王昌晶,陈茜,丁希龙,等. 基于模型驱动的 Web 服务符号执行与验证 [J]. 江西师范大学学报(自然科学版),2022,46(1):37-48.
WANG Changjing,CHEN Xi,DING Xilong,et al. The web service symbolic execution and verification based on model-driven [J]. Journal of Jiangxi Normal University(Natural Science),2022,46(1):37-48.

文章编号:1000-5862(2021)06-0037-12

基于模型驱动的 Web 服务符号执行与验证

王昌晶[1,2],陈　茜[1],丁希龙[1],罗海梅[3],左正康[1*]
(1. 江西师范大学计算机信息工程学院,江西 南昌　330022;2. 江西师范大学管理科学与工程研究中心,江西 南昌　330022;
3. 江西师范大学物理与通信电子学院,江西 南昌　330022)

摘要:Web 服务测试与验证是保证 Web 服务功能正确的关键,目前大多数 Web 服务的研究无法对程序路径穷举遍历,不能保证分析的完备性. 针对该不足,在基于模型驱动的 3 阶段 Web 服务模型转换生成方法的基础上,该文对转换生成的 Java 代码进行符号执行与形式化验证. 符号执行方法可对程序运行的所有路径进行分析,为程序测试提供高覆盖率的测试用例,可以触发深层的程序错误,进而在 Java 代码中加入 JML 方法契约,可对 Web 服务进行形式化验证. 通过 PayPal Web 服务案例,采用模型驱动的方法将 Web 服务模型转换生成方法生成 Java 代码,使用自动化工具对 Java 代码进行符号执行;将 Radl-WS 服务建模语言转换为 JML 方法契约,并对 Java 代码进行形式化验证. 符号执行与形式化验证方法确保了生成的 Java 代码可靠性与正确性,提高了自动化程度.

关键词:Web 服务;Radl-WS;Java 代码;符号执行;验证
中图分类号:TP 311　　**文献标志码**:A　　**DOI**:10.16357/j.cnki.issn1000-5862.2022.01.06

0　引言

随着云计算技术的高速发展,Web 服务作为一种新兴的网络应用模式得到了迅速的发展. 基于面向服务架构的(Service Oriented Architecture,SOA)Web 服务是一个独立、自包含、低耦合、可编程的应用程序,在电子商务、资料查询、办公自动化等若干核心领域中有着广泛应用.

符号执行技术是目前研究领域广泛认可的形式化、自动化测试技术. 近年来,符号执行技术不断地发展与优化以及与一些新技术(如 SMT、Fuzzing 等)结合,这极大地提高了符号执行的可用性与实用性. 符号执行工具(如 LLVM、KLEE、S2E、SED 等)的出现与发展提高了自动化程度. 因此,符号执行技术作为一种重要的形式化方法[1]和软件分析技术,引起了国际学术界的广泛研究.

如何保证 Web 服务可靠性与正确性并满足用户需求是目前 Web 服务研究领域的热点. Web 服务的测试[2-3]与验证能提前发现 Web 服务存在的问题,这是发布高可靠 Web 服务的有效途径. 但在实际测试中,由于资源有限所以使得测试不充分,而且 Web 服务测试复杂、费时费力. 国内外对于 Web 服务验证的研究一般是基于某种形式化方法,大致可以分为 3 类:Petri 网、自动机理论和进程代数. 其中 Petri 网与自动机的方法对服务组合的验证[4]计算量大,其复杂度随着服务规模的增大而急剧增大;进程代数比较抽象,难以被广泛使用.

模型驱动架构的方法采用主流的商业过程建模[5-6],支持从系统高层模型逐步转换,但难以生成 Web 服务可执行代码. 面向服务的建模分析方法(SOMA)缺乏形式化语义. 针对该不足,本文前期工

收稿日期:2021-09-12
基金项目:国家自然科学基金(61762049,61862033)和江西省教育厅科技重点课题(GJJ210307)资助项目.
作者简介:王昌晶(1977—),男,江西南昌人,教授,博士,博士生导师,主要从事软件形式化方法、形式规格说明方法和 Web 服务方面的研究. E-mail:wcj771006@126.com
通信作者:左正康(1980—),男,江西抚州人,教授,博士,主要从事泛型程序设计和可信软件的研究. E-mail:kerrykaren@126.com

作使用自定义的建模语言 Radl-WS 对 Web 服务需求建模,使用模型驱动的方式对 Web 服务建模与转换,并对转换生成的 Java 代码进行符号执行与验证.

本文采用模型驱动的方法对 Web 服务进行符号执行与验证. 在前期研究的基础上,对转换生成的 Java 代码符号执行,并将 Radl-WS 需求规约对应转换成 JML(Java Modeling Language)方法契约加入 Java 代码进行验证. 本文使用基于 Eclipse 平台扩展的符号执行工具 SED(Symbolic Execution Debugger),对基于模型驱动的 Web 3 阶段生成的 Java 代码进行符号执行与验证.

1 3 阶段 Web 服务模型转换生成方法

模型驱动架构 MDA(Model Driven Architecture)是由 OMG 定义的一个软件开发框架[7],以模型为核心,将实际问题抽象化并建立相关模型,再对模型进行转换与精化. MDA 开发过程是从 3 个不同的层次建立系统模型:第 1 层为计算无关模型(Computational Independent Model,CIM),第 2 层为平台无关模型(Platform Independent Model,PIM),第 3 层为平台相关模型(Platform Specific Model,PSM).

基于模型驱动的 3 阶段 Web 服务模型转换生成方法的过程如下:首先使用代数规范的 Radl-WS 对 Web 服务需求建模;然后将 Radl-WS 服务需求模型转换成 Apla 服务设计模型;再将 Apla 服务设计模型通过 PAR 方法及其支持平台程序转换器生成可执行代码[8-12];最后将可执行代码封装成服务. 基于模型驱动的 3 阶段转换生成方法如图 1 所示.

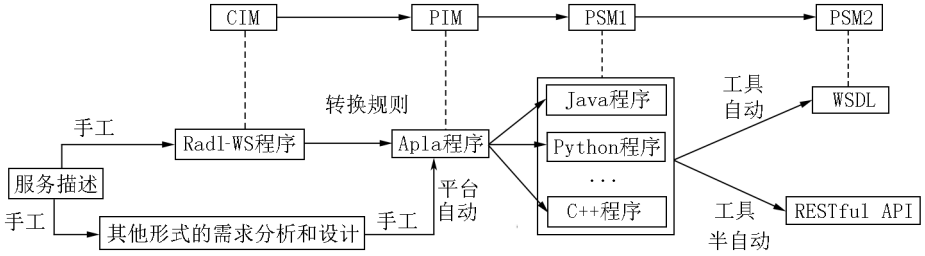

图 1 基于模型驱动的 3 阶段转换生成方法

其中 Radl-WS 服务需求模型对应 CIM,Apla 服务设计模型对应 PIM,经过 PAR 方法及其支撑平台系列转换器(Apla→Java、Apla→C++等)生成的可执行代码对应 PSM1,最后封装成的服务 WSDL/RESTful API 对应 PSM2.

为了提高模型的可靠性与正确性,在前期模型驱动的 3 阶段转换生成方法的基础上,本文对 Web 服务进行符号执行与验证.

