动静态信息相结合的 UML2.0 序列图逆向生成方法*)

梁若莹　陈　平　胡圣明　刘鹏飞
(西安电子科技大学软件工程研究所　西安 710071)

摘　要　符合 UML2.0 标准的序列图在 UML1.x 序列图的基础上添加了控制流信息。为此,本文提出一种基于动静态信息相结合的 UML2.0 序列图逆向生成方法。该方法首先利用目标程序运行时的动态信息产生基本的方法调用序列,然后依据静态的程序依赖图对其进行补充和调整,在其上添加方法间的逻辑关系,使产生的序列图带有控制流信息。这种方法生成的序列图符合 UML2.0 标准,可以很好地辅助用户理解目标程序。

关键词　逆向工程,程序理解,序列图,UML2.0

Reverse Engineering of UML2.0 Sequence Diagrams Using Static and Dynamic Information

LIANG Ruo-ying　CHEN Ping　HU Sheng-ming　LIU Peng-fei
(Software Engineering Institute, Xidian University, Xi'an 710071, China)

Abstract　Compared with UML1.x, UML2.0 introduces the flow of controls to sequence diagrams. A method to reverse engineer UML sequence diagrams using dynamic information combined with static information is presented in this paper. In the proposed method, basic sequence of method calls is first generated from dynamic information obtained during the runtime of the target program, and then added with the flow of controls (i.e. the relationship of method calls) based on the static dependency graph. The sequence diagram reverse engineered by this method is self-contained and compatible with the UML2.0 standard, and can efficiently help users to understand the target program.

Keywords　Reverse engineering, Program understanding, Sequence diagrams, UML2.0

1　引言

统一建模语言(UML)已经成为对软件的结构和行为进行建模的标准[1],其中序列图是展现软件系统行为的关键模型。该模型显示了一系列对象相互作用的顺序和交换的信息、描述对象间发送的消息以及这些消息如何合作完成特定的功能[2]。

面向对象程序最难理解的部分就是对象间的交互[3]。程序阅读者很难直接从代码中理解一组相关对象通过相互通信完成的任务,而序列图结构化地展现了对象间传递的消息,有序地将对象间的通信组织在一起,对程序理解有很大的帮助。所以在逆向工程(Reverse Engineering)中,序列图的逆向生成是非常重要的。用户可以通过逆向生成出的序列图,深入理解软件的功能和行为特点,从另一个角度理解对象间的关系。在此基础上,用户能够进一步对软件进行处理,如系统的维护和升级、系统文档的维护和恢复、再工程以及系统的移植等。

当前逆向生成的序列图大多是面向 UML1.x 标准的。在 UML1.x 标准中,序列图着重显示对象间交互的时序关系,而为了显示迭代控制、条件控制以及其它各种行为控制,UML2.0 标准对序列图增加了交互框架图示法(interaction frame)。该图示法引入了组合框架这个符号元素,一个组合框架用来把一套消息组合在一起,在一个序列图中显示条件分支。组合框架分为变体、选择项和循环,分别代表一种结构:变体用来指明在两个或更多的消息序列之间的、互斥的选择;选择项用来为消息序列建模,序列在一定条件下或者会执行,或者不执行;循环用来指明一段重复的消息序列。交互框架图示法为序列图引入了控制流信息,更完全地显示了程序全貌[2]。为此,本文提出利用目标程序的动态信息——程序运行信息和静态信息——程序依赖图,结合用户交互生成目标程序的序列图,使之符合 UML2.0 标准、具有更好的可读性。

本文第 2 节介绍序列图逆向生成的相关研究,第 3 节详细描述动静态信息相结合的 UML2.0 序列图逆向生成方法,第 4 节通过实验证明该方法的有效性,最后进行总结。

2　相关研究

2.1　基于动态信息的方法

基于动态信息逆向生成序列图[4,11]的方法简单直接,其主要思路是保存程序的执行路径,并将其转换成对应的序列图。该类方法的优点是速度快、正确率高,并且允许用户自由选择执行路径。尤其对于面向对象程序来说,多态和动态绑定是基于静态信息生成序列图方法很难解决的问题,而基于动态信息的方法不存在这样的问题。但是这类方法生成的序列图存在的问题是内容不完备,不能进行进一步的优化,没有控制流信息,仅适用于生成 UML1.x 序列图。

