软件逆向工程中动态剧情抽象新方法*)

李 凡 李青山 陈 平
(西安电子科技大学软件工程研究所 西安 710071)

摘 要 研究了逆向工程中动态剧情的模式发现以及抽象问题。提出并实现了动态剧情中交互模式的自动发现、交互层次的自动恢复以及基于类图的设计模式识别,并实现了以此为依据对动态剧情的抽象。同时,使用 Rational Rose 的扩展机制,将以上功能无缝嵌入到 Rose 开发环境中,从而使逆向工程分析工具 XDRE 具备了在可视环境下以不同抽象层次、不同侧面观察和分析目标系统行为的功能。

关键词 逆向工程,UML,序列图,模式发现

Dynamic Interaction Information Oriented Pattern Discover Technology in Reverse Engineering

LI Fan LI Qing-Shan CHEN Ping
(Software Engineering Institute, Xi'dian University, Xi'an 710071)

Abstract This paper emphasizes on the research of pattern discover and scenario abstraction of dynamic interaction information in software reverse engineering. Under the background of the development of the tool for reverse engineering analysis named XDRE, methods for discovering interaction patterns, recovering interaction levels in sequence diagrams and design pattern detection are present respectively. With these methods, abstraction of interactive messages in sequence diagrams can be performed. Furthermore, through the extendibility of Rational Rose, the abstract methods are implemented and seamlessly integrated into Rose development environment.

Keywords Reverse engineering, UML, Sequence diagram, Pattern discover

软件行为分析是软件逆向工程研究领域的一个重要方法。目前,逆向工程工具都具有不同程度的行为分析能力,获取系统实际运行中的动态剧情[1]。但是,当被分析的目标系统比较复杂以及随着动态监控时间的延长,由动态信息生成的序列图会在水平和垂直方向快速地增长;同时,随着动态分析的持续进行,动态信息中往往包含着同一交互模式的重复出现,影响系统执行效率,而且不利于使用者对序列图的理解。因此,对反映系统行为的动态剧情进行抽象是十分必要的。目前已发表的逆向工程研究中,支持动态剧情抽象的还很少。文[2]中提出了基于精确匹配、可包含匹配和可间隔匹配的消息抽象方法,文[3]中提出了机遇交互对象组合的抽象,但它们都要通过用户交互指定待抽象的消息序列模式或对象。我们曾在文[4,5]中分别介绍了基于交互和基于类层次的交互,该方法已应用在逆向工程开发工具 XDRE[6] 中。但对于消息序列图中经常出现的序列重复和序列嵌套,还没有自动识别和抽象的能力。随着在 XDRE 工具的开发研究中的不断深入,本文提出并实现一种自动发现交互模式、交互层次方法,并以此为依据对序列图进行抽象,同时提出了通过识别类图中的设计模式,从而进行动态剧情抽象的新方法。

1 动态剧情中的模式识别方法

1.1 交互模式的识别

针对动态剧情中的交互消息序列,交互模式是指交互消息序列中的重复序列,通过发现序列图中重复的消息序列,并使用一条消息在其出现的位置代替被发现的消息序列[7],从而简化序列图的视图,有利于抓住主要流程,发现系统设计规律。

首先从第一条消息开始的序列片断入手,顺序遍历整个序列图,找出该序列片断重复出现的所有位置,形成一个集合;然后将该集合中的所有序列在其原有长度上加 1。如果仍有相同的,则将相同的序列形成一个新集合。剩下的恢复原来长度,并形成另一个集合。在得到生长的所有集合上重复以上操作。直到没有产生序列更长的集合为止。以当前处理过的下一条消息为起点,重复以上操作,如果该消息已经被作为序列的起点分析过,则应跳过。如此,直到结束。

下面给出消息重复模式自动发现算法的形式化描述。

算法 1 交互重复序列的发现

```
输入:消息交互序列 MessageQueue
输出:交互重复模式集合 PatternResult
ForEach Message from MessageQueue
{
  if(Message 已作为过模式的首消息)then 跳过该消息;
  end if
  PatternQueue=CreateMessageList()//条件是 List 中的消息为是
     Message 的最短序列片断
  do//pattern 的生长、分裂
  {  将各 Pattern 延长一条消息;
     将 PatternList 中的相同的 Pattern 各自分组;
     将未得到分组的 Pattern 退回一条消息,并分为一组,
     该组不再参与生长、分裂
  }while(产生了新的分组)
  LocalHarvest()//以所指定的原则整理本次结果,到 PatternResult
```