2 Java 代码符号执行

2.1 符号执行

符号执行是一种程序分析技术,通过分析程序来得到让特定代码区域执行的输入[13-14]. 在使用符号执行分析一个程序时,用符号值代替具体值作为输入. 通过追踪在程序中各个可能执行状态处的符号值及所需要满足的条件,计算可满足的具体值,将它作为输入代入程序中执行,从而验证在不同状态输入下程序的正确性.

符号执行路径是一个 true 和 false 序列 seq = $(P_0, P_1, \cdots, P_n)$,若一个条件语句为 P_i = true 则这表示条件语句为 true,否则为 false. 对如图 2 所示的代码进行符号执行,分析程序 testme( ),符号执行有 3 条路径,形成如图 3 所示的符号执行树.

```
1 int twice (int v){
2       return 2 * v;
3 }
4 int testme(int x,int y){
5       int z = twice(y);
6          if(z > x){
7             if(x > y + 10){
8                error!
9                   }
10             else{
11                return y;
12                   }
13          }
14          else
15          {
16                return x;
17                }
18       }
```

图 2 符号执行示例代码

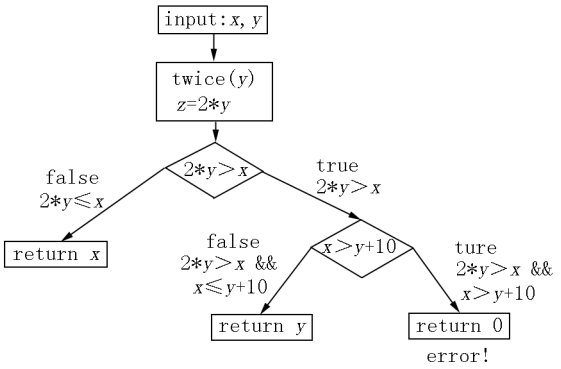

图 3　示例程序代码执行树

以图 2 的代码为例,符号执行会在全局中维护 2 个变量(σ,P_C). 符号状态 σ 记录在程序中每个变量到符号表达式的映射. 符号化路径约束 P_C 表示路径条件,初始值为 true. σ 和 P_C 在符号执行过程中不断更新,当符号执行结束时,求解 P_C 就可得覆盖所有路径的输入. 将输入 x、y 定义为符号变量 a、b 的过程如下:

(i) σ:$x\to a$,$y\to b$,P_{C_1}:true;

(ii) σ:$x\to a$,$y\to b$;$z\to 2b$,P_{C_2}:true;

(iii) σ:$x\to a$,$y\to b$;$z\to 2b$,P_{C_3}:true$\wedge 2*b\leqslant a$;

(iv) σ:$x\to a$,$y\to b$;$z\to 2b$,P_{C_4}:true$\wedge 2*b>a$;

(v) σ:$x\to a$,$y\to b$;$z\to 2b$,P_{C_5}:true$\wedge(2*b>a)\wedge(a\leqslant b+10)$;

(vi) σ:$x\to a$,$y\to b$;$z\to 2b$,P_{C_6}:true$\wedge(2*b>a)\wedge(a>b+10)$.

在符号代码执行过程中,若遇到赋值操作,则也将新产生的变量当成符号变量进行处理. 如代码第 5 行变量赋值,符号变量在处理后为 $z\to 2b$. 在分析处理完毕后,可以使用 SMT 约束求解器求解满足路径约束 P_C 的解,即为沿着该路径执行的某一具体测试案例. 如在该示例中的路径约束 P_{C_6}:true $\wedge$ $(2*b>a)\wedge(a>b+10)$,当 $a=30$、$b=16$ 时满足此路径约束(即当以 $x=30$、$y=16$ 作为其中一个测试案例输入)时,可以触发程序错误.

2.2　对 Java 代码符号执行

SED(Symbolic Execution Debugger)是 Eclipse 平台扩展的交互式符号执行调试器[15],可以安装到 Java 编译环境中,方便使用. 交互式符号执行调试器 SED 与传统调试器一样能定位在源码中的缺陷,SED 还提供了一个 Java 程序符号执行引擎 KeY[16],支持用 JML 规范注释的 Java 代码. SED 不仅可以提高对 Java 代码的审查与错误定位,还可验证任务的有效性与正确性. 本文使用交互式调试器 SED 作为自动化验证工具来验证实验的有效性与正确性.

将在图 2 中的伪代码转换成 Java 可以执行代码,使用 SED 交互式符号执行调试器执行. 如图 4 所示,可以看出自动化工具生成的符号执行树与图 3对示例程序分析生成的执行树分支路径一致,达到了预期效果,这说明符号执行的正确性. SED 视图结构形象直观、操作界面友好、交互方便且易使用易上手. SED 不仅支持严格的验证,还提高了代码审查与证明的效率.

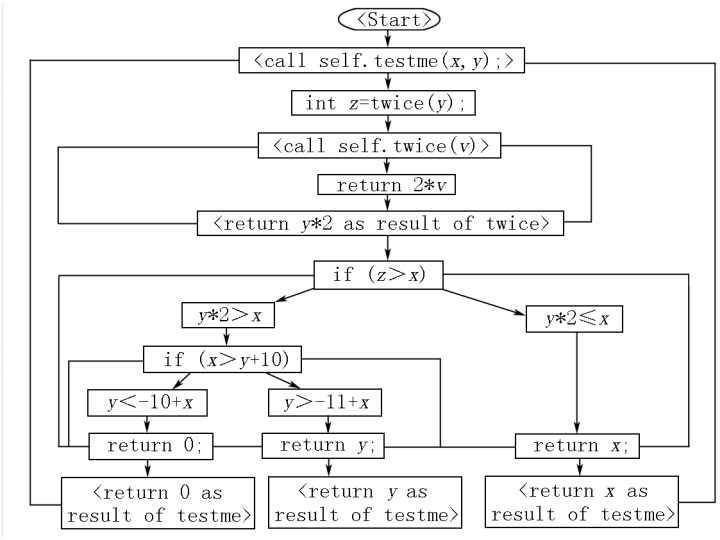

图 4　Java 代码符号执行

3　Java 代码形式化验证

3.1　Radl-WS 建模语言转换为 JML 方法契约

Radl-WS 是在 Radl 语言的基础上扩展而来的,是一种新型的基于代数规范的 Web 服务建模语言. Radl-WS 代数规范由若干规格单元组成,代表在软件系统中的一个实体类型[17-20]. 每个规格说明单元包括一个〈sort name〉,其中又包括 2 个更小的单元:(i)签名单元(signature unit),其语法由签名来定义;(ii)公理单元(axiom unit),其语法由若干必须满足的公理来定义. 使用 BNF 描述 Radl-WS 代码规

范如下:

〈spec unit〉::=spec〈sort name〉
[extends〈extend sort list〉]
[imports〈import sort list〉]
〈signature unit〉;
[〈axioms unit〉]
end-spec.

其中签名单元(〈signature unit〉)BNF 描述如下:

〈signature unit〉::[sorts〈sort list〉];
ops 〈operation list〉.

在 Radl-WS 的签名单元(〈signature unit〉)中,前置断言用 Q 开头的谓词表达式来表示,后置断言用 R 开头的谓词表达式来表示,规格说明如下:

|[标志符声明]|
Q:谓词表达式;
R:谓词表达式.

