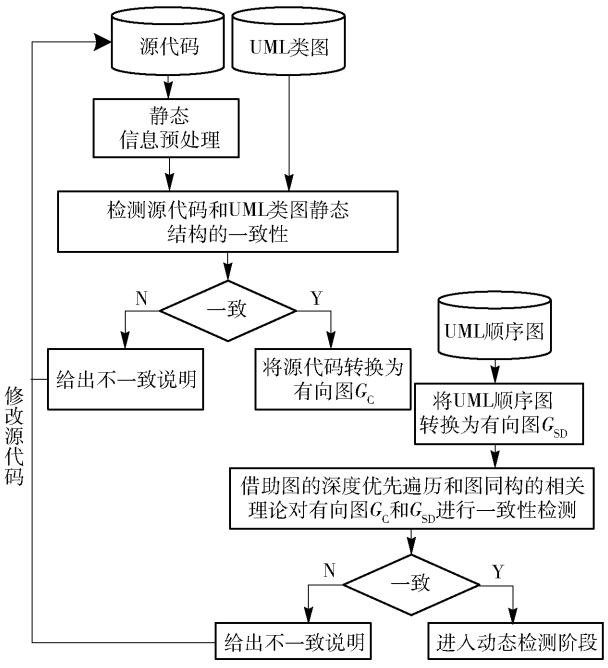

图 1　一致性静态检测基本流程

2.2　源代码静态信息预处理

C + + 代码的静态信息有以下 2 类.

1）静态结构信息. 包括类的个数、类名、成员变量序列、成员函数序列.

2）静态行为信息. 主要是指函数调用关系.

C + + 代码静态信息提取的主要思想是通过词法分析和语法分析来获取类的个数、类名、成员变量序列、成员函数序列以及函数调用关系等信息，并将这些信息存储起来.

2.3　静态结构一致性检测

根据获取到的静态结构信息，依次检测源代码和 UML 类图类的个数、类名、成员变量序列（变量名、可见性、类型）、成员函数序列（函数名、可见性、返回类型、参数列表）的一致性. 若存在不一致，则给出不一致说明，并手动修改源代码，直到源代码的静态结构与 UML 类图完全一致.

2.4　静态行为一致性检测

UML 模型和源代码缺乏精确的语义，使得利用计算机算法自动检测 UML 模型和源代码的静态行为一致性变得困难. 将 UML 模型和源代码的静态行为转换为有向图的形式是一种有效的解决方案. 核心思想是将消息（函数调用）用有向图的顶点来表示，消息（函数调用）的时间顺序用顶点之间的边来表示，并用三元组

（tClassName，rClassName，msgName）

标记每个节点. 其中 tClassName 表示消息发送对象所属类的名称，rClassName 表示消息接收对象所属类的名称，msgName 表示消息的名称. 此外，添加一个初始节点和若干个终止节点.

下面以笔者所在实验室研发的 UML 模型正确性检测工具 UMLChecker 1.0 为例来说明 UML 顺序图和源代码到有向图的转换. 为了便于说明问题,这里仅考虑模型导入模块. 该模块的主要功能是对导入 XML 格式的 UML 文件进行解析,从中获取类以及类之间的关系信息,并在显示区绘制出模型. 该模块的类图(未列出成员变量和成员函数)和顺序图分别如图 2 和图 3 所示.

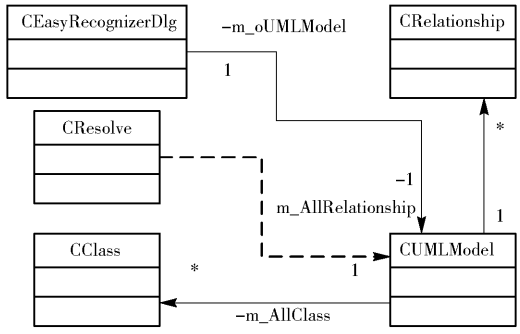

图 2 模型导入功能的 UML 类图

根据上述转换规则,可得 UML 顺序图对应的有向图 G_{SD} 和源代码对应的有向图 G_C 分别如图 4(a)和 4(b)所示.

将待检测 UML 顺序图和源代码转换为有向图后,就可以利用图的深度优先遍历和图同构判定算法对 UML 顺序图和代码进行静态行为一致性检测. 从开始节点出发同时对 UML 顺序图对应的有向图和源代码对应的有向图进行深度优先遍历,依次比