*)国家自然科学基金(项目编号:60473063)、国家教育部博士点基金(项目编号:20030701009)及"十五"国防预研项目(项目编号:41306060106)、"十五"军事电子预研重点课题"C3I 系统应用软件逆向工程开发工具的研究"(编号:413060601)。李 凡 博士研究生,研究兴趣为软件逆向工程、人工智能应用;李青山 博士、副教授,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;陈 平 博士、教授、博士生导师,主要研究领域为电子信息系统软件开发理论与技术、面向对象技术、软件工程、逆向工程。

```
}
GlobeHarvest()//以所指定的原则整理全局结果
```

在算法 1 中，LocalHarvest() 函数完成单次循环后对 Patternlist 中生成的模式进行筛选，并加入到模式容器 PatternResult 中。筛选的目的是把长度不满足要求（即指定的最小模式长度）的模式以及经过生长后出现次数等于 1 的模式删除掉。GlobeHarvest()将结果集中的所有模式进行一次检查，消除冲突和不合要求的模式，并在 MessageQueue 中出现各模式的地方进行标记，以备模式和抽象图的输出。

在输出抽象后的序列图时，各模式之间往往会同时占有某几条消息，这时就必须有一种规则来选择模式。目前可用的规则有：模式长度优先、出现次数优先和先出现优先，不同规则会出现不同的抽象效果。

1.2 交互层次关系的发现

如前所述，本文所研究的动态剧情是由对目标系统的执行过程进行动态分析而自动产生的，反映了系统运行期间对象之间的方法调用关系和调用顺序。当对象 A 调用了对象 B 中的方法 B. m1，即一次消息调用，而在 B. m1 执行期间又调用了对象 C 的方法 C. m1。如此下去，形成了对象调用之间的层次关系。利用这种层次关系，可以实现对序列图的基于交互层次的抽象，从而屏蔽细节过程，突出核心流程。

但是，由动态分析获得的序列图，由于分析方法的不同，不总是可以完整地获得每一次消息调用的返回情况。同时，在 Rose 的扩展接口中没有提供对消息调用层次的支持，所以在此环境上生成的序列图没有包含层次抽象所必需的消息层次信息。因此，在进行层次抽象之前，必须对序列图的消息层次关系的完整性进行检查，并设法恢复出不完整的部分。

本文提出一个消息调用层次的恢复算法。其主要思想是这样的：在引言部分所给出的进程内部交互序列图中，程序是按照调用顺序依次执行的，而在序列图中，体现为第 n 条消息的接收方是第 $n+1$ 条消息的调用方；一旦出现第 $n+1$ 条消息的发送方不是第 n 条消息的接收方，则必然在第 n 条消息后，程序的执行点返回到了第 $n+1$ 条消息的发送方。我们将第 n 条消息所在的位置称为该消息序列的一个折返点，这些折返点体现了消息调用的返回时机。同时，各折返点在此之前必然有其出发点。结合各出发点和折返点，就可以恢复出一部分消息层次。进一步观察序列图发现，有以下几种情况必须考虑进去：

首先，如果消息的发送方调用了自身，同时下一条消息仍由该对象发送，此时该对象能获得执行点有可能是因为被调用，但也有可能是上一次调用得到了返回，所以无法直接判断是否发生过返回。此时可以发现各出发点和折返点成对组合后，各对之间不能有交叉（可以是包含）。利用这一规律，先假设发生了返回，如果由此引起交叉，则假设不成立。

其次，对于所发现的出发点以外的消息调用，没有直接的返回点与之对应，这是因为这些消息返回后直接又返回到了更高层的消息调用，所以可以使用其高层消息调用的返回点，作为其返回点。

