BPEL 程序的变异测试优化技术与集成化支持工具研究

潘琳

北京科技大学

密　　级：公开

论文题目：**BPEL 程序的变异测试优化技术与集成化支持工具研究**

学　　号：G20148605

作　　者：潘琳

专 业 名 称：计算机技术

2016 年 12 月 21 日

BPEL 程序的变异测试优化技术与集成化支持工具研究

Research on Optimization Techniques of Mutation Testing for BPEL Programs and Their Supporting Tool

研究生姓名：潘琳

指导教师姓名：孙昌爱

北京科技大学计算机与通信工程学院

北京 100083，中国

Master Degree Candidate： Pan Lin

Supervisor： Sun Changai

School of Computer and Communication Engineering

University of Science and Technology Beijing

30 Xueyuan Road，Haidian District

Beijing 100083，P.R.CHINA

分类号：TP311　　密　　级：公开

ＵＤＣ：004.41　　单位代码：１０００８

北京科技大学硕士学位论文

论文题目：BPEL 程序的变异测试优化技术与集成化支持工具研究

作者：潘琳

指 导　教 师：孙昌爱　　教授　　**单位：**北京科技大学

指导小组成员：　　**单位：**

指导小组成员：　　**单位：**

论文提交日期：2016 年 12 月 21 日

学位授予单位：北 京 科 技 大 学

致　谢

时光飞逝，两年半的研究生生活很快就要划上句点。在这期间，我的知识水平和实践能力上了一个新的台阶，各方面的素质得到了很大的提高，当然这离不开我的导师孙昌爱老师的耐心指导与教育。

首先，衷心感谢我的导师孙昌爱老师对本次毕业设计的耐心指导与大力支持。本文的研究工作从选题、开题、中期及最后的论文修改等各个方面都离不开孙老师耐心的帮助和教导。其次，研究生期间，孙老师一丝不苟的工作态度，宽厚的生活作风，都给我留下了深刻的印象，对我孜孜不倦的耐心教导，使我取得了进步，也为以后的工作和生活打下了基础。

同时感谢在硕士阶段所有教过我的老师，从他们身上我领略到学术大师的风采，研究人员该有的严谨态度。也要感谢在我的研究阶段帮助过我的王巧玲师姐和王真师妹，感谢他们对我的无私帮助。

最后，非常感谢我的父母，是她们始终站在我背后给予我无私的爱，精神上不断的鼓励，物质上不断的支持才使得我不断前进。

摘　要

在云计算环境中越来越多的应用程序部署为 Web 服务，为了实现复杂的功能，通常需要组装多个 Web 服务。BPEL 是一种基于 XML 格式的服务组装语言，广泛用于编排多个 Web 服务实现复杂、灵活的业务流程。由于 Web 服务的动态性、松耦合特性、部署与运行于开放的网络环境，如何保证服务组装程序的可靠性尤显重要。变异测试是一种基于故障的软件测试技术，广泛用于评估测试用例集的充分性与测试技术的故障检测能力。然而，变异测试产生大量变异体，从而导致测试开销大的问题，不利于在实践中广泛应用。

为了增强变异测试的实用性，本文研究如何降低面向 BPEL 程序的变异测试的开销问题。从变异算子之间包含关系、变异算子优先级、二阶变异三个方面，提出了一组面向 BPEL 程序的变异测试的优化技术，并开发了相应的支持工具。本文取得的主要成果如下：

(1) **提出了三种面向 BPEL 程序的变异测试优化技术：**通过分析 BPEL 程序的变异算子，定义识别具有包含关系的变异算子规则，提出了基于包含关系的变异算子优化技术；通过度量变异算子的质量，并依据变异算子质量对变异算子进行优先级排序，提出基于变异算子优先级的优化技术；此外，还提出了面向 BPEL 程序的二阶变异体优化技术。

(2) **开发了面向 BPEL 程序的变异测试集成化支持工具 μBPEL：**μBPEL 支持 BPEL 程序变异测试的全过程，包括变异体程序的自动生成、测试用例集生成、测试执行和测试结果统计，同时支持本文提出的变异测试优化技术。

(3) **经验研究：**使用 6 个 BPEL 程序实例验证并评估提出的面向 BPEL 程序的变异测试优化技术的有效性。

本文提出的面向 BPEL 程序的变异测试优化技术，可以有效地减少变异体数目并不显著降低变异测试有效性。相应的支持工具提升变异测试过程的自动化程度，进一步提高变异测试的效率。

关键词： 服务组装，BPEL，变异测试，测试工具，包含关系

Research on Optimization Techniques of Mutation Testing for BPEL Programs and Their Supporting Tool

Abstract

Distributed applications are increasingly deployed as Web services in the cloud computing environment. Multiple services are normally composed to fulfill complex functionalities. Business Process Execution Language (BPEL) is an XML-based service composition language that is widely used to define a complex business process by orchestrating multiple services. Due to unique features of Web services, such as dynamics, loose coupling, and open deployment and execution environment, it is an important issue how to assure the quality of Web services and their compositions. Mutation testing is a kind of fault-based software testing technique that has been widely used for evaluating the adequacy of test suites and the fault-detection effectiveness of software testing methods. However, mutation testing failed to be widely practiced because it normally generates a large amount of mutants and thus incurs a high cost.

In order to improve the practicability of the mutation testing technique, this thesis studies how to decrease the cost of mutation testing for BPEL programs. We propose a series of optimization techniques in terms of subsuming relation of mutation operators, the prioritization of operators, and second-order mutation, and develop a tool named μBPEL to implement these optimization techniques. The main contributions made in this thesis are as follows:

(1) **Three optimization techniques of mutation testing for BPEL programs**, including a subsuming relation based optimization technique that is based on the analysis of mutation operators and the subsuming relation among operators, an operator priority based optimization technique that is based on ranking of mutation operators using a quality merit, and a second-order mutant optimization technique for BPEL programs.

(2) **An integrated mutation testing system for BPEL programs called μBPEL,** which supports the whole lifecycle of the mutation testing for BPEL programs, including mutant generation, optimization techniques, test case generation, test execution, and analysis of testing results.

(3) **An empirical study** in which six representative BPEL programs are used to validate and evaluate the effectiveness of the proposed optimization techniques.

In summary, the proposed optimization techniques of mutation testing for BPEL programs are able to reduce the number of mutants without significantly jeopardizing its fault detection effectiveness. The supporting tool further improves the automation of mutation testing and thus increases its efficiency.

Key Words: Service Compositions, BPEL, Mutation Testing, Testing Tool, Subsuming Relation

目　　录

1 引言

本章介绍课题研究背景与意义、研究内容与成果，以及论文组织结构。

1.1 研究背景与意义

面向服务的架构 SOA（Service-Oriented Architecture）逐渐成为开发应用程序的主要范式[1]。遵循 SOA 架构的 Web 服务受到越来越多的重视。然而，单一的 Web 服务只能提供一些简单的功能，无法满足日益复杂的需求，需要将多个服务组装以实现更复杂和灵活的业务流程。BPEL（Business Process Execution Language）[2]是一种基于 XML 格式的服务组装语言，可以编排不同的 Web 服务实现复杂的业务流程。同时，也将自身流程暴露为 Web 服务，支持更大粒度的服务组装。由于各个 Web 服务相互协作的动态性，互联网环境的开放性，Web 服务组合的松耦合性所导致的组合和运行 Web 服务的不确定性[3]，对 Web 服务及其服务组装的质量保证带来挑战。

测试是公认的质量保证、验证和确认的有效手段。变异测试[4]是一种基于故障的软件测试技术，具有较强的错误检测能力，广泛用于评估和改进测试用例集完备性和测试技术的有效性。

在 BPEL 程序的变异测试研究方面，Estero-Botaro 等人[5]提出一系列面向 BPEL 程序的变异算子，为 BPEL 程序的变异测试提供基础，并评估不同类型的变异算子的适用性和有效性[6]。课题组前期工作对 BPEL 程序的变异测试也进行了一系列研究[7,8]，提出一个面向 BPEL 程序的变异测试框架，并开发相应支持工具。此外，还采用 6 个 BPEL 实例程序对变异算子进行经验研究，评估面向 BPEL 变异算子的有效性和变异算子被检测到的相对难易程度，寻找精简变异算子的方案，以提高 BPEL 变异测试效率。

