逆向工程中的UML序列图抽象技术*)

李 凡 陈 平
(西安电子科技大学软件工程研究所 西安710071)

摘 要 研究了逆向工程中序列图的抽象问题。以逆向工程分析工具RER的开发为背景,针对其逆向生成的进程间交互序列图和进程内部交互序列图,引入并实现了面向交互的抽象、面向类的抽象、面向进程模块的抽象和面向模式的抽象四种序列图抽象方法。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RER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序列图的功能。

关键词 逆向工程,UML,序列图,抽象

Abstract Technology of UML Sequence Diagram in Reverse Engineering

LI Fan CHEN Ping
(Software Engineering Institute, Xidian University, Xi'an 710071)

Abstract This paper emphasizes on the research of the technology for scenario abstraction of UML sequence diagram. Four kinds of abstraction Methods are presented. These abstraction Methods are interaction oriented, class oriented, process module oriented and pattern oriented abstract method. At the same time, based on the extendibility of Rational Rose, these abstraction Methods are implemented and seamless integrated into Rose development environment.

Keywords Reverse engineering, UML, Sequence diagram, Abstract

1 引言

软件逆向工程是分析软件构件和它们的关系,以另一种形式或在更高抽象层次上描述该软件的过程[1]。目前出现了很多工具支持逆向工程,总的来说工具的能力包括静态解析和动态分析两方面,分别获取软件的静态信息和动态信息。

本论文的研究工作是逆向工程工具RER系统的一部分。RER系统是一套逆向工程工具,提供符合UML标准的动态模型的逆向生成、符合UML标准的静态模型的逆向生成与分层抽象等方面的能力;同时,这一套工具被无缝集成到了Rational开发环境中,并与该环境中的其他工具协同工作,以扩充Rational开发环境在与源代码结构和语义相关的工具方面的支持[2]。论文的主要研究对象是UML序列图。序列图表示一个交互过程,是在一个要达到预期的操作或结果的协作中,一组类元角色间的消息[3]。RER系统通过植入技术和动态分析自动逆向生成的UML序列图,这些序列图反映了目标系统在实际运行过程中各进程内部对象之间和整个系统内进程之间的动态消息交互情况。

动态交互信息是运行时最真实的信息,但当被分析的目标系统变得稍微复杂一点的时候,由动态信息生成的序列图往往会在水平和垂直方向快速地增长,增大了用户理解系统的难度,因此需要提高序列图的抽象层次。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,这些交互模式的重复出现,大大增加了序列图的规模,影响系统执行效率;同时,也不利于使用者对序列图的理解。所以有必要进行面向各种侧面的抽象和合并,从而使得使用者可以从不同的层次和不同关注点来分析动态信息。

本论文针对逆向自动生成的序列图,引入了多种序列图的分割和抽象方法,并在可视化的交互环境下将其实现。为软件系统的逆向分析和高层抽象提供了有利的支持。为了不失完整性并明确作为论文研究对象的各种动态交互序列图的内含,下面先简要介绍一下这些序列图的生成过程和特点。

2 序列图的自动生成

逆向工程工具RER系统利用程序植入技术对目标系统的源代码进行植入和动态分析,经过对动态信息进行收集和过滤,生成反映目标系统动态消息交互情况的序列图。总体设计如图1所示。

生成序列图的总体工作流程是这样的:首先对目标系统的源代码进行全部植入。根据全部植入的源代码充分执行的结果构造程序依赖图,在用户根据程序依赖图确定关注的植入范围后,把植入范围信息交给植入子系统,由后者对目标系统的源码进行部分植入。

在此之后,对经过部分植入的源代码系统进行编译执行,这样得到的系统包含着植入代码,在使用过程中将产生系统的动态行为信息;此后,对收集到的这些动态信息进行过滤,并根据目标系统的层次结构划分出不同层次的动态信息,转换成XML文件。至此便形成了产生序列图所需的动态信息文件。

序列图产生模块根据这些动态信息XML文件,自动生成符合UML标准的序列图,添加到目标系统的初始模型中。由于动态分析的特点所致,所生成的序列图分为两种,一种反映了目标系统在实际运行过程中各进程内部对象之间动态交互情况,称为进程内交互序列图,有几个进程就有几个这样的序列图;另一种反映整个系统运行时所有进程之间的交互情况,称为进程间交互序列图,该图在每次运行时只生成一个。