JML 是一种形式化的行为接口规范语言,是为 Java 量身定制的. JML 引入模型域、量词、前后置断言、预处理、条件继承及异常行为处理规范等描述行为到结构中. 通过使用标识符来说明一个方法的预期功能,而不关心具体的实现.

如求一个整型数组的平均值,该方法命名为 sum_average,该方法的 Java 代码很简单,具体代码如图 5 所示.

```
public int sum_average(int[ ] a){
    int sum=0;
    for (int i=0;i<a.length;i++){
        sum+=a[i];
        }
    return sum/a.length;
}
```

图 5 求数组平均值的 Java 代码

本文采用 Radl-WS 对该方法进行建模,由对该方法需求分析可知,该方法成立的前提是数组不为空,即作为在 Radl-WS 规范中前置断言 Q:a! = null. 该方法在执行结束后,需满足 i 大于等于 0 且小于数组长度,用 Radl-WS 后置断言 R 描述为 R:$i \geqslant 0$ && i<#(a);$a[i]$. 根据 Radl-WS 语法规则,关键词 spec 规格说明是一个方法,规格说明单元 sort 为数组类型,op 表示具体操作为检查观察子(observer),in 表示输入,out 表示输出. 该方法的 Radl-WS 规约如图 6 所示.

```
spec Method
imports:integer
sort:array
op:
  observer:
      sum_average:array→int
      |[in a:array;out sum:int]|
      {Q:a! = null}
      {R:i≥0 && i<#(a);a[i]}
axioms:
end-spec
```

图 6 sum_average Radl-WS 需求规约

以图 6 为例,将 Radl-WS 规约转换成 JML 方法契约,具体转换步骤如下:

(i)名称和标识符声明转换. 将观察子中对应的操作的名称 sum_average 转换成 Java 代码的方法名 sum_average(方法名称不变),Radl-WS 中输入 in 操作(a:array)转换成 Java 方法中的参数输入(a[ ]),其中 array→int 表示数组的类型为整型,对应 Java 中的整型数组定义.

(ii)前后置断言转换. 关键词对应转换,Radl-WS 代数规范中前置断言 Q 与后置断言 R 分别对应 JML 关键词 requires 及 ensures. 将 Radl-WS 描述的规约转换成符合 JML 语法规范的描述. 在此例中,前置断言转换不变. 对于后置断言部分在执行结束后的返回值 JML 中使用关键词\result,其中 Radl-WS 规范中#(a)表示求数组长度,对应地在 JML 中调用 length. 其他部分对应转换,主要是把在 Radl-WS 中前后置断言的谓词表达式转换成符合 JML 语言规范的描述.

(iii)异常处理. 在头部中添加关键词 normal_behavior 表示正常执行部分. 异常处理关键词 exceptional_behavior,从 aslo 开始到最后是异常处理部分.

(iv)其他部分的关键词为变量约束,根据具体情况来添加符合 JML 规范的描述即可.

将图 6 的 Radl-WS 需求规约按照上述转换步骤进行转换,转换后的 JML 代码如图 7 所示.

JML 语言能较好地与 Radl-WS 语言的关键部分(前后置断言)对应起来,其他部分按照语义使用 JML 规范书写即可.

3.2 对 Java 代码进行验证

本文采用符号执行的自动化验证工具 SED 对

JML 方法契约验证. 如图 8 所示,使用方法契约及循环不变量来代替内联和展开,从而保证有限深度的符号执行,方法 indexOf 返回过滤器接受的在给定数组中第 1 个索引. 接口 Filter 不用具体实现,但它的方法 accept 可用 JML 指定. 如图 8 所示,在对该方法进行验证时,在符号执行树左侧任意循环迭代中,循环结束节点出现交叉线,这表明该方法验证有误.

由图 8 可知:在 if 分支最左边的分支结束节点上出现了交叉线条,这说明程序存在问题. 分析 if 条件是否满足,并找到原因是 i 没有增加,这里不满足循环不变量的递减子句 decreasing array. length $-i$ 的递减要求,所以在 if 子句中最后加上 $i++$ 即可满足,若抛出异常或跳出循环(break、continue 或 return),则循环不变量不必保持,并且在循环外继续执行. 修改后再验证,结果如图 9 所示.

```
/*@ normal_behavior
@ requires a! =null;
@ ensures \result = =(\sum int i;i≥0 && i < a.
length;a[i]);
@ also
@ exceptional_behavior
@ requires a = =null;
@ signals_only NullPointerException;
@ signals (NullPointerException) true;
@ */
Public static/*@ pure @ */int sum_average(/*
@ nullable @ */int[] a){
}
```