然而，变异测试技术由于变异体数量庞大、等价变异体识别困难、缺乏相应自动化支持工具等原因，难以在实际中广泛应用。目前，研究人员主要从减少变异体数量和降低变异体执行时间两个角度入手，对变异测试技术进行优化来减少测试开销、提高测试效率。

本文从变异算子包含关系、变异算子优先级、二阶变异三个方面，研究面向 BPEL 程序的变异测试优化技术，并开发相应的支持工具。

1.2 研究内容与成果

BPEL 在 Web 服务组装领域日趋成熟、业务流程逐渐庞大且复杂，通过

测试来保证 BPEL 流程的质量和可靠性也变得越来越重要。变异测试是一种有效的故障检测技术[9]，但由于测试开销大等问题难以广泛在实际中应用。本文研究如何降低面向 BPEL 程序的变异测试的开销问题，从变异算子之间包含关系、变异算子优先级、二阶变异体优化三个方面，提出一组面向 BPEL 程序的变异测试优化技术，并开发了面向 BPEL 程序的变异测试集成化支持工具，来提高 BPEL 程序的变异测试过程的自动化程度。此外，采用经验研究，使用 6 个 BPEL 程序实例验证并评估所提出的面向 BPEL 程序的变异测试优化技术的有效性和工具的可用性。

本文研究工作得到了国家自然科学基金面上项目（项目编号：61370061）和北京市优秀人才培养项目（项目编号：2012D009006000002）的资助。

1.3 论文组织结构

本文的组织结构安排如下：

第一章， 引言，主要包括课题的研究背景与意义、研究内容与成果以及论文组织结构。

第二章， 背景介绍，主要包括相关概念及技术、国内外研究现状。

第三章， 介绍三种变异测试优化技术，包括原理描述、规则描述、一般步骤和举例说明。

第四章， 讨论 BPEL 程序的变异测试集成化支持工具设计与实现，包括需求分析、系统设计与实现以及系统演示。

第五章， 对 BPEL 实例进行经验研究，验证优化技术的可行性与有效性，包括实验对象的说明、实验一般步骤、度量指标以及实验结果分析。

第六章， 研究工作的总结以及未来的展望。

2 背景介绍

本章介绍 BPEL 与变异测试的相关概念、国内外研究进展、研究背景和意义以及论文组织结构。

2.1 相关概念与技术

2.1.1 BPEL

BPEL 是一种 OASIS 标准、实现 Web 服务组装的工业级编程语言[2]，基于 XML 格式，使用很多 XML 规范，包括 WSDL(Web Services Description Language)，XML Schema，XPath (XML Path Language)和 XSLT (Extensible Stylesheet Language Transformations)。其中，WSDL 消息和 XML Schema 的数据类型定义用来描述 BPEL 的数据模型；XPath 和 XSLT 用来提供 BPEL 的数据操作；WSDL 用来描述 Web 服务的接口信息。

BPEL 程序通常由变量声明(variable declaration)、伙伴链接声明（partner link）、处理器声明（handler）及业务流程描述四个部分组成[6]。图 2-1 展示的是一个 BPEL 程序，描述的是一个供销链管理的业务流程。其中，伙伴链接声明部分定义一系列参与交互的 Web 服务；变量声明部分定义 BPEL 流程中使用的数据变量；业务流程描述是实现特定工作流的主体，由一组活动（activity）及其之间的联系构成，这些活动可以分为基础活动和结构性活动两大类。基础活动是不可再分活动，只包含一个操作，例如从一个合作伙伴接收消息、发送消息到一个伙伴、给变量赋值等。典型的基础活动包括 *invoke*、*assign*、*throw*、*reply*、*receive*、*wait*、*empty* 等。结构性活动定义业务逻辑，提供执行的控制形式，包括 *switch*、*sequence*、*flow*、*while*、*pick* 等。一个结构性活动还可以包含其他基本活动和结构性活动。此外，BPEL 程序在活动中支持并发和同步机制，主要通过 *flow* 结构活动和带有 *link* 标签的 *flow* 来实现。活动可以有自己的属性和一组相关的容器，而且这些容器也可以包含自己的元素和属性。

BPEL 语言尽管与传统的程序设计语言一样有顺序、分支、循环等标准的控制结构，但也有其与众不同的特点，主要体现在如下四个方面[10]：

(1) BPEL 提供一种显式的集成机制组装 Web 服务，而这样的集成在传统程序中是隐式；

(2) BPEL 参与组装的服务可以采用不同语言实现，而传统的程序设计语

言模块通常需要由同一种语言实现；

(3) BPEL 程序表示为 XML 文件，不同于传统应用程序；

(4) BPEL 通过 *flow* 活动支持并发机制、带有 *link* 的 *flow* 活动支持同步机制。

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="SupplyChainProcess"
 targetNamespace="http://ustb.vxbpel.org"
 xmlns:suc="http://supplychain.org/wsdl/supply-chain/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" …>
 <import namespace="http://supplychain.org/wsdl/supply-chain/"
  location="SupplyChainArtifacts.wsdl"
 importType="http://schemas.xmlsoap.org/wsdl/" />
```

name space

```
 <partnerLinks>
   <partnerLink name="ordergoodsPL" partnerLinkType="suc:OrderGoodsLT"
    myRole="consumer" />
    …
 </partnerLinks>
```

partner links

```
 <variables>
     <variable name="input" messageType="order:OrderGoodsRequest" />
     <variable  name="output" messageType="order:OrderGoodsResponse"/>
  </variables>
```

variable declarations

```
<sequence name="main">
     <receive createInstance="yes" name="receiveInput"
      operation="OrderGoods" partnerLink="ordergoodsPL"
      portType="order:OrderGoodsPT" variable="input"/>