2.2　基于静态信息的方法

根据静态信息逆向生成序列图的方法[3,6-10,12]适用面广,可以应用于局部程序甚至是无法执行的程序,其主要思路是从种类齐全的静态信息中提取与方法调用相关的信息生成序列图。这类方法生成的序列图内容完备、易于优化、可以产生控制流信息,但同时也存在多方面的不足,例如无法恢复出控制流的条件信息、粒度只能达到类级别、无法确定对象、不适

*)国家教育部博士点基金(项目编号:20030701009);面向对象逆向工程工具研究;国家自然科学基金(项目编号:60473063);研究生创新基金(编号 05009)。**梁若莹**　硕士研究生,主要研究方向为逆向工程、程序理解、面向对象和软件体系结构。

合大型软件系统、程序入口和主要的路径难以确定等等，而且丧失了序列图最主要的特点：时序性，也不能生成 UML2.0 序列图。

由上述分析可以看出，单纯的依赖于动态信息或静态信息的方法虽然能在一定程度上逆向生成目标程序的序列图，但是均存在一定的不足，不能生成符合 UML2.0 的序列图。因此，本文提出以目标程序动态运行信息为主，静态信息提供依据，两者结合恢复序列图的方法，其结果符合 UML2.0 标准，结构清晰，内容完备，条理分明，弥补了以前序列图生成方法的不足。

3 动静态信息相结合的 UML2.0 序列图逆向生成方法

为了逆向生成符合 UML2.0 标准的序列图，必须借助静态信息。但完全根据静态信息生成的序列图没有主要路径，难以整理和理解，所以必须以动态信息为主线产生序列图的主要路径。本文方法首先对目标程序进行植入和运行分析，获取其动态信息（程序运行时的方法调用序列），并将该信息记录在动态信息文件中；然后，对目标程序进行静态分析，获得反映程序结构的静态程序依赖图（包括实体间的调用关系及函数体的结构特点）；最后，以动态信息为主线，在静态程序依赖图中搜索合适的信息，将两者有机结合，生成序列图。

3.1 动态信息文件

动态信息文件按时序保存跟踪目标程序的执行所产生的信息，是生成序列图的主要依据。该文件结构为：

File = DATA

DATA = Package1 Package2

Package1 = Package3

Package2 = Package4 | Package4 Package2

Package3 = CALL1 | CALL1 Package3

CALL1 = 〈CALL〉〈CALL/〉〈/CALL〉

CALL = ClassName ObjectName MethodName [Package]

Package4=CALL2 | CALL2 Package4

CALL2=〈CALL〉〈CALL/〉 | 〈CALL〉CALL2〈/CALL〉

一个动态信息文件中的数据包括两种 Package 结点。第一种 Package 结点 Package1 中包含一个 Package3 节点，Package3 包含一个或多个 CALL1 结点，每个 CALL1 结点深度为 2，有一个孩子结点，代表父结点对子结点的方法的一次调用；第二种 Package 结点 Package2 由一个或多个 Package4 组成，Package4 由一个或多个 CALL2 结点组成，每个 CALL2 结点的深度大于等于 1，可以没有孩子结点，也可以有多个孩子结点。父结点与其子结点之间是调用关系，每个 CALL 表示正在执行对象 ObjectName（类为 ClassName）的方法 MethodName。

在该文件中，Package3 保存进程间通信的数据，CALL1 结点代表进程间通信执行的一个激励。每个 Package2 结点和 Package4 结点都有一个 Name 属性，Package2 的 Name 属性记录一个进程号，Package2 对应这个进程内部交互及其与外部通信的数据。Package2 结点中包含多个 Package4 结点，Package4 结点的 Name 属性记录一个包名，Package4 表示 Package2 进程在这个包层次上的动态信息，记录了这个包内部交互及其与外部通信的数据。Package2 结点包含了一个进程内从最高层的包间的交互信息到最底层的对象间的交互信息，是个逐步细化的过程。