*)本文工作来源于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 博士生,研究兴趣为软件逆向工程、人工智能应用;陈 平 教授,博士生导师,主要研究兴趣为面相对象技术、软件(逆向)工程。

以上功能模块实现了一种对软件系统的源码进行动态分析的方法，该模块所产生的序列图是对目标系统实际运行情况的真实反映，并为序列图向状态图的转换等进一步逆向建模提供了基础[4]。图2显示了一个进程的内部对象交互序列图的局部。

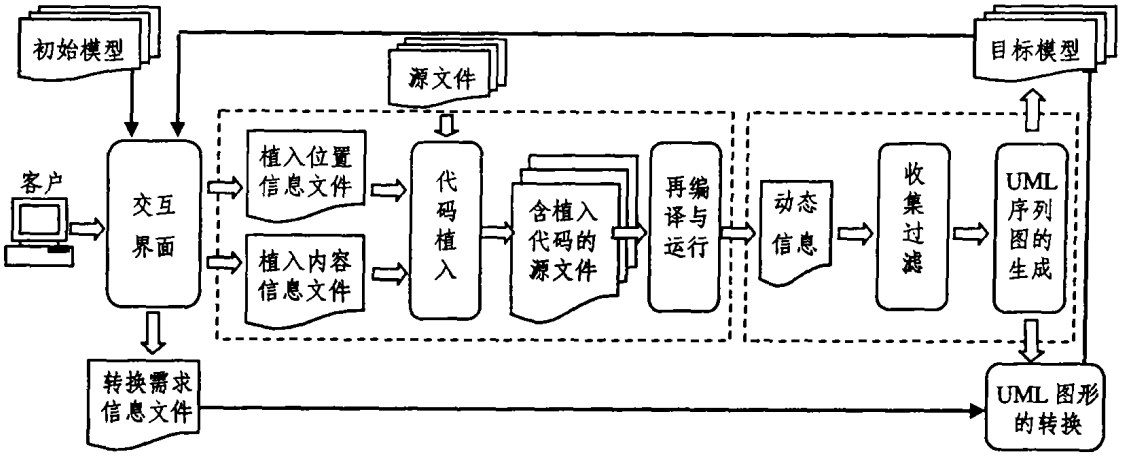

图1 RER 系统的总体设计

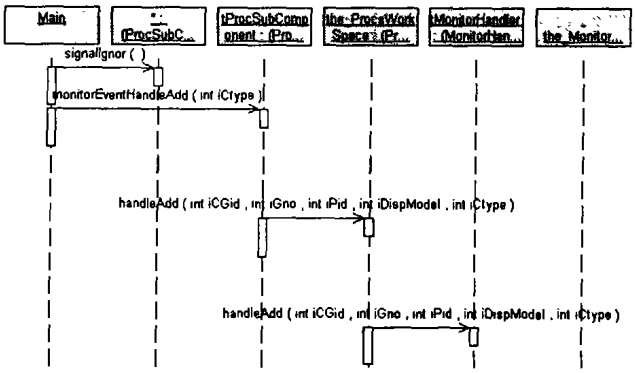

图2 进程内部对象交互序列图示例(局部)

前面已经指出，根据动态生成的序列图的特点，有必要对其进行各种侧面的抽象和合并，从而使得使用者可以从不同的层次和不同关侧面来分析动态信息。下面将依次介绍为实现这一目标所使用的四种抽象方法。

3 序列图的抽象方法

3.1 序列图的用户交互式抽象

所谓用户交互式抽象是指由使用者在可视环境下，从序列图中选择其认为需要合并的列(即交互对象)，把要合并的列包起来，把这几列的内部交互作为一个局部序列图进行记录，并且在原序列图的基础上再产生一个新的序列图，这个序列图在保证不改变原序列图的语义的前提下，忽略这几列的内部交互[5]。

对序列图进行这种处理有两个作用，一是方便用户查阅合并列的内部交互，另一个是作为一个基础工作，为后面提出的面向类和面向进程的自动抽象提供支持；同时，还可以在工具由动态信息文件生成序列图时，自动查询文件中是否存在合并的这几列，如果存在通知用户并询问是否需要合并这几列。