最后，对于进程间交互，各进程是并发执行的，而算法要求调用是按顺序依次执行，所以对这种序列图，算法失效。下面给出该算法 2 的形式化描述。

算法 2 交互层次关系的发现

```
输入：进程内消息交互序列 MessageQueue，
关注的消息序号 MesID
输出：MesID 对应的返回点 BackPoint
if 没有折返点信息 then
    MessageQueue. SignBackPionts()
end if
if MessageQueue. IsAStartPoint(MesID) then
    return MessageQueue. GetBackPiont(MesID)
else int ParentStartPoint =
    MessageQueue. GetParentStartPiont()
    return MessageQueue. GetBackPint(ParentStartPiont)
end if
```

函数名：SignBackPionts

```
输入：进程内消息调用序列 MessageQueue
输出：标注了折返点信息的序列 MessageQueue
MessageRecord Mess, MessNext, MessStart;
for (int i=1; i<MessageQueue. GetSize(); i++)
  Mess = MessageQueue. GetAt(i);
  MessNext = MessageQueue. GetAt(i+1);
  if(Mess. Receiver ! = MessNext. Sender) then
    MessageQueue. SetAsBackPiont(Mess)
    for(int j = Mess. GetID; j >0; j--)
      MessStart = MessageQueue. GetAt(j)
      if MessStart. Sender == MessNext. Sender then
        MessageQueue. SetAsStartPoint (MessStart, Mess.
        GetID())
      end if
    end for
  else if Mess. Receiver == Mess. Sender then
    MessageQueue. SetAsSelfCaller()
  end if
end for
MessageQueue. DealWtihSelfCaller()
```

1.3 设计模式的发现

设计模式是对重复出现问题的可重用的解决方案[8]。对于设计模式的结构描述，一般是由类和类间的关系组合而成。利用设计模式对消息序列图中的交互对象进行划分和抽象，为使用者提供设计模式侧面上观察和分析系统行为的能力。为了能自动获得系统中所存在的设计模式，我们需要从目标系统源代码提取出类结构信息，以此发现设计模式实例。

首先对设计模式的结构特征进行归纳和分类，使其转化成关于类特征的规则。然后，利用开发编译工具 OpenC^{++} 从目标系统的源代码中提取识别模式所必需的类信息，其中起决定作用的是类间关系，主要是继承、聚合、组装以及依赖关系，提取出的各种关系构成事实库。模式识别的过程就是在这种事实库上进行模式规则匹配的过程。

算法 3 设计模式的自动识别

```
输入：设计模式的结构规则 PatternDXML
      目标系统的类关系事实库 SystemDXML
输出：识别出目标系统中的设计模式实例 ResultSet
load pattern description file PatternDXML// 设计模式的结构规则
load target system description file SystemDXML// 目标系统的类关
    系事实库
//第一阶段，匹配类内部特征
for each class Pclass∈PatternDXML
    if Pclass. BeTrated then conntine //跳过已被作为相似类处理
        过的类
    for each class TestClass∈SystemDXML
        if IsValid(TestClass, Pclass)//关于类内部特征的匹配
            操作
            add TestClass to Candidates//分别加入到 Pclass
                及其相似类的候选列表 置
    end for
end for
//第二阶段，匹配类间关系
for each Pattern∈PatternDXML
    for each Candidate∈ShortClass
    //ShortClass 是 pattern 中 candidate 最少的类
        if FindPatterns(Candidate, Pattern)
        //根据模式的类间关系，在 Pattern 的 candidate 集合
            中寻找 Pattern 的实例
            Add Pattern to ResultSet
        end if
    end for
end for
```

在识别出目标系统中的设计模式实例之后，利用它们对消息序列图中的交互对象进行划分，从而产生出反映设计模式内部交互行为的序列图，以及屏蔽了内部交互而只反映设计模式实例外部行为特征的序列图。

2 系统的设计实现

通过使用以上 3 种抽象策略，并调用 Rose 的可扩展接口，我们分别实现了基于自动发现交互模式的序列图抽象、基于消息层次的序列图抽象以及基于设计模式识别的序列图抽象。

在系统设计中，将 Rose 模型中的序列图看成是线性的消息序列。为了能提高算法的执行效率和可重用性，设计了更快捷的序列容器即消息序列容器来支持该算法，并在其中建立了编码机制。通过改变运算格式，较大地提高了识别速度。系统结构功能图如图 1 所示。