动态信息文件和反映目标系统动态行为的序列图的对应关系如图 1 所示。

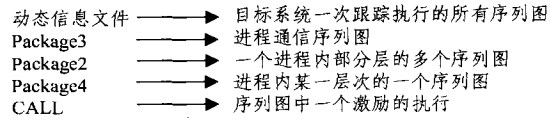

图 1 动态信息文件和序列图的对应关系

3.2 静态程序依赖图

原始静态程序依赖图（Dependency Graph）为二元组：*DependencyGraph* = 〈*N*, *E*〉，其中：*N* 为图的顶点：*N* = 〈*ClassName*, *MemberName*, *MemberType*〉，*E* 为顶点间的依赖关系，*E* = 〈*N*, *N*, *DependencyType*〉，如图 2 所示。顶点的类型有：全局方法、类的操作、类的属性和系统调用。顶点间的依赖关系包括：方法调用、属性读、属性写。构造静态程序依赖图是直接从程序代码中得到各实体间的相互调用关系，具有全面、完整、普遍的特点，主要关注方法间的调用关系，对于属性的读写暂不作考虑。

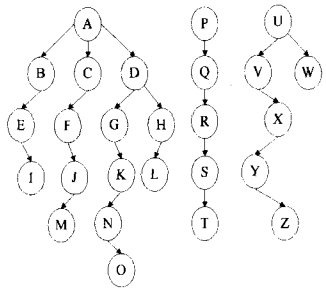

图 2 原始静态程序依赖图

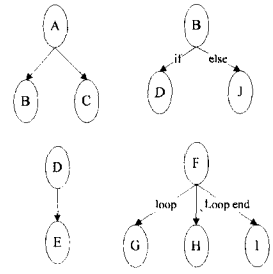

图 3 静态程序依赖图

为了符合实际需求，本文所使用的静态程序依赖图不仅包含上述依赖关系，还要显示方法内部的具体结构，包括顺序、分支和循环，如图 3 所示。图 3 中的方法 B 与方法 C 由方法 A 顺序调用；方法 D 和 J 是分支的关系；方法 G，H 和 I 组合成一个循环体。在原始的静态程序依赖图中加入这三种基本结构，就可以表示出方法调用之间的逻辑关系。这些附加信息仅是辅助信息，并不影响依赖图的整体结构。