在实际使用时，由用户在序列图中选择要合并的列集合，并为其命名，而工具要完成的工作是这样的：

算法1

step1：令用户关注的序列图为 sourceDiagram，由使用者选择需要合并的列，形成列的集合 objsCombined；

step2：创建新的序列图 combinedDiagram，将 sourceDiagram 中不属于 objsCombined 以外的所有对象加入到 combinedDiagram 中，同时将 objsCombined 作为一个新对象加入，命名为 Combined；

Step3：创建新的序列图 innerDiagram，将 sourceDiagram 中属于 objsCombined 的所有对象加入到 innerDiagram 中；

Step4：

```
int i=0;
int j=0;
for each message in sourceDiagram
{
  if message.Sender ∈ ObjsCombined And message.Sender ∈ ObjsCombined Then
       innerDiagram.CreateMessage ( message.Sender, message.Receiver,++i)
    Else
      if message.Sender ∈ ObjsCombined Then
        message.Sender = Combined
      Else if message.Reciever ∈ ObjsCombined
        message.Receiver = Combined
      End if
       combinedDiagram.CreateMessage ( message.Sender, message.Receiver,++j)
  End if
}
```

Step5：将修改保存到 Rose 模型文件中

通过算法1实现了在 Rose 环境下序列图的交互式抽象。

3.2 面向类的序列图抽象

上面给出的用户交互式抽象提供了一种方法，使用户可以把其感兴趣的列集分离出来单独观察。同时，面向对象的软件系统通常存在着以类为核心的层次结构，而在一个进程内，通过动态分析逆向生成的序列图，其关注的层次定位在对象

层。所以，当使用者希望对类之间的动态交互关系进行分析时，就需要实现对序列图面向类的抽象，为使用者生成类之间的消息交互序列图。

本系统中目前所实现的面向类的抽象针对的是进程内部对象间的交互序列图。所采用的方法是通过静态和动态分析得到类与对象的对应关系；利用这些对应关系作为合并依据，逐次使用算法1中的合并方法(step2-step5)，生成所需要的序列图。

实现过程如算法2所示：

算法2

令该序列图中所出现的类的集合为 ClassList，且用户关注的序列图为 sourceDiagram；

```
SourceDiagram;
Int LoopFlag = 0;
For each Class in ClassList
{
  令集合 objsCombined 为当前 Class 所有对象的集合
  执行算法1的 Step2-Step5;
  If LoopFlag!= 0 Then
    删除 SouceDiagram;
  End if
  SouceDiagram = CombinedDiagram;
  LoopFlag = 1;
}
```

通过以上步骤，可以产生类这一层面上的交互序列图和各个类各自所生成的对象内部之间的交互序列图。

3.3 面向进程模块和子系统的抽象

如果站在程序执行的角度，可以存在：进程→进程模块→子系统这样的系统体系结构，其中的进程模块指的是由相同程序代码所产生的进程的集合，而子系统是指按照设计需要，对进程进程模块集合进行的划分。

前面曾经提到，通过序列图自动生成模块，可以产生目标系统在运行时各进程之间的交互序列图。对于这种序列图，可以把属于同一进程模块的进程合并起来，从而得到反映进程模块间交互的序列图。为了实现这一目的，需要收集目标系统运行时进程与进程模块的对应关系，这一信息是在动态分析时获得的。根据这些对应关系就可以对进程间交互序列图进行抽象，这一过程与前面所提到的关于类的抽象十分的相似。只是将对象替换为进程，将类替换为进程模块。

算法3：

令该序列图中所出现的进程模块的集合为 ProcessModList，且用户关注的进程间交互序列图为 sourceDiagram；

```
Int LoopFlag = 0;
For each ProcessMod in ProcessList
{
  令集合 ProcessMod 为当前 objsCombined 所有进程的集合
  执行算法1的 Step2-Step5;
  If LoopFlag!= 0 then
    删除 SouceDiagram;
  End if
  SouceDiagram = CombinedDiagram;
  LoopFlag = 1;
}
```

生成了进程模块间的交互之后，结合子系统的划分情况，可以进一步对进程模块进行抽象，获得子系统级的交互序列图。这一功能的前提是目标系统在设计时存在子系统的划分，同时由于现有的动态分析还不能得到子系统的划分信息，所以在本系统中，子系统的划分情况是由使用者通过 XML 文件提供给抽象模块的。而具体的实现方法与面向类和面向进程模块是类似的，这里不再赘述。