     <invoke name="Warehouse"  outputVariable="warehouseAmessagen"
      inputVariable="input"  operation="InquireGoods"
      partnerLink="warehouseAPL" portType="waa:WarehouseAPT">
     </invoke>
     <if name="warehouse">
          <condition>($warehouseAmessage.WarehouseAResponse &gt;=
$input.amount)
         </condition>
          <sequence>
            <invoke inputVariable="input" name="Shipper"
             operation="ShipGoods" outputVariable="shipperAmessage"
              partnerLink="shipperAPL" portType="sha:ShipperAPT">
             </invoke>
            …
         </sequence>
     <reply name="reply" operation="OrderGoods" partnerLink="ordergoodsPL"
      portType="order:OrderGoodsPT" variable="output">
     </reply>
 </sequence>
</process>
```

activity

interactions

图 2- 1 BPEL 流程示例

基于 BPEL 语言的这些特性，当开发和实现 BPEL 程序时，会产生一些有别于传统应用程序的新类型错误，这必然给基于 BPEL 程序的软件测试带

来一定挑战。

2.1.2 变异测试

变异测试（Mutation Testing）是 Richard A. DeMillo 提出的一种基于故障检测的软件测试技术[4]，广泛用于评估和改善测试用例集的有效性。

变异测试技术基于两个重要的假设[11]：熟练程序员假设和耦合效应假设。熟练程序员假设指程序员因编程经验丰富，编写出的代码与正确代码非常接近，仅包含一些细微的缺陷。耦合效应假设指复杂的缺陷可以耦合成一系列简单的缺陷，那么一个测试用例集就能够检测出简单缺陷，该测试用例也易于检测出更多的复杂故障。

变异测试基本原理（图 2-2）是：对于待测程序 *P*，应用变异算子植入合乎语法规则的故障，得到存在故障的程序集合，这些含有故障的程序称为 *P* 的变异体（Mutant）。变异算子（Mutation Operator）是植入的故障类型[11]。对于待测程序 *P* 和变异体集分别执行测试用例 *TS*，如果测试用例在原始程序和某变异体中展现出不同的执行行为（通常为不同输出结果），则称这个变异体“被杀死”；反之，称该变异体“存活”。

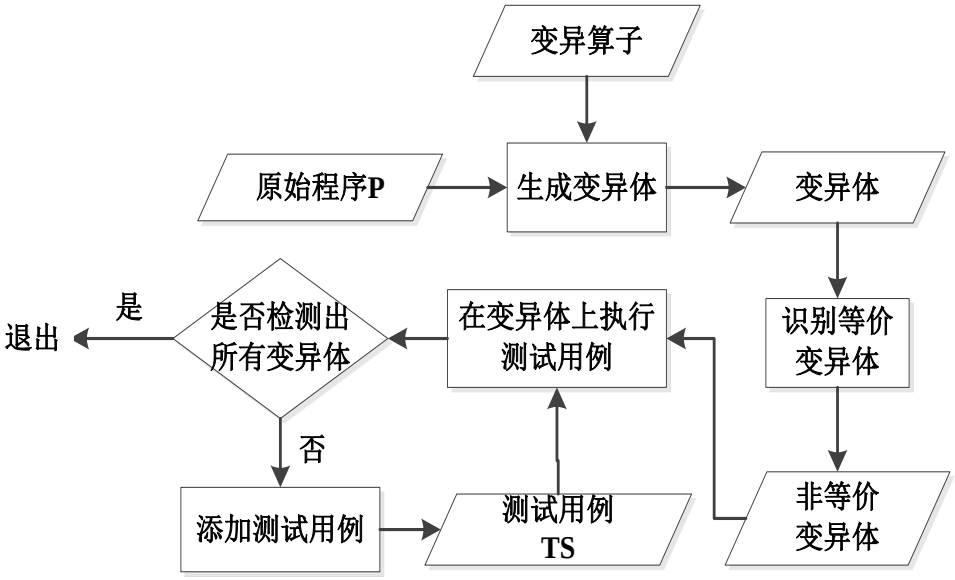

图 2- 2 变异测试基本原理图

测试用例执行完成后，“存活”的变异体分为两种。一种是可以被杀死的变异体，只是执行的测试用例集不够充分，没有检测出其包含的故障。对于这一类变异体需要根据其故障类型增加专门的测试用例将其杀死。另一种“存活”的变异体称为等价变异体，该类变异体语法上与原始程序 *P* 上不同，但

语义与原始程序 *P* 保持一致[12]。因此，不存在测试用例能够将此类变异体杀死。表 2-1 给出了一个典型的等价变异体示例，该例子中把 *for* 循环的判断条件“<”变异为“！=”，若循环体中没有对变量“*j*”的值进行修改，那么此处的变异不会影响程序的执行结果，也就是说该变异体 *P'* 与原始程序 *P* 执行任何测试用例的输出结果都一样，称 *P'* 为等价变异体。

表 2- 1 典型等价变异体示例

原始程序 P	变异体 P'

for (int j = 0; j <5 ; j++) {
a[j]+=1;
}
	for (int j = 0 ; j !=5; j++) {
	a[j]+=1;
	}

变异测试技术起初用来产生充分的测试用例集。例如，Fraser 和 Zeller[13] 研究如何应用变异测试技术产生单元测试用例等。此外，该技术也被用于检测 SQL 的注入漏洞问题[14]。最近，变异测试又被用于衡量不同代码的覆盖标准[15,16]和不同测试技术的有效性[17,18,19,20]。还有研究学者积极尝试研究 Web 服务的变异测试技术[21,22]。

变异测试普遍被认为具有很强的故障检测能力[4]，可以产生较好的测试效果[23]。然而，变异测试技术要应用到测试工业界，存在一些问题。例如，大量变异体的生成使得变异测试和分析时的开销极为高昂、对于等价变异体检测主要还靠手工方式完成、缺乏有效的自动化测试工具支持。

2.1.3 BPEL 程序的变异算子

变异算子是在符合语法规则的前提下，定义从原始程序生成极小差别的程序的转换规则[12]，用来模仿程序员在编写程序可能出现的错误。

Estero-Botaro 等人[5]针对 BPEL 程序提出了 26 种的变异算子。这 26 种变异算子被分为四类，分别是标识符替换变异算子、表达式变异算子、活动变异算子和异常与事件变异算子。活动变异算子又被分为与并发相关和与并发无关两类。通过这些变异算子来指导产生变异体，变异体作为测试数据来评估面向 BPEL 程序测试技术的故障检测能力[21]。最近，Garc á-Dom ínguez [24] 等人更新了 BPEL 变异算子种类。新增 *EIN*，*EIU*，*EAP* 和 *EAN* 四个变异算子到表达式变异算子类型中，并新添覆盖准则变异算子类型，包括 *CFA*，*CDE*，*CDC* 和 *CCO*，如表 2-2 所示。

图 2-3 示例了一个 BPEL 程序片段及可能出现的 *ISV* 类型错误。对于该 BPEL 实例，应用 *ISV* 变异算子，即将正确的 *reply* 活动变量引用 *output*，变

异成 *input*。不难看出，该变异体的输出将等于输入，导致程序出错。此类变异算子模拟了程序员弄错变量标识符的错误。

表 2-2 BPEL 的变异算子描述

变异算子	描述
标识符替换变异算子	
ISV	用一个相同类型的变量标识符替换另一个
表达式变异算子	
EAA	用一个相同类型的运算符对算术运算符(+，-，*，/)进行替换
EEU	移除所有表达式中的一元减法运算符
ERR	将关系运算符(<，>，<=，>=，=，!=)用另一相同类型的运算符进行替换
ELL	将逻辑运算符(and，or)用另一个相同类型的运算符进行替换
ECC	将路径运算符(/，//)用另一个相同类型的运算符进行替换
ECN	增加或减小一个单元中数字常量的值，添加或移除一个数字
EMD	将持续时间用 0 或者是其一半代替
EMF	将截止时间用 0 或者是其一半代替
EIN	插入 XPath 格式 not 到逻辑表达式中
EIU	插入 XPath 格式一元减法到算数表达式中
EAP	用表达式的绝对值替代该表达式
EAN	用表达式的绝对值的负值替代该表达式
活动变异算子 –与并发相关	
ACI	将 createInstance 属性从“yes”变为“no”。
AFP	将顺序的 forEach 活动改为并行的
ASF	用 flow 活动替换 sequence 活动
AIS	将一个域的 isolated 属性改为“no”
活动变异算子 –与并发无关	
AEL	删除一个活动
AIE	移除 if 活动中的 elseif 或 else 元素
AWR	用一个 repeatUntil 活动替换 while 活动
AJC	移除 joinCondition 元素
ASI	交换两个 sequence 孩子活动的序列
APM	移除 pick 活动的 onMessage 元素
APA	从 pick 活动或者事件处理程序中删除 onAlarm 元素
异常与事件变异算子	
XMF	移除错误处理程序中的 catch 或 catchall 元素
XMC	移除补偿处理程序的定义
XMT	移除终端处理程序的定义
XTF	用 throw 活动替代抛出故障
XER	移除 rethrow 活动
XEE	从事件处理程序中移除 onEvent 元素
覆盖准则变异算子	
CFA	用 exit 活动替代其他活动
CDE	应用判断覆盖准则到程序中
CCO	应用条件覆盖准则到程序中
CDC	应用判断/条件覆盖准则到程序中

```
//original “reply” activity
<reply name="reply" operation="OrderGoods"
    partnerLink="ordergoodsPL"
    portType="order:OrderGoodsPT" variable="output"
</reply>
```