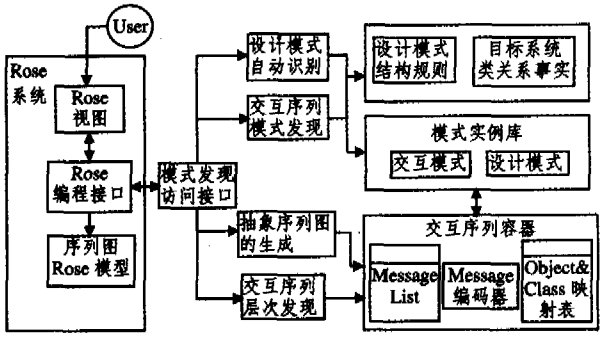

图 1 系统功能结构图

3 实验案例介绍

在 XDRE 系统的开发中，我们实现了文中所介绍的序列图抽象算法，同时利用 Rational Rose 的扩展性，将所实现的功能以 COM 插件的形式无缝集成到了 Rose 环境中。测试过程中，在使用了一些试验程序作为例子的同时，还选用了一些已经在生产中使用的软件系统进行试验。这一部分，主要介绍以“客户服务中心”系统中的对象交互协议[9]为目标系统进行的试验。该系统已在邮政 185 呼叫中心中得以使用。它基于 Unix 平台，实现主机内、主机间的进程通信和进程监控。由 90 个类组成，在此之上运行了 15 种进程组构件、25 种进程交互。

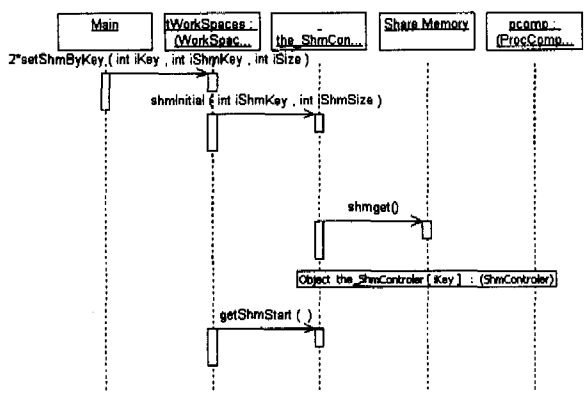

图 2 5889 序列图原图(局部)

在实验中，利用 XDRE 逆向工具对其源代码进行逆向分析，并生成目标系统运行时的各进程内和进程间的序列图。共自动逆向生成了 28 幅进程内交互序列图、1 幅进程间交互序列图。然后在此基础之上，执行我们所实现的各种序列图抽象功能。下面我们以 5889 进程内部交互为例，进行消息交互模式的自动发现与抽象。在实验中，设定的模式最短长度 MinPatternLength=3。当模式之间消息发生冲突时，以长度优先的原则进行取舍。结果如图 2～5 所示。

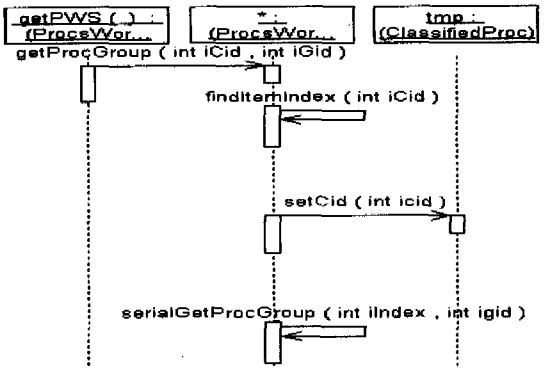

图 3 发现的模式 1

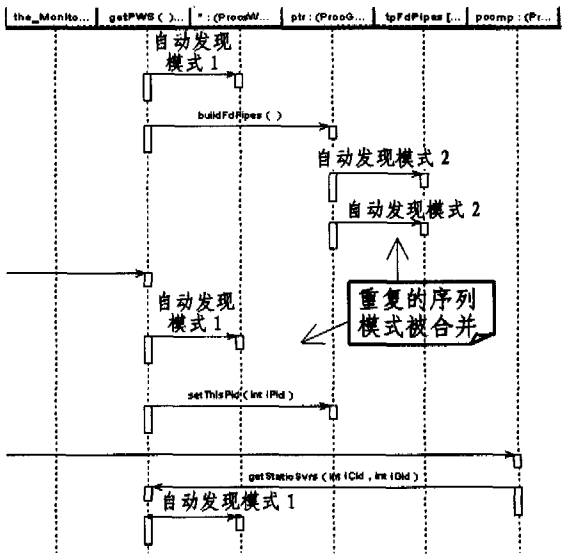

图 4 抽象后的 5889 序列图(局部)