图 7　sum_average JML 方法契约

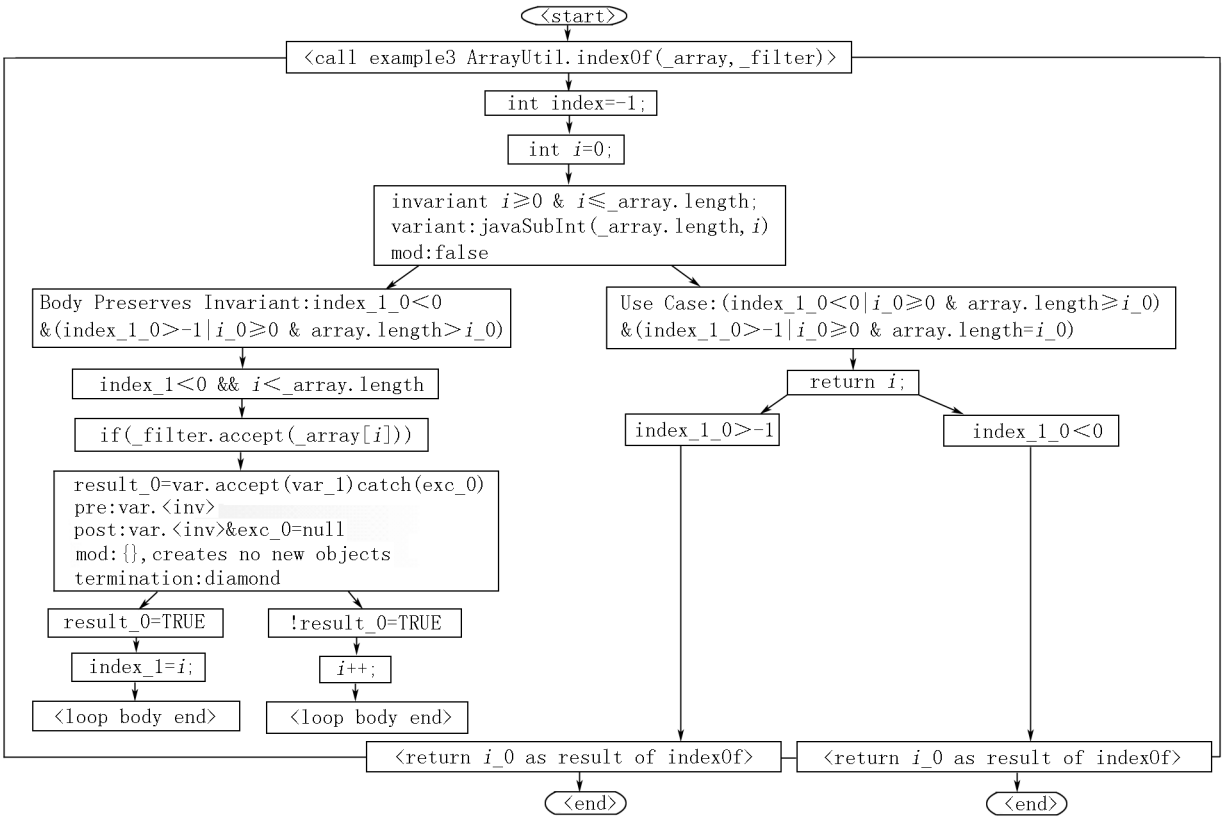

图 8　过滤器查找数组元素验证失败

图 8 的程序代码如下:

```
public class ArrayUtil {
    /*@ normal_behavior
    @ requires \invariant_for(filter);
    @ */
public static int /*@ strictly_pure @ */ indexOf
(Object[] array,
        Filter filter) {
    int index = -1;
    int i = 0;
    /*@ loop_invariant i ≥ 0 && i ≤ array.
length;
    @ decreasing array.length - i;
    @ assignable \strictly_nothing;
    @ */
    while (index <0 && i < array.length) {
        if (filter.accept(array[i])) {
            index = i;
```

```
            }
            else {
                i++;
            }
        }
        return i;
    }
    public static interface Filter {
        /*@ normal_behavior
        @ requires true;
        @ ensures true;
        @ */
        public boolean /*@ strictly_pure @ */ accept(/*@ nullable @ */ Object object);
    }
}.
```

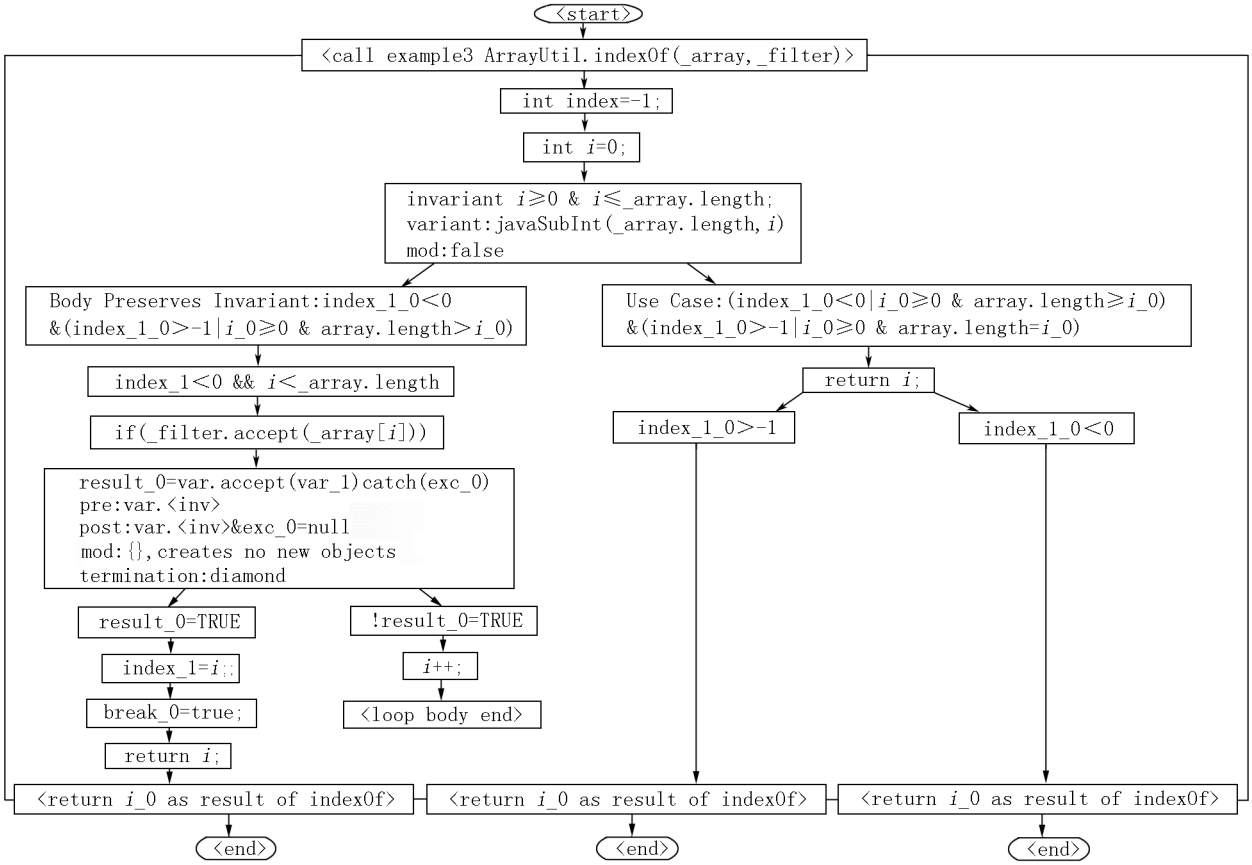

图 9 修改后验证成功

图 9 的程序代码如下:

```
public class ArrayUtil {
    /*@ normal_behavior
    @ requires \invariant_for(filter);
    @ */
    public static int /*@ strictly_pure @ */ indexOf(Object[] array,
            Filter filter) {
        int index = -1;
        int i = 0;
        /*@ loop_invariant i≥0 && i≤ array.length;
        @ decreasing array.length - i;
        @ assignable \strictly_nothing;
        @ */
        while (index < 0 && i < array.length) {
            if (filter.accept(array[i])) {
                index = i;
                break;
            }
            else {
                i++;
            }
        }
        return i;
    }
    public static interface Filter {
        /*@ normal_behavior
        @ requires true;
        @ ensures true;
        @ */
        public boolean /*@ strictly_pure @ */
```

```
accept(/*@ nullable @ */ Object object);
        }
    }
```

使用该方法可对上述求数组平均值 sum_average 方法的验证,根据 Radl-WS 转换生成的 JML 方法契约,使用 test 函数来调用该方法,进而验证sum_average 方法是否正确. 其中,在使用符号执行工具对该方法进行验证时,需要在符号执行选项设置中选择需使用方法. 在具体执行时应根据自己的验证条件及需求设置合适的符号执行选项. sum_average 方法验证结果如图 10 所示.

在符号执行或者验证中,若在某个节点上出现了交叉线条,则该程序或者方法契约不正确. 具体操作步骤及图标含义参见文献[15]. 在图 10 的整个执行树中每一个节点都没有交叉线条,这说明此方法契约正确,验证成功.