```
//mutated “reply” activity
<reply name="reply" operation="OrderGoods"
    partnerLink="ordergoodsPL"
    portType="order:OrderGoodsPT" variable="input"
</reply>
```

图 2- 3 植入 ISV 类错误的 BPEL 程序片段

2.2 国内外研究现状

介绍变异测试优化技术与 BPEL 测试技术的国内外研究现状与进展。

2.2.1 变异测试优化技术研究现状

变异测试技术主要的不足是大量变异体的高计算消耗、等价变异体识别的人力消耗、缺乏相应的自动化工具等。目前，针对于变异测试优化技术的研究，主要从变异体选择优化和变异体执行优化两个角度展开[11]。变异体选择优化策略关注如何从生成的大量变异体中选择出典型的变异体，主要有随机选择法、聚类选择法、变异算子选择法、高阶变异体优化法和基于控制流的变异体精简技术等。

随机选择法是从生成的变异体集合中随机选择出一定比例的变异体，以替代全部的变异体进行测试。Wong 和 Mathur[25]通过对 Mothra 系统中的 22 种变异算子展开研究，设置一系列选择比例进行随机选择变异体。结果发现，相对于选择所有变异体，此方法得到的变异评分并没有明显降低，反而大幅度减少测试开销。但是，因为是随机选取变异体，所以每次的选择结果都会不同、变异得分不稳定、有可能不包含一些特殊的变异算子产生的变异体，从而导致变异体集对于检测这类错误不充分。

基于此类问题，研究人员又提出使用聚类算法对变异体进行聚类分析的方法，被相似的测试用例检测到的变异体划分在一个聚类中，从每个聚类中个选择出典型的变异体以替代全部的变异体进行测试。Hussain[26]选择两种聚类算法（k-means 和 agglomerative clustering），根据测试用例的检测能力对所有变异体进行聚类分析，选择出变异体。结果表明聚类选择法是有效的，可以对变异测试进行优化。然而聚类算法需要确定 *K* 值来进行类别划分，*K* 值选的越好聚类效果就越好。因此若 K 初始值选取不当，聚类选择法并不能达到较好的效果。

变异算子选择法从变异算子选择角度出发，在不影响变异得分的前提下，约简变异算子数量，从而减少变异体的生成。King 和 Offutt [27,28]对 FORTRAN 语言的变异算子进行研究，根据变异算子的测试有效性对其进行取舍，称为约束变异，获得了数目较少且更难被杀死的变异体。

高阶变异体优化方法关注于如何生成高阶变异体，以代替一阶变异体进行变异测试。高阶变异体是指在原始程序上执行多次变异算子产生的变异体。Langdon 等人[29]应用多目标方法来指导生成高阶变异体。实验表明，该方法可以有效地产生相比一阶变异体更难检测的高阶变异体，同时产生的等价变异体概率更小，减少了测试开销。

课题组前期工作提出一种基于控制流的变异体精简方法[30,31]，其基于随机选择法和变异算子选择法，通过在程序结构分析的基础上，定义模块深度和循环/分支深度的概念；设计了 4 种基于深度优先的变异体选择启发式规则，赋予启发式规则不同的优先级，进一步提出 4 种变异体精简策略；并使用 11 个 C 程序进行实验，结果表明精简策略确实有效地减少变异测试开销。

变异体执行优化方法关注于减少变异体的执行时间，主要包括变异体检测优化、变异体变异优化和并行执行变异体优化方法。Krauser 等人[32]在 SIMD（Single Instruction Multiple Data）计算机上提出一种可并发执行变异体的方法。此方法对于变异体的无变异部分执行一次，变异的部分进行并发执行，有效地减少变异体执行时间。

2.2.2 BPEL 测试研究现状

为了保证 BPEL 程序的质量，牟小玲[33]通过研究适合于描述异步、并发的行为特点的 Petri 网，将基于 BPEL 描述的 Web 服务组装转化为扩展的着色 Petri 网，提出了一种针对 Web 服务组装交互行为的基于 ECPN 控制流和数据流结合的测试方法。Lee 和 Offutt[34]在 2011 年首次将变异测试应用到

Web 服务测试中。对于 BPEL 程序的变异测试，Boonyakulsrirung 等人[35]提出一种面向 BPEL 的弱变异测试框架，此方法通过牺牲变异得分来提高变异分析的效率。Estero-Botaro 等人[6]运用变异测试技术，使用遗传算法来获得变异体集合的子集，选择出高质量的变异体。赵彦[36]提出了一种基于场景的 BPEL 测试用例生成方法，通过将 BEPL 流程转换为抽象的图模型，然后基于指定的覆盖准则，生成测试场景及测试数据，并运用变异测试技术进行评估，证明了该方法的有效性。

目前也有一些支持 BPEL 程序自动化变异测试的工具。Domínguez-Jiménez 等人[37]开发了一个名为 GAmera 工具。GAmera 支持 BPEL 程序的变异体生成。其通过解析待测 BPEL 程序识别出可应用的变异算子，利用遗传算法对变异算子进行选择，生成相应的变异体。此外，GAmera 还支持测试用例执行和结果的统计。Boonyakulsrirung 和 Suwannasart[38]研发一个名为 WeMuTe 的工具，用来支持 BPEL 程序的弱变异测试过程。

2.2.3 课题组相关工作

课题组前期对 BPEL 程序的变异测试进行了一系列研究[7,8]，主要内容包括：(1)提出了一个面向 BPEL 程序的变异测试框架，并开发相应支持工具。工具能够解析 BPEL 程序生成相应的变异体，对 BPEL 程序进行变异测试及结果分析。(2)用经验研究的方法，采用 SupplyChain、SmartShelf、Supply-Customer、LoanApproval、CarEstimate 和 TravelAgency 6 个 BPEL 实例程序，运用边界值分析和等价类划分方法为每个实例程序设计 3 组测试用例进行变异测试。根据结果评估面向 BPEL 变异算子的有效性和变异算子被检测到的相对难易程度，寻找精简变异算子的方案，以提高 BPEL 变异测试效率。然而，课题组前期工作仍然存在一些不足：

(1) 提出的精简变异算子方案不够完善；
(2) 开发的面向 BPEL 程序变异测试工具，不仅部分核心功能存在一定的问题，而且也没有对测试用例生成和变异测试优化部分的支持；
(3) 缺少对新型变异算子的经验研究。

3 面向 BPEL 程序的变异测试优化技术

本章介绍面向 BPEL 程序变异测试优化技术的基本原理以及一般过程，并用实例介绍优化技术的使用。

3.1 基于包含关系的变异测试优化技术 MuSR

3.1.1 方法原理