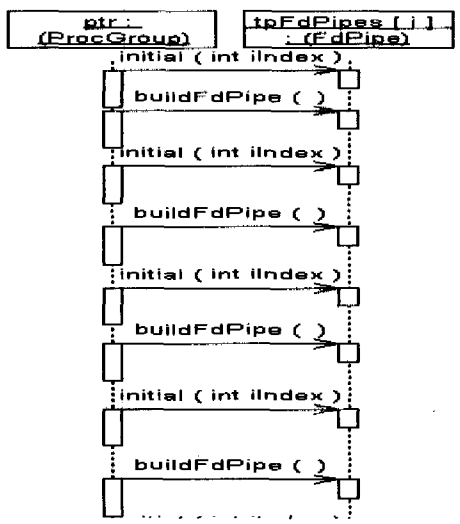

图 5 发现的模式 2

通过对实验结果的分析，发现该算法成功地找出了所有符合要求的交互模式，同时能够正确地进行相关的序列图抽象。在时间复杂度方面，因为需求中没有实时性要求，所以未进行量化测量，而主要是通过使用时的主观感受来判定。我们使用不同规模的序列图进行测试。以上面的 5889 号进程内交互序列图为例，它有 733 条消息调用，在所用实验的序列

(下转第 273 页)

是后续工作需要完善的。

参 考 文 献

1 Netmechanic. http://www.netmechanic.com
2 Web site garage. http://websitegarage.Netscape.com
3 Di Lucca G A, Fasolino A R, Faralli F, De Carlini U. Testing Web applications. In: Proceedings, International Conference on Software Maintenance, Oct. 2002. 310～319
4 Sun Microsystems. SunTest suite. http://www.sun.com/suntest
5 HtmlUnit: htmlunit.sourceforge.net
6 Rational robot. http://www.rational.com/products/robot/index.jsp
7 Sitetools. http://www.softlight.com/sitea/index.asp
8 Technovations load testing products. http://www.technovations.com/home.htm
9 Conallen J. Modeling Web Application Architectures with UML. Communications of the ACM, 1999,42(10)
10 Niese O, Margaria T, Steffen B. Automated Functional Testing of Web-Based Applications. QualityWeek Europe 2002. Brussels, Belgium. Applied computing. Nicosia, Cyprus. 2002. 1662～1669
11 Yang Ji-Tzay, Huang Jiun-Long, Wang Feng-Jian, et al. Constructing an Object-Oriented Architecture for Web Application Testing. Journal of Information Science and Engineering 18, 2002. 59～84
12 E-tester. http://www.rswsoftware.com/products/etester index.shtml
13 Silk test. http://www.segue.com/html/s solutions/silk/s family.htm
14 Watchfire enterprise solution. http://www.watchfire.com/solutions/wes.asp
15 VeriWeb: Automatically Testing. Dynamic Web Sites. Juliana Freire. http://www-db.bell-labs.com/juliana. Bell Labs
16 刘昶. 面向交互式软件的测试模型设计和测试用例生成:[北京航空航天大学硕士论文]. 2005,3
17 蒋宗礼, 姜守旭. 形式语言与自动机理论. 清华大学出版社, 2003. 235～257
18 van Rossum G. Python Programming Language. Python. Orgnization, 1990～2005. http://www.python.org
19 Lee J J. Python Bits. SourceForge. 2004.5. http://wwwsearch.sourceforge.net
20 Richardson L. We called him Tortoise because he taughtus. crummy dot com. 2004.10. leonardr@segfault.org

(上接第 268 页)