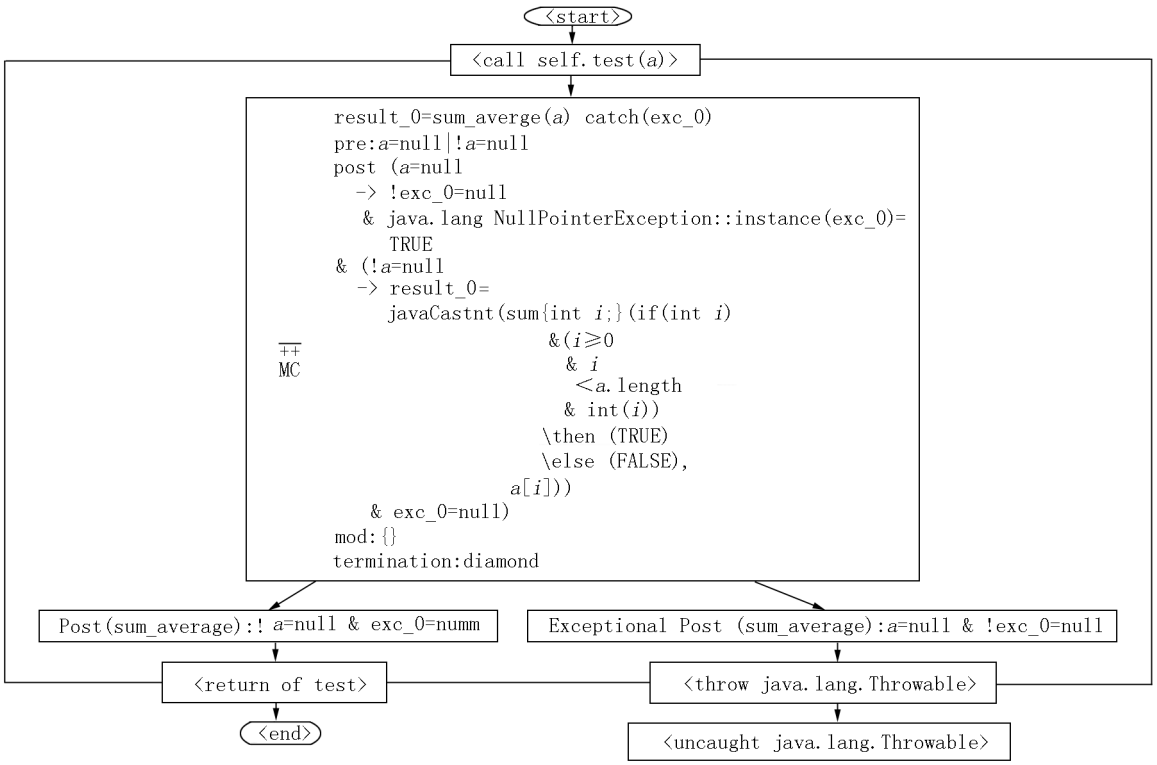

图 10　sum_average 方法契约验证执行树

4　案例:PayPal Web 服务

4.1　服务业务流程

以 PayPal Web 服务为例分别对数据建模与服务建模,其中服务模型会调用数据模型. 本文采用基于模型驱动的 3 阶段方法对 Java 代码符号执行与验证,分别对 PayPal 数据模型与服务模型进行符号执行与验证[21].

首先对 Web 服务进行需求分析,并使用 Radl-WS 语言进行需求建模. 在案例研究分析后,创建数据模型,根据 CRUD 原则,新增 createRecord、查询 findRecordBykey 及更新 updateRecord 和操作删除 deletRecord. 定义了交易记录类 TransRecord 和用户个人信息记录类型 PayerInfoType.

创建服务模型包括 setExpressCheckout 启动付款交易服务、getExpressCheckoutDetail 获取买方信息、doExpresssCheckout 更新余额完成交易. PayPal Web 服务业务具体调用如图 11 所示.

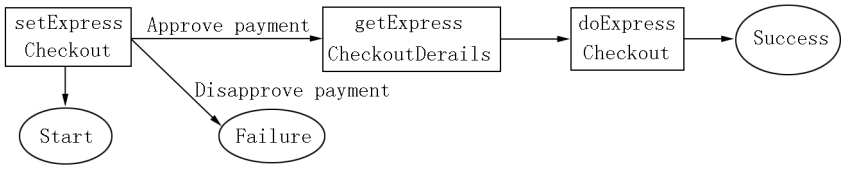

图 11　PayPal 服务调用流程

当买方在商品加入购车后将点击支付,这时会启动 setExpressCheckout 服务,并返回时间令牌值.

然后登录网站,确认用户信息.若用户批准付款,则调用 getExpressCheckoutDetail 服务,返回买方用户信息;若用户不同意付款,则结束服务.用户同意,确认订单,完成支付,调用 doExpresssCheckout 服务,更新余额,完成订单.

4.2 符号执行

4.2.1 *数据模型符号执行* 在模型驱动的 Web 服务建模与 3 阶段模型转换方法的工作基础上,本文对 3 阶段转换生成的 Java 可执行代码进行测试与验证.分别从数据模型与服务模型 2 个方面进行符号执行与验证来检验转换结果的正确性.由于篇幅原因,所以本文仅从数据模型中选择其中的一个方法(查询 findRecordBykey)进行符号执行.findRecordBykey 方法的 Java 代码如下:

```
publicstatic/*@ pure @*/TransRecord /*@ pure @*/findRecordByKey(/*@ nullable @*/ String token){
    TransRecord rec = newTransRecord();
    /*@ loop_invariant i≥0 && i≤transEntity.length;
    @ decreases (transEntity.length - i);
    @ assignable i;
    @*/
    for (int i=0; i<transEntity.length;i++){
    if(token = =transEntity.[i].getToken())
    {
    rec.setToken(transEntity.[i].getToken());
    rec.setTransAmount(transEntity.[i].getTransAmount());
    rec.setPaymentStatus(transEntity.[i].getPaymentStatus());
    rec.setPayerInfo(transEntity.[i].getPayerInfo());
    returnrec;
    }
    }
    returnnull;
    }.
```

符号执行涉及循环,本文使用循环规范处理,如上代码中的第 2 ~4 行.在符号执行选项设置中选择使用循环不变式.对 findRecordBykey 方法符号执行,结果如图 12 所示.