通过研究 BPEL 程序的变异算子操作规则，本文发现不同变异算子指导产生的变异体可能具有相似或相同的错误。在变异测试中，识别并移除出这些具有相似或相同的错误的变异体，可以降低变异体数量同时不影响测试效果。基于以上想法，本文提出基于包含关系的变异测试优化技术，简称为 MuSR 技术，是一种通过移除具有包含关系的变异算子进行变异测试的优化技术。其中，包含关系定义如下：

假设存在两个变异算子 O_A 和 O_B，对于给定的程序分别生成 $M_A=\{M_1^A, M_2^A, ..., M_j^A\}$，$M_A=\{M_1^A, M_2^A, ..., M_j^A\}$ 变异体集合，若存在规则 $\forall M_l^B \in M_B, \exists M_k^A \in M_A, \forall tc$ 杀死 M_k^A，M_l^B 也一定被 tc 测试用例杀死，则说 O_A 包含 O_B，记为 $O_A \leftarrow O_B$。

满足包含关系规则，说明 M_l^B 与 M_k^A 的错误行为是等价的。那么，O_B 产生的变异体可以被 O_A 产生的变异体所替代和精简。

对 BPEL 程序的 34 种变异算子规则和产生的变异体进行研究，共发现如下 7 组具有包含关系的变异算子组合：*AEL* ←*AIE*、*ASF* ←*ASI*、*ERR* ←*EIN*、*AEL* ←*CFA*、*EIU* ←*EAN*、*CDC* ←*CDE* 和 *CDC* ←*CCO*。接下来，以实例阐述这 7 组变异算子组合的包含关系。

(1) *AEL* 变异算子描述的删除一个活动的操作。*AIE* 变异算子描述的是移除 *if* 活动中的 *else if* 活动或 *else* 活动的操作。显然，*AEL* 变异算子可以移除 BPEL 程序中的任意活动，其中也包括 *if* 活动中的 *else if* 活动或 *else* 活动，所产生的变异体等价于 *AIE* 产生的变异体，如表 3-1 所示。可知，*AIE* 产生的变异体完全被包含在 *AEL* 产生的变异体集合中。

(2) *ASI* 变异算子定义交换 *sequence* 活动内的两个子活动顺序的操作。*ASF* 描述的是用 *flow* 活动替换 *sequence* 活动的操作。*flow* 表示支持并发的活动。因为将 *sequence* 活动替换为 *flow* 活动后，其子活动执行顺序将不确定，此举与 *ASI* 算子行为相同，如表 3-2 所示，所以，*ASF* ←*ASI*。

表 3-1 AIE 与 AEL 变异算子应用实例对比

变异算子	程序片段	操作
原始程序	<if name="If1"> <condition>.....</condition> <sequence>.....</sequence> <else>.....</else> </if>	无
AIE	<if name="If1"> <condition>.....</condition> <sequence>.....</sequence> </if>	删除 if 活动中的 else 活动
AEL	<if name="If1"> <condition>.....</condition> <sequence>.....</sequence> </if>	删除 else 活动

表 3-2 ASI 与 ASF 变异算子应用实例对比

变异算子	程序片段	操作
原始程序	<sequence name="Sequence1"> <invoke .....>.....</invoke> <assign name="Assign">.....</assign> </sequence>	无
ASI	<sequence name="Sequence1"> <assign name="Assign">.....</assign> <invoke .....>..... </invoke> </sequence>	交换 assign 与 invoke 活动
ASF	<flow name="Sequence1"> <invoke .....>..... </invoke> <assign name="Assign">.....</assign> </flow>	替换 sequence 为 flow 活动

(3) *EIN* 变异算子是在逻辑表达式前插入 *not* 语句，就是对逻辑表达式的值取反的操作。*ERR* 变异算子是将关系运算符用另一个相同类型运算符替换的操作。若 *ERR* 将逻辑表达式中的“≥”替换为“<”，这与逻辑表达式取反是同样的错误，如表 3-3 所示。可知，*ERR*←*EIN*。

(4) *CFA* 变异算子是移除流程中的一个活动，用 *exit* 活动替换的操作。*exit* 活动用于立即结束业务程序实例。*AEL* 变异算子定义为移除一个活动的操作。两种变异算子均包括移除任一活动操作，所造成的结果相似，如表 3-4 实例所示。此外，*CFA* 是移除活动并结束业务程序，产生的变异体相比 AEL 的变异体更容易被杀死。

(5) *EAN* 变异算子定义为替换表达式为其绝对值的负值的操作。*EIU* 变异算子是对算数表达式取负的操作。如表 3-5 所示，该实例中这两个变异算子产生的变异体相同。

表 3-3 ERR 与 EIN 算子应用实例对比

变异算子	程序片段	操作
原始程序	<condition> ($warehouseAmessage.WarehouseAResponse >= $input.amount) </condition>	无
EIN	<condition> not(($warehouseAmessage.WarehouseAResponse>=$input.amount)) </condition>	条件表达式前加 NOT
ERR	<condition> ($warehouseAmessage.WarehouseAResponse < $input.amount) </condition>	替换>= 为<

表 3-4 CFA 与 AEL 变异算子应用实例对比

变异算子	程序片段	操作
原始程序	<sequence name="Sequence1"> <invoke……> .....</invoke> <assign name="Assign">……</assign> </sequence>	无
CFA	<sequence name="Sequence1"> <exit ……> .....</exit> <assign name="Assign">……</assign> </sequence>	用 exit 活动替换 invoke 活动
AEL	<sequence name="Sequence1"> <assign name="Assign">……</assign> </sequence>	移除 invoke 活动

表 3-5 EAN 与 EIU 算子应用实例对比

变异算子	程序片段	操作