3.3 整体处理流程

本文方法整体处理流程如图 4 所示。

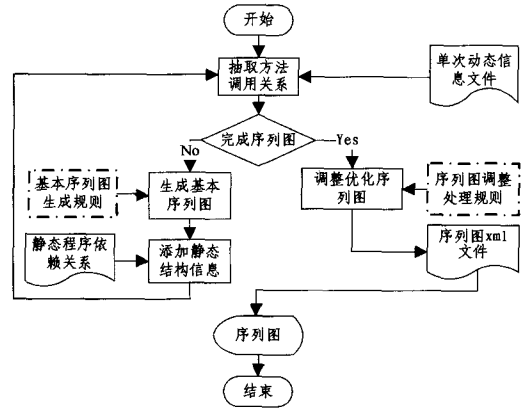

图 4 序列图逆向生成过程示意图

（1）对收集到的动态信息文件进行处理，抽取其中的方法调用序列；

（2）基于抽取的方法调用序列，根据基本序列图的生成规

则生成基本序列图；

(3)在基本序列图的基础上添加静态程序依赖关系文件中的逻辑关系，即控制流信息；

(4)将添加了控制流信息的方法调用关系信息提取出来；

(5)按照序列图调整规则，对得到的方法调用信息进行调整优化；

(6)将调整后的方法调用信息保存为 XML 文件，至此已恢复出了目标程序的序列图；

(7)用 UML 工具显示序列图 XML 文件。

3.4 实例分析

本小节以一个实例来进一步形象地说明本文方法。图 5 是一个以 XML 格式存储的动态信息文件，该文件描述了目标程序 Test 的一次执行过程(进程号为 2464)，这次执行共涉及 4 个对象。根据动态信息文件生成的原始序列图如图 5 所示。此序列图清晰地显示了目标程序执行时的方法调用序列，以及对象间的调用关系。

```
〈DATA〉
 〈Package〉
  〈Package Name=“Total”/〉
 〈/Package〉
 〈Package Name=“2464” ProcessID=“Test”〉
  〈Package Name=“Total” ProcessID=“”〉
   〈CALL ClassName=“”ObjectName=“对象 1” MethodName=
“A()”〉
    〈CALL Class Name=“OjbectName”=“对象 3” MethodName=
“B()”〉
     〈CALL ClassName=“” ObjectName=“对象 4” MethodName
=“D()”〉
      〈CALL ClassName=“”LbjectName=“对象 2” MethodName
=“E()”/〉
     〈/CALL〉
    〈/CALL〉
    〈CALL ClassName=“”ObjectName=“对象 3” MethodName=
“C()”/〉
   〈/CALL〉
  〈/Package〉
 〈/Package〉
〈/DATA〉
```

图 5 动态信息文件

原始序列图仅按时序显示了各实体间的方法调用，不包含这些方法之间的控制流信息。利用图 3 的静态程序依赖图的信息就可以弥补这个不足。由于原始序列图和静态程序依赖图中都包含方法调用信息，因此很容易将二者结合，得到目标程序的 UML2.0 序列图，如图 7 所示。

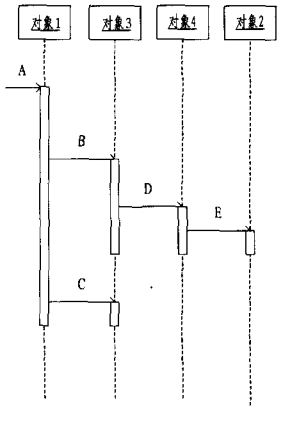

图 6 原始序列图

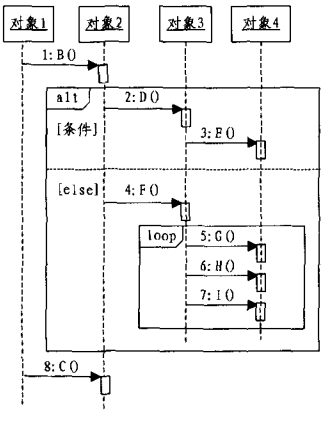

图 7 UML2.0 序列图

4 实验研究

4.1 实验用例描述

实验用例选取的是一个停车场仿真系统。该系统属于典型的 MFC 程序，用以仿真某停车场一段时间内的运营情况，包括各种车辆的进出(每辆车都有一个卡号，在进出停车场时，需对其卡号进行合法性验证)、停放、驶离，以及突发事件处理等。在进行仿真时，系统根据输入的各种参数，仿真停车场的运营情况并即时以图形显示和数据文件等形式输出给用户。用户可以更改停车场的车位数，也可以改变职工数(用来确定产生汽车的卡号)。

4.2 实验结果及分析

图 8 显示了仅根据动态信息生成的 UML1.4 序列图，该图显示了程序运行时的函数调用关系序列。图 8 上半部分中，第 16 条消息，函数 setCarDistance()，是一个控制汽车位置的函数，然而该函数所完成的功能从这个序列图中是无法得到的，只能得出结论这个函数结束后会调用第 17 条消息，函数 setEndDis()。而该图的下半部分中，函数 setCarDistance()(第 84 条消息)结束后先调用了函数 IsHorizontalClashing (acar, another)、IsHorizontalVerticalClashing (acar, another) 及 IsVerticalClashing (acar, another)，最后才调用了函数 setEndDis()(第 88 条消息)。用户根据该序列图不能理解函数 setCarDistance()和函数 setEndDis()的调用关系，以及这个调用序列的形成过程。