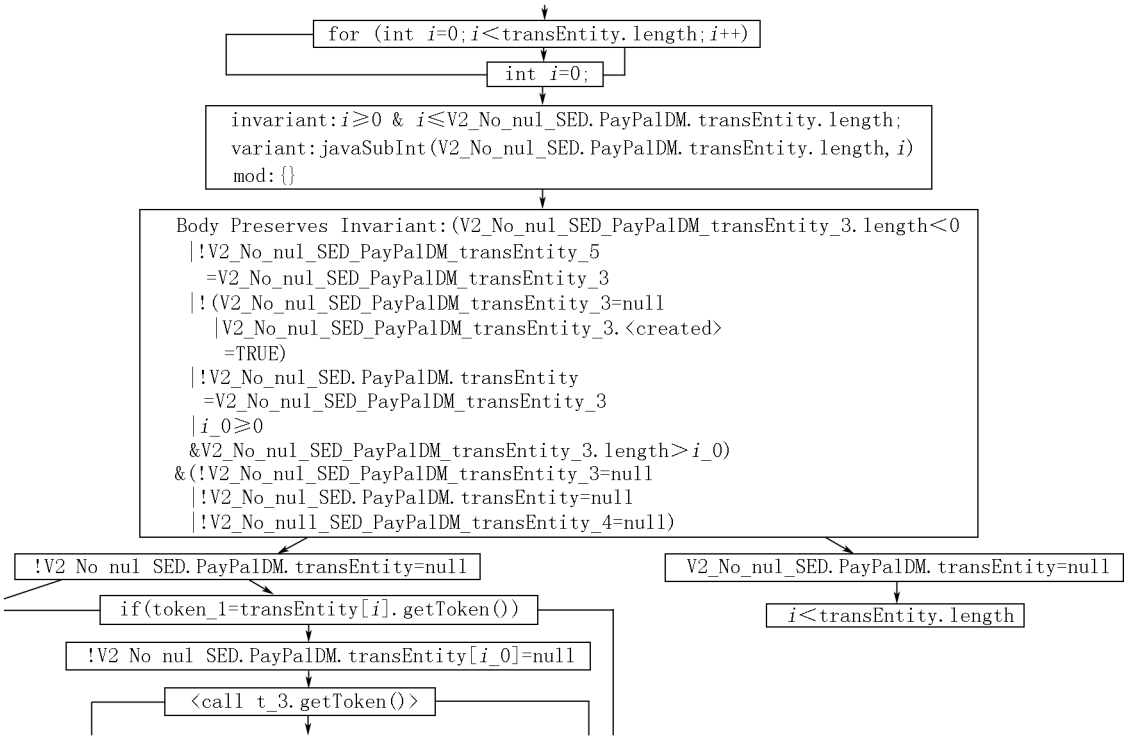

图 12 findRecordBykey 符号执行

案例程序较复杂,符号执行树较大,本文截取关键部分展示.符号执行树与预期结果走向相同,且分支与预先分析设计相同.结果表明 findRecordBykey 执行结果正确.数据模型的其他方法同理执行,符号执行均成功,这表明本文案例数据模型符号执行测试成功.

4.2.2 *服务模型符号执行* 在数据模型成功后,同理可使用符号执行对服务模型测试.本文选取在服务模型中的 setExpressCheckout 服务符号执行测试.该方法的 Java 代码如下:

```
public static/*@ pure @*/ String setExpressCheckout(/*@ nullable @*/intpaymentAmount){
```

```
ppdm.createRecord(rec);
    return rec.getToken();
```

}.

即对该方法 Java 代码进行符号执行(见图 13).

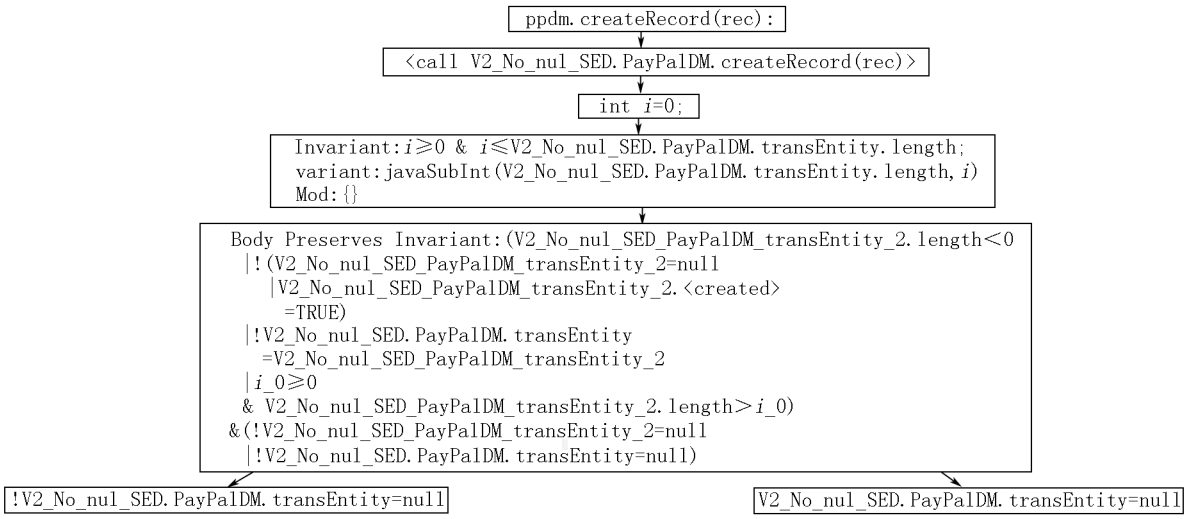

图 13　setExpressCheckout 符号执行

通过符号执行树及节点可以看出该方法执行成功.

数据模型与服务模型符号执行均正确执行,这表明案例符号执行测试成功.

4.3　形式化验证

4.3.1　*数据模型验证*　根据 Radl-WS 规则,对该案例中的新增、查询、更新及删除操作进行数据建模. 为了验证转换后的 Java 代码的正确性,本文将 Radl-WS 需求模型转换成 JML 方法契约,将 JML 方法契约加入 Java 代码中并验证. 以查询操作为例,Radl-WS 代码如下:

其中 type 是类型定义,定义了 2 个记录及变量,该规约规格名为 PayPalDM):

```
type TransRecord = record
    token:string;
    transAmount:real;
    payerInfo:PayerInfoType;
     paymentStatus:{Processed, Inprogress, Denied};
end
type PayerInfoType = record
    age:integer;
    name:string;
    sex:string;
    shipping address:string;
email:string;
end
spec PayPalDM
    Imports:TransRecord,PayerInfoType
    sorts:
      transEntity:set(TransRecord);
    ops:
    observer:
        findRecordBykey: string → table (TransRecord)
        |[in key:string;out result.transEntity:table(TransRecord)]|
        {Q:true}
        {R:result.transEntity =
        Π{token,transAmount,payerInfo,paymentStatus}σ{this.token = key}(this.transEntity)}
    axioms:
end-spec.
```

Radl-WS 建模语言转换成的 JML 方法契约代码如下:

```
/*@ normal_behavior
  @ requirestransEntity != null;
  @ ensures (\exists int i;0 <= i && i < transEntity.length;transEntity[i].getToken() == token);
  @ also
  @ exceptional_behavior
  @ requires transEntity == null;
  @ signals_only NullPointerException;
  @ signals (NullPointerException) true;
  @ */
```

}

public static TransRecord /*@ pure @*/ findRecordByKey(/*@ nullable @*/ String token) {}.

本文以查询操作为例,对数据模型中的 findRecordBykey 查询操作进行验证,验证结果的执行树如图 14 所示.