原始程序	<transitionCondition> ($decryptResponse.creditInformationMessage/ amount <= 1000.0) </transitionCondition>	无
EAN	<transitionCondition> ($decryptResponse.creditInformationMessage/ amount <= -(1000.0 * ((1000.0 >= 0.0) - (1000.0 < 0.0)))) </transitionCondition>	替换 1000 为其绝对值并取负
EIU	<transitionCondition> ($decryptResponse.creditInformationMessage/ amount <= -1000.0) </transitionCondition>	用-1000 替换 1000

(6) *CDC* 变异算子是应用判定/条件覆盖准则到程序中的操作。判定/条件覆盖准则指判定中的每个条件都取到各种可能的值，并使每个分支也要取到各种可能的取值。*CDE* 变异算子是应用判定覆盖准则到程序中的操作。判定覆盖指的是每个分支至少执行一次。在表 3-6 给出的实例中，这两

种变异算子产生的变异体相同。此外，*CDC* 变异算子所能产生的变异体相比 *CDE* 的变异体更加全面。可知，*CDC*←*CDE*。

(7) *CDC* 变异算子是应用判定/条件覆盖准则到程序中的操作。*CCO* 变异算子是应用条件覆盖准则到程序中的操作。条件覆盖准则是指使程序里每个判断中每个条件的可能取值至少执行一次。在表 3-7 给出的实例中，这两种变异算子产生的变异体具有相同错误行为；并且 *CDC* 变异算子所能产生的变异体相比 *CCO* 的变异体更加全面，可知，*CDC*←*CCO*。

表 3-6 CDE 与 CDC 变异算子应用实例对比

变异算子	程序片段	操作
原始程序	<condition> $quantity.CheckQuantity< $commodity.amount </condition>	无
CDE	<condition>true()</condition>	将条件表达式取真
CDC	<condition>true()</condition>	将条件表达式取真

表 3-7 CCO 与 CDC 变异算子应用实例对比

变异算子	程序片段	操作
原始程序	<condition> $quantity.CheckQuantity< $commodity.amount </condition>	无
CCO	<condition>true()</condition>	将条件表达式取真
CDC	<condition>true()</condition>	将条件表达式取真

3.1.2 一般过程

MuSR 技术的一般过程是：首先，运用包含关系规则，识别出具有包含关系的变异算子；然后，从变异算子集合中删除被包含的变异算子，获得精简的变异算子子集；最后，使用变异算子子集指导产生变异体，进行变异测试。

3.2 基于二阶变异体的变异测试优化技术 MuSOM

3.2.1 方法原理

对于变异测试开销大的问题，存在一种高阶变异体优化技术。该技术是由高阶变异体替代一阶变异体进行测试的优化技术。其中，一阶变异体指对

原始程序应用一个变异算子且植入一处错误而生成的变异体；高阶变异体指对原始程序植入 M（$M \geqslant 2$）处错误而生成的变异体。一个高阶变异体可以看成由多个一阶变异体组成。高阶变异体的优化技术的提出基于两个推测[11]：1）执行一次 M 阶变异体等同于执行 M 个一阶变异体；2）高阶变异体比一阶变异体产生等价变异体的概率小。基于这两个推测，可以看出高阶变异体优化技术主要是通过减少待执行变异体数目和等价变异体识别开销来降低测试开销。

指导生成高阶变异体的方法有很多，这里讨论一种基于 *LastToFirst* 算法[39]来生成二阶变异体的方法。*LastToFirst* 算法描述按照一定顺序组合两个一阶变异体，产生二阶变异体的过程，其原理如图 3-1 所示。其中，P 指待测程序，M_1 至 M_n 代表原始程序的一阶变异体，M_{kk+1} 代表通过组合两个一阶变异体而生成的二阶变异体。本文将 *LastToFirst* 算法应用到面向 BPEL 程序的变异测试的中，提出面向 BPEL 程序的二阶变异体优化技术，简称为 MuSOM。

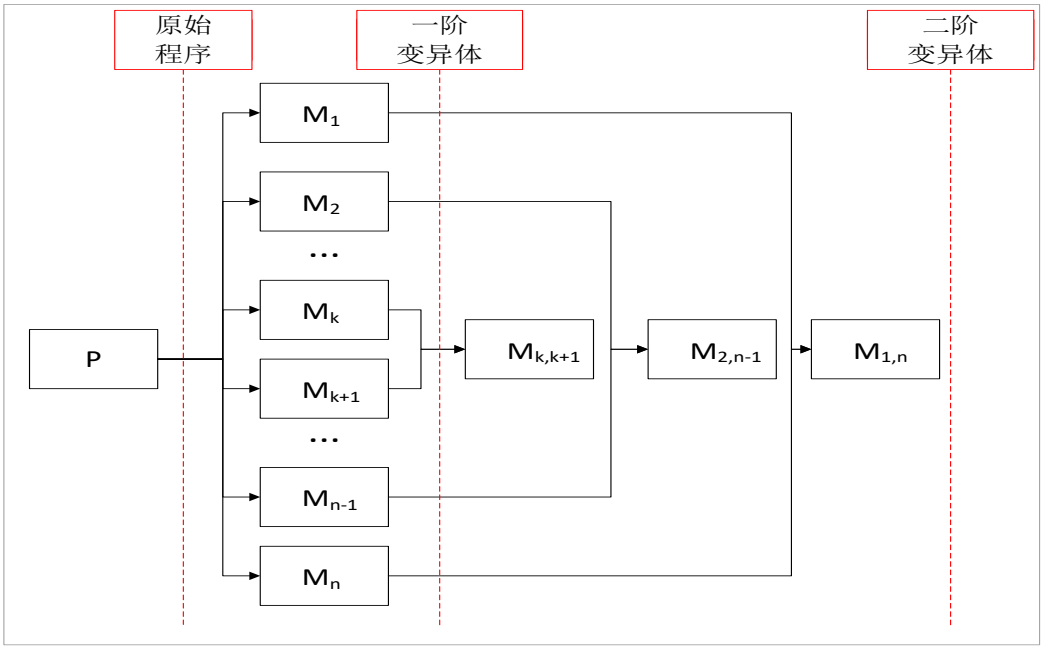

图 3-1 *LastToFirst* 算法生成二阶变异体过程

3.2.2 一般过程

MuSOM 描述的二阶变异体生成步骤如下：

(1) 为 BPEL 程序 P 生成一阶变异体；按照一阶变异体产生的顺序，给变异体命名并编号为 *1* 至 n，n 为一阶变异体的总数；得到的变异体的集合表示为 $M_{1o}=\{M_1,M_2,...,M_k,M_{k+1},...,M_{n-1},M_n\}$。

(2) 将 M_{1o} 中的一阶变异体组合生成二阶变异体，组合方法为：将编号为 *1* 与编号为 *n* 的两个一阶变异体组合，生成二阶变异体，记为 $M_{1,n}$；将编号为 *2* 与编号为 *n-1* 的两个一阶变异体组合，生成二阶变异体，记为 $M_{2,n-1}$；以此类推，使得每个一阶变异体都用来生成二阶变异体。

(3) 当 *n* 为奇数时，除了一阶变异体 $M_{(n+1)/2}$，其余的一阶变异体都两两组合产生了二阶变异体。这种情况下，规定将 $M_{(n+1)/2}$ 与 $M_{(n+1)/2+1}$ 变异体进行组合，来生成二阶变异体。

(4) 最终，得到二阶变异体集合为 $M_{2o}=\{M_{1,n},M_{2,n-1},...,M_{k,k+1}\}$。

3.2.3 方法示例

为了方便理解 MuSOM 技术，使用 SupplyChain[10]实例进行演示。SupplyChain是一个BPEL实例，同时也在第5部分用于实例验证。SupplyChain实例共生成 34 个一阶变异体。采取 MuSOM 方法，为该实例生成二阶变异体。图 3-2 展示一个二阶变异体生成过程。其中，*CDE_1* 代表生成的第一个一阶变异体的部分程序代码。该变异体使用了 *CDE* 变异算子，应用判断准则到程序中，将条件表达式用 *true()*替换。图中对变异的位置用加粗放大字体进行标识。*ASI_4* 代表生成的最后一个一阶变异体部分程序代码。该变异体应用 *ASI* 变异算子，交换了 *reply* 活动与 *if* 活动的位置。通过读取这两个变异体所植入的错误，同时应用到原始程序 *P* 中，得到一个二阶变异体，命名为 *CDE1_ASI4*。

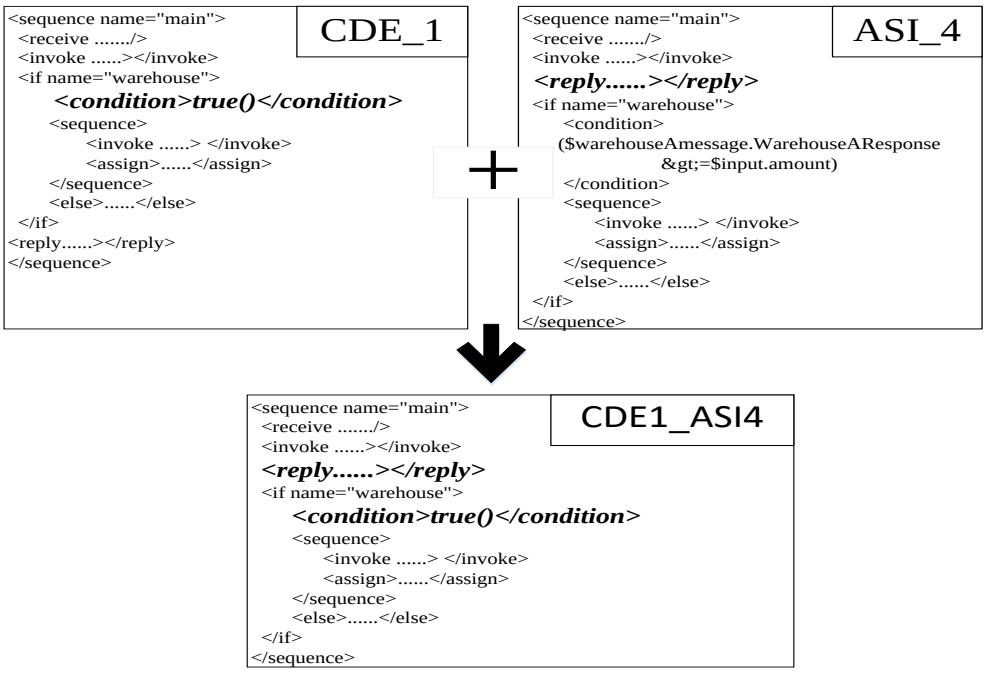

图 3-2 SupplyChain 的二阶变异体生成示例

3.3 基于变异算子优先级的变异测试优化技术 MuPri