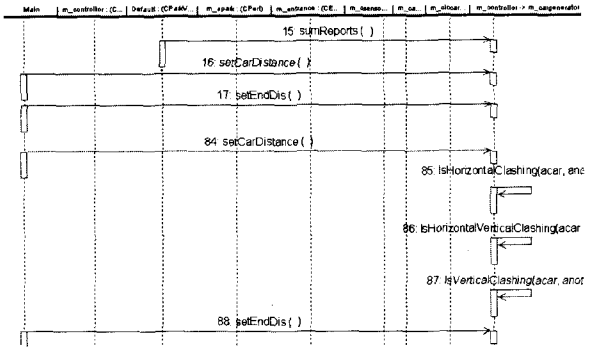

图 8 原始序列图

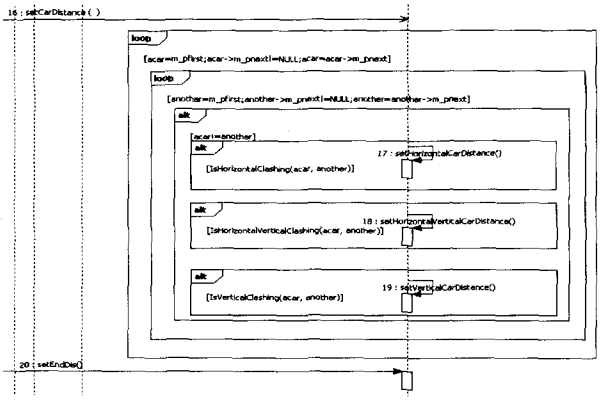

图 9 修改后生成的序列图

图 9 所示为动静态信息相结合后产生的符合 UML2.0 标准的序列图模型的实验结果。图中的矩形区域即 UML2.0 标准增添的组合框架。框架中 loop 代表循环，alt 代表选择项。该图中的消息 16 和消息 20 分别对应于图 9 的消息 16 和消息 17，这两条消息之间的内容即为函数 setCarDistance()的具体执行内容。函数 IsHorizontalClashing (acar, another)，IsVerticalClashing (acar, another) 及 IsHorizontal VerticalClashing (acar, another)是循环体中条件语句的一部分。

如果该循环体没有执行，就出现了图 8 消息 16、17 所示的调用关系；如果循环体的判断条件被满足，而条件语句的判断条件不满足，调用关系就如图 8 中消息 84 至 88 所示。从修改后的序列图中很容易理解到 setCarDistance 函数的作用是将停车场中所有的车遍历一遍，以不冲突为前提放置场内汽车。IsVerticalClashing(acar, another) 等函数和函数 setCarDistance()的关系完全体现出来，逻辑关系清楚。

需要注意的是，在图 9 中的选择项中，有的判断条件本身就是函数。为了避免重复显示造成混淆，在生成序列图时将这种函数视为条件，不视为消息，也不进行消息编号。

从实验分析中可以看出，基于动静态信息相结合逆向生成的序列图最小粒度为对象，并且深入函数内部，表现出了函数间的深层调用关系，而不仅仅是函数的调用顺序。所以，基于动态信息和静态的程序依赖关系生成的序列图更为合理和全面，有助于用户对目标程序的理解。

结束语 在 UML2.0 序列图中，不但要表现出对象间消息传递的执行序列，还要体现出消息间的控制流，这给序列图的逆向生成带来了难度。本文结合本项目组开发的逆向工程工具集 XDRE 的功能特点，提出一种新的动静态信息相结合的 UML2.0 序列图逆向生成方法。该方法将程序执行所收集到的动态信息与静态程序依赖图相结合，动态信息保证序列图的有效性，静态信息保证序列图的完备性。实验表明，该方法生成的序列图易于理解、符合实际模型、主线清晰、信息全面，适合于实际应用。

参 考 文 献

[1] Rountev S R. A Interactive Exploration of UML Sequence Diagrams. Visualizing Software for Understanding and Analysis, 2005

[2] Albir S S. UML in a NutShell. O'Reilly, 1998

[3] Rountev A, Connell B H. Object naming analysis for reverse-engineered sequence diagrams // International Conference on Software Engineering. 2005: 254-263

[4] Ghezzi C, Jazayeri M, Mandrioli D. Fundamentals of Software Engineering. Prentice Hall International Ed. 1991

[5] Walker R J, Murphy G C, Freeman-Benson B, et al. Visualizing Dynamic Software System Information through High-level Models // Proceedings of the Conference on Object-oriented Programming, Systems, Languages, and Applications. 1998: 271-283