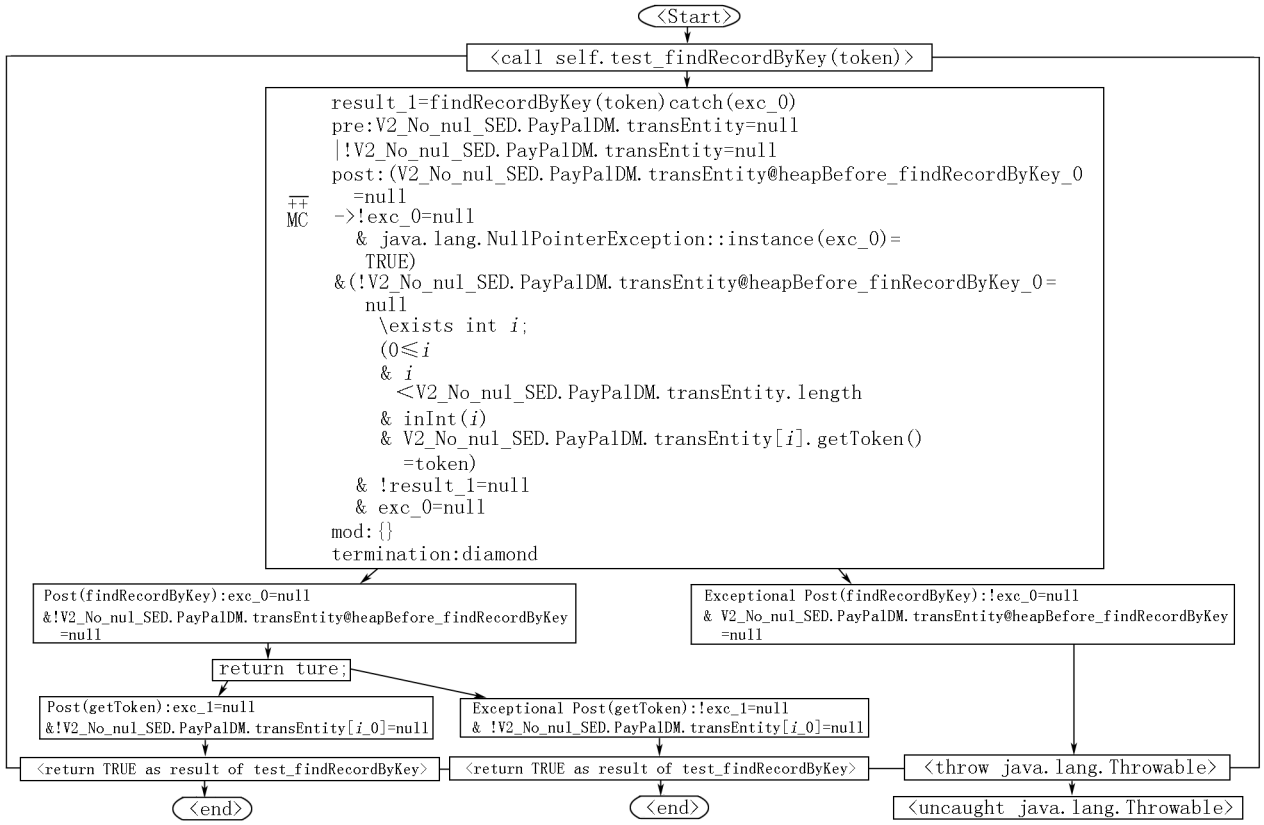

图 14 findRecordBykey 方法契约验证执行树

从图 14 可以看出:该方法的每一个节点图标均未出现交叉线. 这表明执行成功,即该方法契约正确. 同理,表明对数据模型中每一个方法契约进行验证,均成功执行,PayPal 数据模型验证成功.

4.3.2 服务模型验证 基于模型驱动的 Web 服务不仅能对数据模型验证,而且也可对服务模型进行验证. 同理,使用 Radl-WS 服务建模语言,关键代码如下:

setExpressCheckout:real→string

|[in sPaymentAmount:real;out result:string;]|

{Q:true}

{R: rec. token ≠ null ∧ ppdm. findRecordByKey(rec. token) = null}

{R:rec. payerInfo ≠ null ∧ rec. transAmount = sPaymentAmount ∧ result = rec. token }

{R:rec. paymentStatus = InProgress}

{R:ppdm′ = ppdm. createRecord(rec)}.

由于篇幅原因,所以本文仅以 setExpressCheckout 服务为例来说明对服务模型的验证. 首先将 Radl-WS 需要模型转换成的 JML 方法契约代码如下:

/*@ modifiesrec,URL,ppdm;

@ requiresURL! = null && URL = = Checkout_URL;

@ ensures(rec.getToken() ! =null) &&@ (\old(ppdm). findRecordByKey(rec. getToken()) = =null);

@ ensuresrec. getPayerInfo() ! = null && rec. getTransAmount() = = \old(paymentAmount);

@ ensures (URL = = success_URL &&rec. getPaymentStatus() = = "InProgress") ||

@ (URL = = cancel_URL && rec. getPaymentStatus() = = "Denied");

@ ensures\old(ppdm). createRecord(rec);

@ ensures\result = = rec. getToken();

@ also

@ exceptional_behavior

@ requires URL = = null;

@ signals_only NullPointerException;

@ signals (NullPointerException) true;

@ * /
public String setExpressCheckout (int sPaymentAmount) { } .

然后使用自动化工具对服务模型的方法契约进行验证,服务模型验证结果的执行树如图 15 所示.

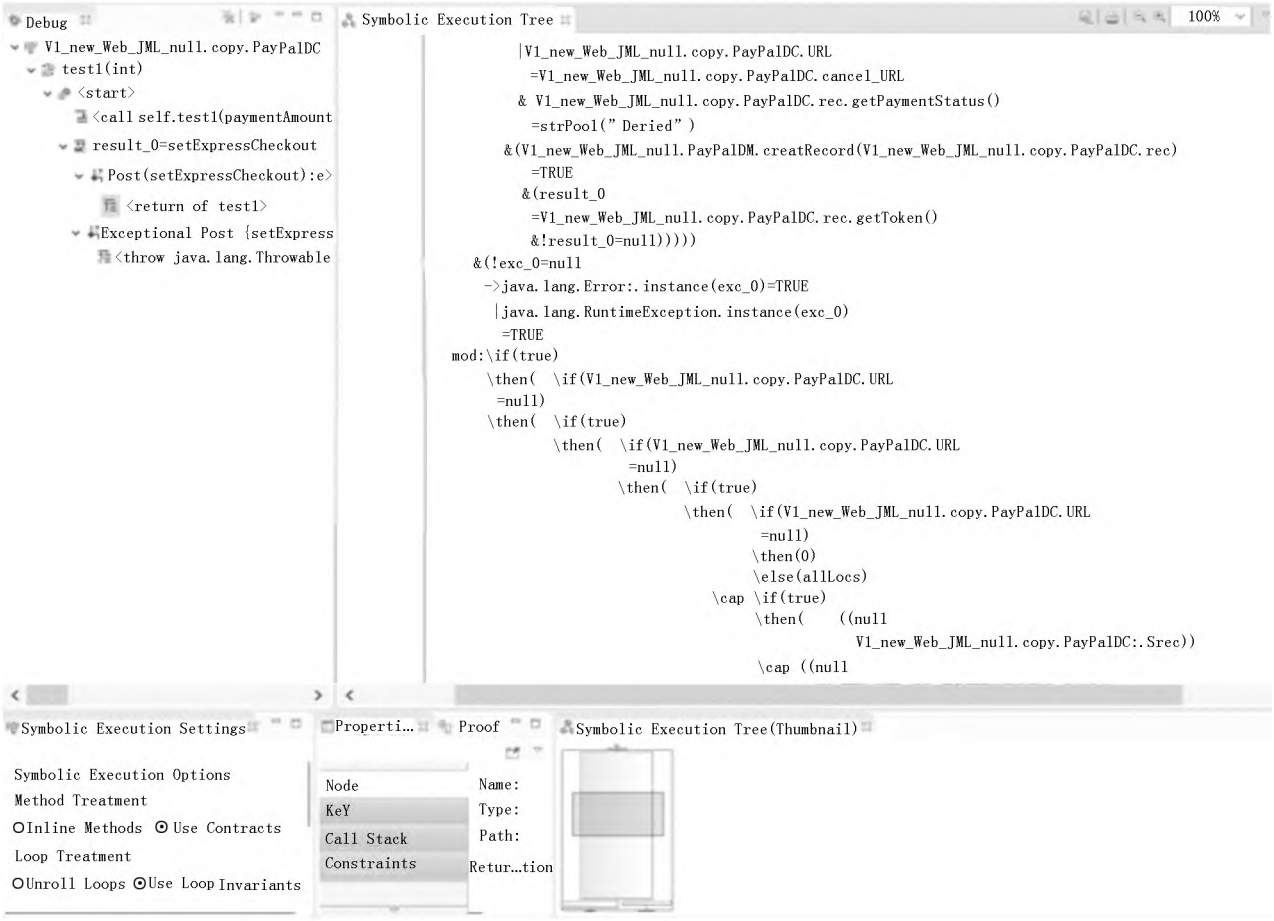

图 15　setExpressCheckout 方法契约验证执行树

该工具图形界面显示方法在契约验证时会把细节显示在方框中. 图 15 的契约节点处方框较大,可以从左侧节点层级目录中看出:该方法每个节点图标均未出现交叉线. 这表明执行成功,即该方法契约正确. 使用同样的方法对剩下的服务模型进行验证,执行结果均成功,这表明 PayPal 服务模型验证成功.