3.3.1 方法原理

(1) 基本原则

在变异测试中，有的变异体可以被绝大多数测试用例“杀死”，而有的变异体只能被特殊的测试用例检测出来。若一些较难被测杀的变异体都能被测试用例“杀死”，那么理论上该测试用例也能“杀死”那些容易被测杀的变异体。基于以上内容，我们从变异体执行顺序的角度来考虑减少测试开销，优先使用这些可以产生较难被测杀变异体的变异算子，理论上可以生成更难被测杀的变异体集合，这些变异体可以更好的衡量测试用例有效性和完备性。

(2) 相关规则定义

为了衡量变异算子产生变异体被测杀的难易程度，我们引入一个变异算子质量度量指标。在提出该指标之前，约定 *P* 指待测的 BPEL 程序；*TS* 为有效测试用例集，$TS=\{t_1,t_2,\ldots,t_i,\ldots,t_n\}$，其中 t_i 为测试用例集中第 *i* 个测试用例，*n* 为测试用例集的用例总数；*MO* 表示 BPEL 语言的变异算子集合，$MO=\{O_1,O_2,\ldots O_i,\ldots,O_{34}\}$，其中 O_i 表示变异算子列表中第 *i* 个变异算子。

变异算子质量的定义从故障检测率推导而来，**故障检测率**（Fault Detection Rate，FDR）[4]定义为测试用例“杀死”变异体的比例，广泛用来衡量用例的故障检测能力。FDR 计算公式为：

$$FDR(M_j^i,TS)=\frac{NC(M_j^i,TS)}{|TS|} \qquad (3\text{-}1)$$

其中，M_j^i 表示 O_i 变异算子的第 *j* 个变异体，$NC(M_j^i, TS)$表示杀死变异体 M_j^i 的测试用例数目。FDR 值越大，说明测试用例集 *TS* 杀死该变异体的几率越大，该变异体越容易被杀死。从而，将**变异算子质量**定义为：

$$Q_o=1-\frac{\sum_{j=1}^{NO_i}FDR(M_j^i,TS)}{NO_i} \qquad (3\text{-}2)$$

其中，NO_i 表示 O_i 变异算子产生的非等价变异体总数。通过 FDR 的定义可知，Q_o 的值越大，表明越少的测试用例将该类变异算子产生的变异体杀死，反映出该类变异体的质量较好。

3.3.2 一般过程

本文提出一种基于变异算子优先级的变异测试优化技术，简称为 MuPri。

MuPri 是通过衡量变异算子质量，为变异算子分配测试优先级的一种优化技术。MuPri 技术基本过程为：首先对 BPEL 程序进行变异测试；然后根据测试结果，计算变异算子的质量，按照质量由高到底的顺序为其排序，质量好的变异算子在变异测试中分配较高的优先级，质量差的变异算子分配较低的优先级；按照变异算子优先级顺序指导生成变异体集合，这些变异体则是按照被测试用例“杀死”由难到易的顺序生成。通过这个顺序来选择测试用例，可以优先选出能将较难“杀死”的变异体检测出的测试用例，从而可以达到测试用例排序的效果，并可以更快捷的衡量测试用例的有效性和完备性。所以，MuPri 技术不仅可以给测试用例集合排序，得到检错效率更高测试用例集，而且可以依据变异算子的优先级，优先使用质量好的变异算子生成变异体，增强变异体集质量并减少测试开销。其中，MuPri 技术为测试用例排序的一般过程描述如下：

(1) 选取实例程序 *P*，将其可应用的变异算子加入到集合 *O* 中；

(2) 根据变异算子的优先级排名，按照排名从小到大的顺序，依次取出变异算子加入到 *OP* 队列中，得到 $OP=[OP_1,\cdots,OP_Z,\cdots,OP_n]$；

(3) 令 z=1；

(4) 取出 OP_Z 变异算子，生成其一阶变异体集合 $MO_Z(P)$；

(5) 使用测试用例 *TS* 执行 $MO_Z(P)$中的变异体，其中，*TS* 表示具有顺序的用例队列。根据执行结果，从 $MO_Z(P)$中删除被 *TS*“杀死”的变异体。判断 $MO_Z(P)$是否为空，如果是则执行第七步；否则执行第六步；

(6) 添加一个测试用例到 *TS* 中并返回上一步；

(7) 判断 z 是否等于 n，如果是，则输出测试用例 *TS*；否则令 $z=z+1$，返回第四步。

3.3.3 方法示例

采用 SupplyChain 实例演示 MuPri 技术为测试用例排序过程。首先对该实例进行变异测试，统计变异算子质量，得到变异算子顺序为“*CDE->CCO->CDC->ERR->AIE->ASF->AEL->CFA->EIN->ASI->ACI*”。按照 3.3.2 节提出的一般过程，对 SupplyChain 实例的测试用例进行排序，结果如表 3-8 所示。表中的对勾表示该测试用例是测试用例集 *TS* 中，第一个将该变异体“杀死”的用例。因此，这个实例根据 MuPri 技术得到测试用例集的顺序是“*T1->T2->T3*”。

表 3-8 SupplyChain 变异体执行测试用例 *TS* 结果

Rank	Mutant		Test Case		
			T1	T2	T3
1	CDE	1	√		
		2		√	
2	CCO	1	√		
		2		√	
3	CDC	1	√		
		2		√	
4	ERR	1	√		
		2			√
		3	√		
		4		√	
		5	√		
5	AIE	1		√	
6	ASF	1	√		
		2	√		
7	AEL	1	√		
		2	√		
		3	√		
		4	√		
		5	√		
		6		√	
		7	√		
8	CFA	1	√		
		2	√		
		3	√		
		4	√		
		5	√		
		6		√	
		7	√		
9	EIN	1	√		
10	ASI	1	√		
		2	√		
		3	√		
		4	√		
11	ACI	1	√		

3.4 小结

本章结合 BPEL 程序基本特点和变异测试基本原理，从变异算子之间的包含关系、变异算子的优先级和二阶变异技术三个方面进行研究，提出了三种变异测试优化技术；阐述了优化技术的基本原理，一般过程，并利用 SupplyChian 程序实例来演示了 MuSOM 和 MuPri 技术的具体过程。

4 面向 BPEL 程序的变异测试集成化工具设计与实现

本章介绍面向 BPEL 程序的变异测试集成化支持工具的设计与实现，包括需求分析、工具设计与实现，最后用一个实例验证并演示工具。

4.1 需求分析

为了提高面向 BPEL 程序的变异测试自动化程度和便于使用本文提出的面向 BPEL 程序的变异测试优化技术，在扩展课题组前期研发的相关工具基础上，本文开发了一个面向 BPEL 程序的变异测试的集成化支持工具 μBPEL（A Mutant Testing System for BPEL Programs）。μBPEL 不仅支持变异体的生成、测试用例的执行和测试结果验证，同时支持变异体的精简和测试用例的生成。采用 UML 用例图对 μBPEL 进行需求分析，如图 4-1 所示。各用例描述如下：