3.4 面向交互模式的序列图抽象

面向交互模式的序列图抽象是指由用户在序列图上可视化地定义自己关注的交互模式，即选择若干个相邻消息作为一个交互模式，然后选择匹配方式，在不同的层次匹配这个模式，用一个代表这个模式的符号替换序列图中重复出现的该模式，从而在垂直方向减少序列图中显示的信息[6,7]。这部分工作使用户能够控制模式的抽取，更容易从较高层次理解系统。

这里所说的交互模式跟设计模式不一样。设计模式就是对一个特定环境中经常出现问题的解决方案的描述，通过设计模式，软件的设计经验可以被文档化并被其他人重用[8]。这里定义交互模式是因为它也是可重复的实体，并且在可视化面向对象消息跟踪时，他们在视图中创建了可视化的模式。两种类型的模式的关系是交互模式来源于各种设计模式，并且可以作为设计模式存在的底层证据。

用户在序列图中选择几个消息，作为一个交互模式并为它命名之后，选择工具所提供的匹配方式，而工具要完成的工作是：

算法4：

Step1：由用户指定消息序列作为模式，定义模式名为：Pattern；当前关注的序列图为：SourceDiagram；合并后的序列图为 MatchedDiagram；

Step2：令 MessID 为当前消息在消息序列中的序号，初值为1；

```
For each Message in SourceDiagram
{
  以 MessID 为起点，对 SourceDiagram 中的消息序列
  MessList 关于 Pattern 进行匹配操作(所采用的匹配
  算法为：MatchMethod，返回匹配结果和匹配后 Mes-
  sID 的增量 MessIDInc);
  If 匹配成功 Then
    在 MatchedDiagram 中追加名为 Pattern 的消息;
    MessID += MessIDInc;
  Else
    在 MatchedDiagram 消息序列中追加 SourceDia-
    gram 的当前消息 Message;
    MessID++;
  End if
}
```

Step3：将 MatchedDiagram 保存到当前的 Rose 模型中；创建新的序列图 MatchPattern，将 Pattern 中的交互信息加入到 MatchPattern 中，并保存。

在算法4中，MatchMethod 是匹配策略，目前共确定了四种匹配方式：

1)完全匹配(exact)，匹配的条件是，被匹配的模式必须跟选择的交互模式完全一样，两个模式中每一条消息的发送者、接收者、消息名要完全相同，并保证消息的连续性；

2)可间隔 (interleaved)，匹配的条件比完全匹配条件宽松一点，即不必保证消息的连续性，中间可间隔一些其他消息；

3)包含 (contained)，匹配时保证交互的连续性，但是消息的发送者、接收者不必完全一样，只要存在包含(contain)关系；包含关系是指同一个类的对象、同属一个子系统或一个是对象，另一个是子系统，且这个对象属于这个子系统；

4)可间隔包含(contained interleaved),匹配的条件比包含层条件宽松,不必保证消息的连续性;

MatchMethod 选择哪一种匹配模式是在算法执行之前由使用者决定,MatchMethod 返回匹配结果和消息的序号增量 MessIDInc,其值为此次匹配中 MessList 参与匹配的消息总数。

4 案例介绍

在实际中,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,将所实现的功能以插件的形式无逢集成到了 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。

在实验中,利用 RER 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,以进程间交互序列图为例。该序列图包含的交互对象为进程和 SharaMemory、Synchronization 和 MsgQuene 这三个系统对象,如图3所示。

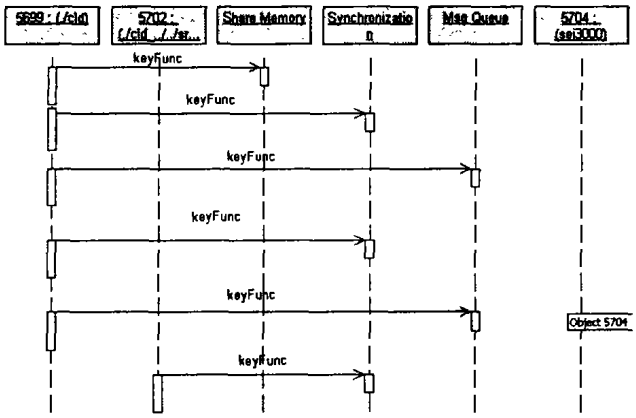

图3 进程间交互序列图(局部)