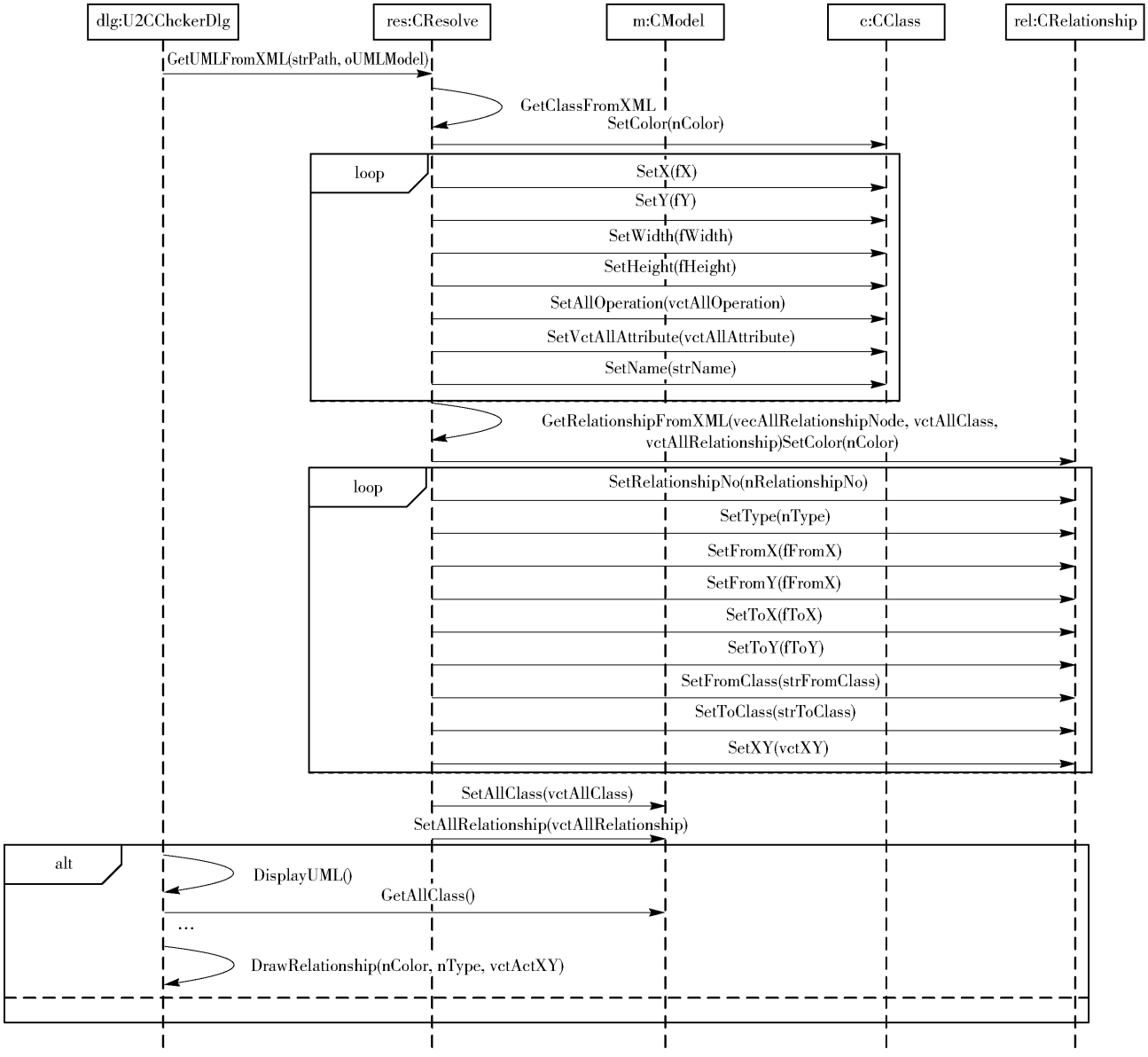

图 3 模型导入功能的 UML 顺序图

图 4　顺序图的有向图和源代码的有向图

较二者当前访问节点的标记是否一致. 此外,还需要判断已访问节点和当前访问节点及其之间的边构成的 2 个有向图是否同构(这里使用文献[9]中的图同构判定算法进行判断).

下面以 UMLChecker 1.0 为例来说明 UML 顺序图和源代码的静态行为一致性检测. 当遍历到图 5 所示的有向图 G_{SD} 和 G_C 的节点 10 时,由已访问节点和当前节点构成的有向图 G'_{SD} 与 G'_C 不同构,则输出不一致信息“UML 顺序图(CResolve, CClass, SetName)与源代码(CResolve, CClass, SetName)不一致”. 事实上,源代码将本不需要放入 for 循环中的函数调用 oClass. SetColor(BLACK)也放入了 for 循环中. 文献[6]中提出的 SD-CG(sequence diagram call graph)虽然可以表示程序的顺序结构,却无法表示循环结构和选择结构,因此无法发现该不一致.

修改源代码后,重新遍历到 G_{SD} 节点 11 时,其标识为(CResolve, CResolve, GetRelationshipFromXML),而 G_C 相应节点的标识为(CResolve,

CUMLModel, SetAllClass),则输出不一致信息“UML顺序图(CResolve, CResolve, GetRelationshipFromXML)与源代码(CResolve, CUMLModel, SetAllClass)不一致”.

3 UML模型与代码一致性动态检测

在对UML模型与代码进行一致性静态检测后,接着进行一致性动态检测,以提高检测的精确率.

UML模型与代码一致性动态检测的基本流程是:首先将UML顺序图转换为FSM(借鉴了文献[10]中的转换方法,并对其进行了细化);然后监视正在分析程序的函数调用,并将它们与UML顺序图转换得到的FSM进行匹配. 若匹配不成功,则说明存在不一致,修改源代码,直到源代码的动态行为与UML顺序图完全一致.