5　总结与展望

基于前期工作的研究,本文提出一种基于模型驱动的 Web 服务符号执行与验证方法,该方法是对基于模型驱动的 3 阶段转换生成 Java 代码进行符号执行测试与验证. 本文提出了运用符号执行的方式对 Web 服务功能正确性进行验证,并采用基于 Eclipse平台扩展的交互式符号执行调试器 SED,使用自动化工具对结果检验,提高了 Web 服务开发的可靠性.

通过 PayPal 实例来展示本方法的实际效果. 首先对 3 阶段转换生成的数据模型与服务模型的 Java 代码进行符号执行;在成功执行后,在 Radl-WS 服务建模语言的基础上,将 Radl-WS 语言描述的需求模型转换成 JML 方法契约,并将此加入 Java 代码中分别对数据模型及服务模型进行验证;在自动化工具的支持下,数据模型与服务模型的符号执行与验证均成功,这说明该方法有效、可靠.

基于本文工作与 3 阶段转换生成方法,计划将此方法扩展到对 Web 服务组合的测试与验证,以及在 Web 服务组合的基础上加上可交易服务,验证可交易的 Web 服务组合可靠性与正确性.

6　参考文献

[1] SOURI A,RAHMANI A M,NAVIMIPOUR N J,et al. A symbolic model checking approach in formal verification of distributed systems [J]. Human Centric Computing and Information Sciences,2019,9(1):1-27.

[2] 彭焕峰,黄纬,范大娟,等. Web 服务演化综述 [J]. 科

学技术与工程,2015,15(30):63-70.
[3] 丁志军,周泽霞. Web 服务组合测试综述 [J]. Journal of Software,2018,29(2):299-319.
[4] 廖军,谭浩,刘锦德. 基于 Pi-演算的 Web 服务组合的描述和验证 [J]. 计算机学报,2005,28(4):635-643.
[5] 罗异. 基于模型驱动架构的 Web 代码生成方法研究与应用 [D]. 重庆:重庆邮电大学,2018.
[6] DO KIM H. BPMN-based modeling of B2B business processes from the neutral perspective of UMM/BPSS [EB/OL]. [2021-03-16]. https://ieeexplore. ieee. org/document/4590645.
[7] SADOVYKH A,DESFRAY P,ELVESAETER B,et al. Enterprise architecture modeling with soaML using BMM and BPMN-MDA approach in practice [EB/OL]. [2021-03-16]. https://ieeexplore. ieee. org/document/5783155.
[8] 徐华珍,薛锦云,朱小征. Apla→Java 程序生成系统中泛型机制实现方法研究 [J]. 江西师范大学学报(自然科学版),2017,41(1):52-55.
[9] 石海鹤,薛锦云. 基于 PAR 的算法形式化开发 [J]. 计算机学报,2009,32(5):982-991.
[10] 谢武平. Radl→Apla 程序生成系统及其可靠性研究 [D]. 南昌:江西师范大学,2009.
[11] 石海鹤. 支持泛型程序设计的 Apla-Java 自动程序转换系统 [D]. 南昌:江西师范大学,2004:1-74.
[12] 左正康,薛锦云. Apla 中泛型约束机制研究 [J]. 软件学报,2015,26(6):1340-1355.
[13] 叶志斌,严波. 符号执行研究综述 [J]. 计算机科学,2018,45(6A):28-35.
[14] CADAR C,SEN K. Symbolic execution for software testing: three decades later [J]. Communications of the ACM,2013,56(2):82-90.
[15] HENTSCHEL M,BUBEL R,HÄHNLE R. The symbolic execution debugger (SED):a platform for interactive symbolic execution,debugging,verification and more [J]. International Journal on Software Tools for Technology Transfer, 2019,21(5):485-513.
[16] AHRENDT W,BECKERT B,HÄHNLE R,et al. Verifying object-oriented programs with KeY:a tutorial [EB/OL]. [2021-03-19]. https://dl. acm. org/doi/10. 5555/1777707. 1777713.
[17] 王昌晶,薛锦云. Radl 形式规格说明相对正确性研究 [J]. 软件学报,2013,24(4):715-729.
[18] 王昌晶. 基于扩展逻辑变换系统 μTS 证明循环变换正确性 [J]. 计算机研究与发展,2012,49(9):1863-1873.
[19] 张广泉. 形式化方法导论 [M]. 北京:清华大学出版社,2015:1-256.
[20] 张琦,王昌晶,罗海梅,等. WSDL→Radl-WS 生成方法及自动转换系统 [J]. 江西师范大学学报(自然科学版),2018,42(3):298-303.
[21] SALEH I,KULCZYKI G,BLAKE M B,et al. Formal methods for data-centric Web services:from model to implementation [EB/OL]. [2021-03-20]. https://ieeexplore. ieee. org/document/6649596.

The Web Service Symbolic Execution and Verification Based on Model-Driven

WANG Changjing[1,2],CHEN Xi[1],DING Xilong[1],LUO Haimei[3],ZUO Zhengkang[1*]
(1. College of Computer Information Engineering,Jiangxi Normal University,Nanchang Jiangxi 330022,China;
2. Management Science and Engineering Research Center,Jiangxi Normal University,Nanchang Jiangxi 330022,China;
3. College of Physics and Communication Electronics,Jiangxi Normal University,Nanchang Jiangxi 330022,China)

Abstract:Web service testing and verification are the keys to ensuring the correct function of Web services. Most current research on Web services cannot traverse the program path exhaustively,and cannot guarantee the completeness of the analysis. For this problem,the symbolic execution and formal verification are performed on the Java code generated by the conversion based on the previous model-driven three-stage Web service model conversion generation method. The symbolic execution method can analyze all paths of program running,provide test cases with high coverage for program testing and trigger deep program errors. Furthermore,the JML method contract is added to the Java code to verify the Web service. In the PayPal Web service case,the model-driven method is adopted to generate Java code from Web service model transformation generation method,and the automated tools are used to perform symbolic execution of the Java code. In addition,the Radl-WS service modeling Language is transformed into JML method contracts and the formal validation of Java code is performed. Symbolic execution and verification methods ensure the correctness and credibility of the generated Java code and improve the degree of automation.

Key words:Web service;Radl-WS;Java;symbolic execution;verification

(责任编辑:冉小晓)