首先对其进行面向进程模块的抽象,由此所生成的序列图称为进程模块间交互序列图,它将原来的序列图中的交互对象即各个进程按照相同的进程模块(即生成进程的代码)进行合并,如图4所示。

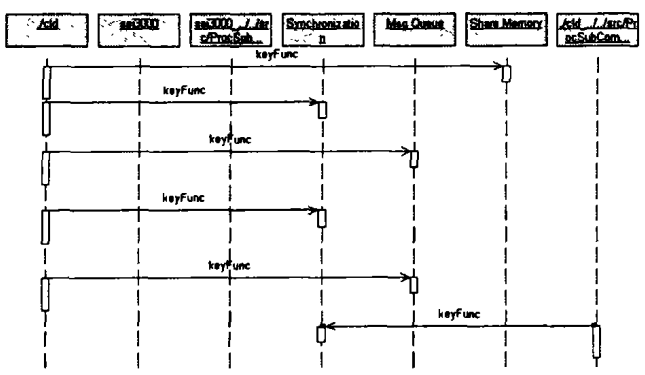

图4 抽象后得到的进程模块级交互序列图(局部)

然后,通过手工选择,将这些进程模块划分为两个子系统 Sub1和 Sub2,以此为依据进行面向子系统的抽象,得到了子系统间交互序列图,它反映了子系统 Sub1、Sub2以及 SharaMemory、Synchronization 和 MsgQuene 这三个系统对象间的交互情况,通过该图可以从更高层次上来观察系统的交互情况,如图5所示。

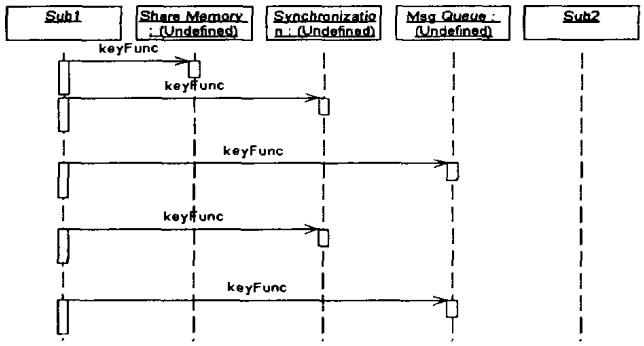

图5 抽象后得到的子系统级交互序列图(局部)

同时,对动态分析得到的序列图还可以进行面向交互的抽象、面向类的抽象和面向模式的抽象,基本实现了系统设计时的要求。

结论 作为逆向工程工具 RER 系统的一个功能模块,本论文中的工作实现了多种对序列图的分割和抽象方法,并在可视化的交互环境下将其实现。为用户提供了反映系统动态行为特征的符合 UML 标准的多层次序列图,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;同时,所产生的符合 UML 标准的序列图可用于转换出状态图和类图。

为了进一步完善序列图抽象的功能,今后的工作将集中在以下几个方面:首先,进一步完善面向类的抽象,使其可以不受进程的约束,实现跨进程的类的抽象和依照类继承关系的多层次抽象;其次,在面向模式的序列图抽象中引入模式挖掘的技术,使得系统具有自动发现序列图中潜在交互模式的能力;最后,在交互模式的包含匹配方式上进行改进,使得判断对象是否与子系统有包含关系时,该子系统可以扩展为静态模型中的子系统,这就需要通过查找静态模型来判断包含关系。

参考文献

1 Chikofsky E J,Cross Ⅱ J H. Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software,1990,7(1):13~17

2 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

3 OMG Unified Modeling Language Specification Version 1.3, June 1999

4 Selonen P,Systä T. Scenario-based Synthesis of Annotated Class Diagrams in UML. 2000

5 Systä T,Koskimies K. Extracting state diagrams from legacy systems. 1996

6 Systä T. Building high-level scenarios using static abstractions. 1996

7 张广红. UML 序列图的逆向自动生成与剧情抽象:[硕士论文]. 西安电子科技大学,2003

8 Kramer C. Design recovery by automated search for structural design patterns in object-oriented software. In: Proc. of the Third Working Conf. on Reverse Engineering, Nov. 1996. 208 ~ 215