UML顺序图到FSM的详细转换方法以及UML模型与代码动态一致性检测的详细算法将在后续的研究中进行讨论.

4 实验及结果分析

目前提出方法的支撑工具U2CChecker 1.0已经实现. 该工具采用MFC开发,导入UML模型、源代码及方法调用跟踪,输出不一致信息. 为说明提出方法的有效性,使用文献[6-8]方法和提出方法对UMLChecker 1.0进行了UML模型与源代码一致性检测. 表1列出了使用不同方法检测出的不一致个数.

表1 检测出的不一致个数

检测阶段	文献[6]方法	文献[7]方法	文献[8]方法	提出方法
静态结构一致性检测	12	12	12	12
静态行为一致性检测	5	—	—	13
动态行为一致性检测	—	—	—	6
总计	17	12	12	31

由表1可知,新方法可以对UML模型与代码的一致性进行检测,且具有较高的检测精度. 对于静态行为和动态行为的一致性检测,新方法的检测精度明显提升.

5 结束语

现有的UML模型一致性检测方法大多只针对UML模型本身的一致性检测或由UML模型自动生成一致性代码,也有一些方法针对UML模型与代码的一致性检测,但是检测精度不高. 所提出的方法通过图深度优先遍历和图同构判定算法检测静态结构与静态行为的一致性,并借助FSM的验证技术检测动态行为的一致性,具有较高的检测精度. 对于静态行为和动态行为的一致性检测,检测精度明显提升.

目前该方法只能检测出UML模型与代码存在的不一致,而不能自动对存在的不一致进行修改. 如何对检测出的不一致进行自动修改是下一步研究的重点.

参考文献:

[1] Ekanayake E M N K, Kodituwakku S R. Consistency checking of UML class and sequence diagrams[C]//8th International Conference on Ubi-Media Computing. Jeju: IEEE, 2015: 98-103.

[2] 钱成, 燕雪峰, 周勇, 等. 基于状态约简的顺序图和状态图一致性检测[J]. 计算机应用研究, 2014, 31(5): 1452-1455.
Qian Cheng, Yan Xuefeng, Zhou Yong, et al. Model checking consistency of sequence diagram and state machine based on state reduction[J]. Application Research of Computers, 2014, 31(5): 1452-1455.

[3] Straeten R V D, Mens T, Simmonds J, et al. Using description logic to maintain consistency between UML models[C]//Lecture Notes in Computer Science. Seattle: Springer Berlin Hei-delberg, 2003: 1-15.

[4] Long Q, Liu Z, Li X, et al. Consistent code generation from UML models[C]//16th Australian Conference on Software Engineering. Brisbane: IEEE Computer Society, 2005: 23-30.

[5] Chama W, Elmansouri R, Chaoui A. Model checking and code generation for UML diagrams using graph transformation[J]. International Journal of Software Engineering & Applications, 2012, 3(6): 39-55.

[6] 曾一, 李函逾, 刘慧君, 等. UML模型和Java代码之间的一致性检测方法[J]. 计算机科学, 2015, 42(4): 151-155.
Zeng Yi, Li Hanyu, Liu Huijun, et al. Consistency detection method between UML model and Java source code[J]. Computer Science, 2015, 42(4): 151-155.

[7] Pires W, Ramalho F. UML-based design test generation[C]//ACM Symposium on Applied Computing. Fortaleza: DBLP, 2008: 735-740.

[8] Chavez H, Shen W, France R, et al. An approach to

checking consistency between UML class model and its Java implementation [J]. IEEE Transactions on Software Engineering, 2016, 42(4): 322-344.

[9] 李锋, 陆韬. 任意图同构判定及其应用[J]. 复旦学报(自然科学版), 2006, 45(4): 480-484.
Li Feng, Lu Tao. Isomorphism judgment of arbitrary graphs and its application [J]. Journal of Fudan University, 2006, 45(4): 480-484.

[10] Wendehals L, Orso A. Recognizing behavioral patterns at runtime using finite automata [C] // Proceedings of the 2006 International Workshop on Dynamic Analysis. Shanghai: ACM, 2006: 33-40.

(上接第 37 页)

[10] Liu J, Wang Z, Yao M, et al. VN-APIT: virtual nodes-based range-free APIT localization scheme for WSN[J]. Wireless Networks, 2016, 22(3): 1-12.

[11] Lasla N, Younis M F, Ouadjaout A, et al. An effective area-based localization algorithm for wireless networks [J]. IEEE Transactions on Computers, 2015, 64(8): 2103-2118.

[12] Yaghoubi F, Abbasfar A A, Maham B. Energy efficient RSSI-based localization for wireless sensor networks[J]. IEEE Communications Letters, 2014, 18(6): 973-976.

[13] Tateishi, Kazuya, Ikegami, et al. Estimation method of attenuation constant during localization in RSSI [J]. International Journal of Radiation Oncology Biology Physics, 2015, 91(5): 1026-1033.