[6] DePauw W, Jensen E, Mitchell N, et al. Visualizing the execution of Java programs. Software Visualization // LNCS 2269. 2002: 151-162

[7] Oechsle R, Schmitt T. JAVAVIS: Automatic program visualization with object and sequence diagrams using JDI. Software Visualization // LNCS 2269. 2002: 176-190

[8] Tonella P, Potrich A. Reverse engineering of the interaction diagrams from C++ code // International Conference on Software Maintenance. 2003: 159-168

[9] Rountev A, Volgin O, Reddoch M. Static control-flow analysis for reverse engineering of UML sequence diagrams // The 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 2005: 96-102

[10] Systa T, Koskimies K, Muller H. Shimba an environment for reverse engineering Java software systems. Software Practice & Experience, 2001, 31(4): 371-394

[11] Richner T, Ducasse S. Using Dynamic Information for the Iterative Recovery of Collaborations and Roles // 18th IEEE International Conference on Software Maintenance (ICSM'02). 2002: 34-43

[12] Kollmann R, Gogolla M, Capturing Dynamic Program Behavior with UML Collaboration Diagrams // The 5th European Conf. on Software Maintenance and Reengineering. Los Alamitos, 2001: 58-67

(上接第 246 页)

其中，x_7，x_8 连续属性，其它均是离散的。根据 2.4 分离连续和离散的属性。得到规则集 4，例如，上面规则(15)，将 x_{19} x_{20} x_{21} x_{23} x_{25} 代入，得

If $(0.57 \leqslant 0.2x_7 - 0.49x_8 \leqslant 2.06) \wedge x_{19}=0 \wedge x_{20}=0 \wedge x_{21}=1 \wedge x_{23}=0 \wedge x_{25}=1$

then $t=1$

结合问题的原始属性输入，上面的规则很容易地转换成：

If $(0.57 \leqslant 0.2 * \text{Trestbps} - 0.49 * \text{chol} \leqslant 2.06)$ ∧ ca is 2 ∧ thal is rever

then $t=1$.

其中，Trestbps 代表 resting blood pressure，chol 代表 cholesteral，ca 代表 the number of vessels colored.

结束语 针对带混合类型属性的问题，目前的规则提取方法所提取的规则都没有区分连续属性与离散属性。本文所提出的规则提取方法，在规则的表示形式上，将连续属性与离散属性区分，使得所提取的规则更易于理解。对于连续属性的相关条件，用相应子空间的划分即子空间的超平面来表示，对离散属性，说明其离散取值。然而，一般情况下，这种方法会使得所提取出的规则数比一般方法的多。因为，我们在分离的时候，对离散属性采取了组合试探的方法。例如，对规则(15)进行分离时，对 5 个变量的取值组合和一个线性不等式所组成的空间进行搜索，可能会有多个搜索点满足期望(∽即 t=1)∽，则一条规则变成了多条。在规则集规模和预测精度上都有待进一步研究和改进。

参 考 文 献

[1] Andrews R, Diederich J, Tickle A. Survey and critique of techniques for extracting rules from trained artificial neural networks[J]. Knowledge Based Systems, 1995, 8 (6): 373-389

[2] Alexander J A, Mozer M C. Template-based procedures for neural network interpretation. Neural Networks, 1999, 12(3): 479-498

[3] Gallant S I. Connectionist expert systems. Communications of the ACM, 1988, 31(2): 152-169

[4] Fu L M. Rule generation from neural networks[J]. IEEE Transactions on Systems, Man and Cybernetics, 1994, 28 (8): 1114-1124

[5] Fu L M. Rule learning by searching on adapted nets // Proc. 9th National Conf. on Artificial Intelligence. 1991: 590-595

[6] Towell G G, Shavlik J W. Extracting refined rules from knowledge-based neural networks. Machine Learning, 1993, 13(1): 71-101

[7] Lu H, Setiono R, Liu H. NeuroRule: A Connectionist Approach to Data Mining // Proceedings of the 21th International Conference on Very Large Data Bases. September 1995: 478-489

[8] Setiono R, Liu H. Neurolinear: From neural networks to oblique decision rules. Neurocomputing, 1997, 17(1): 1-24