图中,属于较大的。从开始发现算法到输出抽象后的序列图,用时在 1～2s 之间,时间等待完全在可接受的范围内。

对于消息层次关系的发现和抽象,也进行了相关实验。以 2523 号进程内交互序列图为示例,由于篇幅所限,我们只给出了抽象后的最终结果,如图 6 所示。

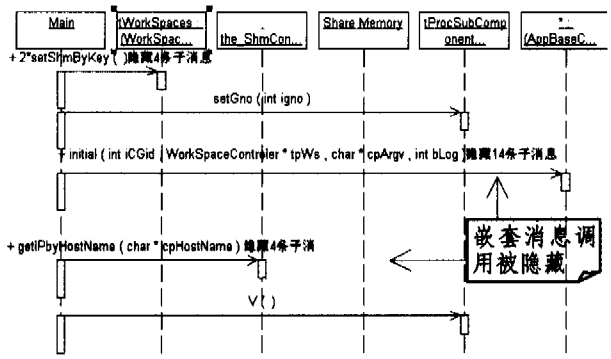

图 6 层次抽象后的 2523 序列图(局部)

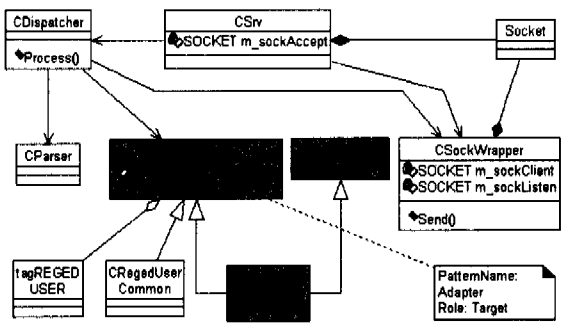

图 7 注册服务系统的服务器端发现 Adapter 模式

图 7 是一个设计模式自动发现的例子,目标系统是一个以客户端/服务器模式进行注册服务的系统。我们找到了 Adapter 模式的一个实例,为了方便用户观察,系统重新绘制类图,通过着色处理把模式突出显示出来,并对模式中的核心类加上 note 标记说明。

结论 作为逆向工程工具 XDRE 系统的一个功能模块,本文提出并实现了交互重复模式的自动发现和交互层次的自动恢复方法,以及基于类图的设计模式发现,并实现了以此为依据对动态剧情的抽象,为用户提供了在序列图上交互地进行剧情抽象的支持。用户可以从系统较低层的行为构造更高设计层的行为模型,在不同的抽象层次上观察系统的动态行为,验证、更新最初的设计模型,认定需要再工程的部分。从使用者的角度来看,该功能模块的时间迟延性、正确性和可用性等都达到了要求。

在已完成的工作基础上,今后的工作将集中在通过对系统的行为特征分析,识别设计模式,使得系统具有自动发现动态剧情中潜在设计模式的能力。

参 考 文 献

1 Ernst M D. Static and Dynamic Analysis: Synery and duality. In: ICSE Workshop on Dynamic Analysis, (Portland, OR), May 9, 2003. 24～27
2 Richner T, Ducasse S. Recovering high level views of object-oriented applications from static and dynamic information. In: Proceedings of International Conference on Software Maintenance, Oxford, UK, IEEE CS Press, 1999. 13～22
3 Systä T. Static and Dynamic Reverse Engineering Techniques for Java Software Systems: [Ph D Dissertation]. Dept of Computer and Information Sciences, Unversity of Tampere, May 2000
4 李青山, 陈平. 利用改进遗传算法恢复高层架构. 软件学报, 2003,14(7):1221～1228
5 李凡, 陈平. 逆向工程中 UML 序列图的自动生成与抽象技术. 计算机科学, 2004(12)
6 陈平. C3I 系统应用软件逆向工程开发工具研究任务申请书. 本项目课题组, 2001
7 张广红. UML 序列图的逆向自动生成与剧情抽象:[硕士论文]. 西安:西安电子科技大学, 2003
8 Kramer C. Design recovery by automated search for structural design patterns in object-oriented software. In: Proceedings of the Third Working Conference on Reverse Engineering, 1996. 208～221
9 陈平. iCALL 呼叫中心应用支撑平台低层通信模型设计文档. 西安电子科技大学软件工程研究所, 2000