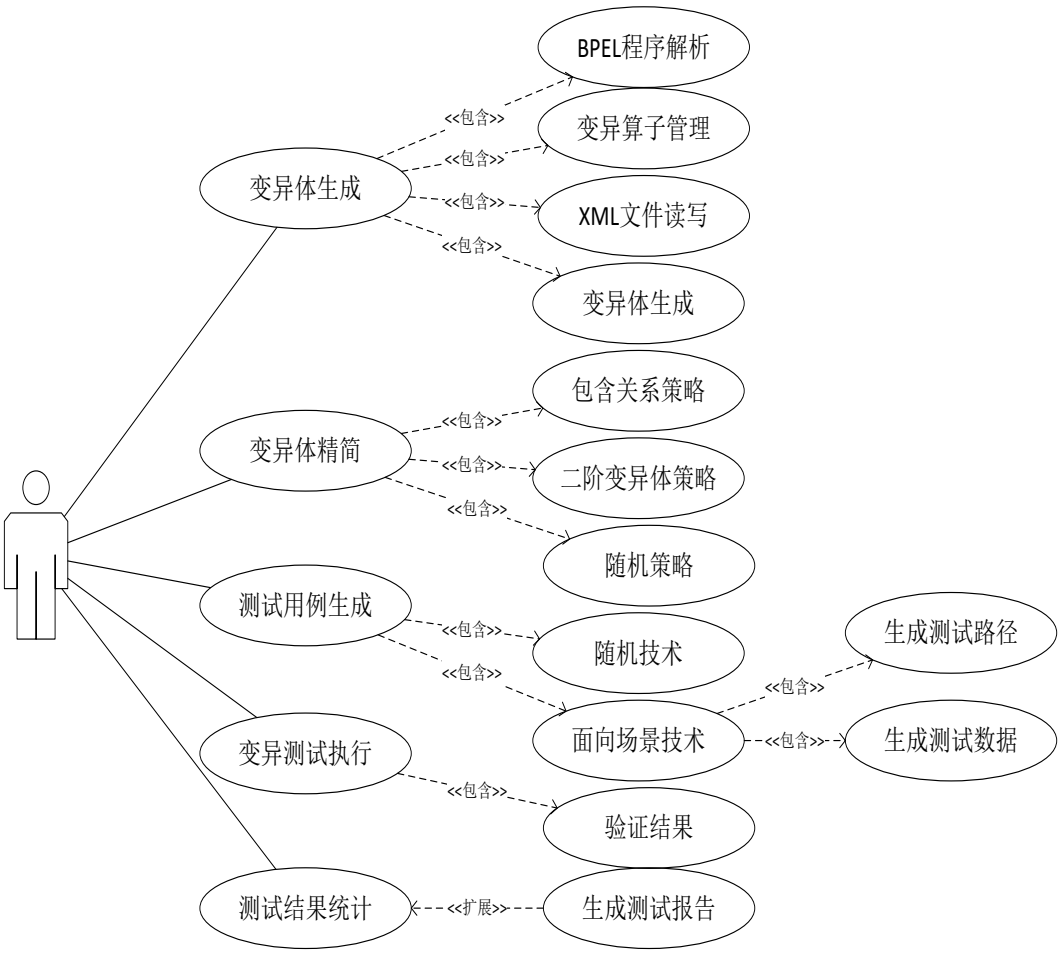

图 4-1 μBPEL 工具用例图

(1) **变异体生成：**用户提供待测的 BPEL 程序，系统对程序进行解析，找到可以变异的元素，应用变异算子生成变异体。其包括如下四个子用例：

- **BPEL 程序解析：**该用例用于对 BPEL 程序进行解析，获得程序的元素、节点等信息。
- **变异算子管理：**该用例定义和存储 34 种变异算子的识别规则和处理操作，用于识别出 BPEL 程序可以应用的变异算子和变异操作。
- **XML 文件读写：**该用例定义 XML 文件读写操作，用来读入 BPEL 程序和植入错误。
- **变异体生成：**该用例根据用户需求，应用相应变异算子，完成变异体的生成。

(2) 变异体精简：该用例提供三种变异测试优化技术：包含关系策略（MuSR）、二阶变异体策略（MuSOM）和随机策略。

- **包含关系策略：**该用例通过解析 BPEL 程序，识别出可以应用的变异算子后，根据 MuSR 优化技术，对这些变异算子进行约简，移除具有包含关系的变异算子；对约简的变异算子集合生成变异体，从而获得优化后的变异体集合。
- **二阶变异体策略：**该用例通过解析 BPEL 程序，由 MuSOM 技术指导生成二阶变异体，获得二阶变异体集合。
- **随机策略：**支持广泛使用的随机变异体选择优化方法。根据用户输入需要精简比例，从所有的变异体中随机选择出相应数量的变异体，获得精简的变异体集合。

(3) 测试用例生成：根据用户的不同需求，生成测试用例。该用例提供两种用例生成技术，一种是广泛使用的随机技术，另一种是课题组提出的面向场景的用例生成技术[36]。

- **随机技术：**根据程序输入变量和约束条件，随机生成满足条件的测试数据。
- **面向场景技术：**通过解析和转换 BPEL 程序格式，基于一定覆盖准则，获得测试路径并针对每条测试路径生成对应的测试数据。

(4) 变异测试执行：用户先将待测的变异体部署到 BPEL 环境中；然后， 在 BPEL 原始程序和变异体上运行测试用例；比对二者的输出结果，记录变异体的测试状态和故障检测率。

(5) **测试结果统计：**该用例通过对变异测试结果进行统计，给出变异体被测杀情况，变异得分等信息，并生成测试报告。

4.2 μBPEL 设计与实现

讨论 μBPEL 的架构以及各组件的设计与实现。

4.2.1 系统架构

图 4-2 描述了 μBPEL 工具的系统架构。选用橙色矩形框表示工具的基本组件，包括变异体生成组件，变异体优化组件，测试用例生成组件，变异测试执行组件和结果评估共五个组件。每个组件由多个模块构成，各个组件的功能设计描述如下：

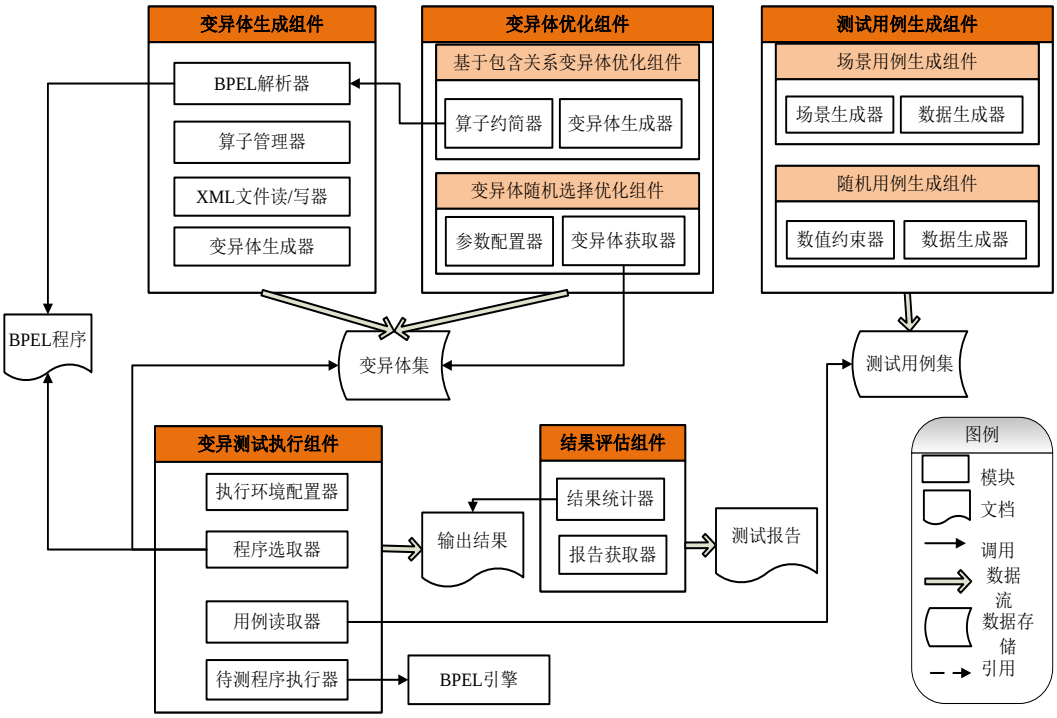

图 4-2 μBPEL 工具系统结构图

(1) 变异体生成组件：负责对待测 BPEL 程序生成相应的一阶或二阶变异体，包括四个子模块。

- **BPEL 解析器：**该模块根据用户输入的 BPEL 程序文件路径，读入待测的 BPEL 程序文件，并进行解析。
- **算子管理器：**定义 34 种变异算子的匹配与操作处理方法。该模块中的每个变异算子均定义两个相关类，一个负责匹配 BPEL 程序，获取可以变异的节点，一个负责对相应节点进行变异处理，并将变异体写出。
- **XML 文件读/写器：**负责 BPEL 程序文件的读入和错误植入后变异体

的输出。通过 DOM4J[40]技术读入 BPEL 文件，将其解析为 DOM 树形结构的 Document 对象，对其中的节点或元素进行查找和修改，完成变异体文件的输出。

- **变异体生成器：**根据记录的可变异位置信息，调用相应变异算子处理方法，生成一阶或二阶变异体，并将变异体文件写出。

(2) 变异体优化组件：支持对 BPEL 程序的变异测试进行优化，这里包括两种变异测试优化技术，一种是 MuSR 技术，一种是 Random 技术，由两个组件组成。

1） **基于包含关系变异体优化组件**

- **算子约简器：**该模块通过 BPEL 程序解析模块，解析获取待测 BPEL 实例可以应用的变异算子列表；通过读取该列表，识别出列表中具有包含关系的变异算子，将被包含的变异算子从该列表中剔除，获得优化后的变异算子子集。
- **变异体生成器：**该模块通过调取变异算子子集中的相应变异算子的操作，完成该类错误植入到程序中生成变异体过程。

2） **变异体随机选择优化组件**

- **参数配置器：**该模块根据用户输入的待优化的变异体集合路径和变异体精简比例，解析并获取该路径下的所有变异体。
- **变异体获取器：**该模块根据参数配置器输入的相应信息，以变异体精简比例，从输入的变异体集合中随机获取相应数目变异体，并将这些变异体保存在指定目录下。

(3) 测试用例生成组件：该模块主要是将课题组前期研发的 TSTG 工具[36]测试用例生成部分功能进行了集成。输入是 BPEL 原始文件，输出期望的测试用例。其中，场景用例生成组件负责解析 BPEL 程序，转换为图模型，基于一定覆盖准则完成测试场景和数据的生成；随机用例生成组件负责解析 BPEL 程序的 WSDL 文档，根据用户输入约束的条件，随机的生成满足条件测试用例。

(4) 变异测试执行组件：该组件负责执行测试用例并获取输出结果，由如下四个模块组成：

- **执行环境配置器：**该模块通过读取 BPEL 的配置信息，获取 BPEL 服务的端口号和操作名称配置执行的环境，执行部署、反部署服务等操作。
- **程序选取器：**通过用户输入的文件路径，依次获取原始程序和变异体

程序文件。

- **用例读取器：**通过用户输入的待执行测试用例集的路径，依次读入并解析相应的测试用例文件，获取用例输入变量的类型、数目、值及用例个数等信息。
- **待测程序执行器：**调用 BPEL 引擎依次对原始程序和变异体运行输入的测试用例集并将输出结果保存在文件中。运用线程的思想，为每个程序执行测试用例的过程分配一个线程，依次批量的完成所有程序的测试过程。

(5) 测试结果评估组件：该模块负责输出结果进行统计分析，由结果统计器和报告获取器两个模块组成。

- **结果统计器：**读入原始程序与变异体程序测试结果的文件，通过逐一的对执行相同测试用例后，原始程序和变异体的输出结果进行对比，若二者结果不同，表明该测试用例将变异体“杀死”，标记为“F”；否则，记为“T”。依次记录变异体被测杀的状态，并统计出针对变异体的每个测试用例集合的的故障检测率信息。
- **报告获取器：**根据结果统计器的输出结果，计算变异得分并生成报告，包括变异体数目，被杀死变异体数目，变异得分信息。

4.2.2 工具实现

下面讨论支持工具μBPEL 的实现。μBPEL 核心功能的类图如图 4-3 所示，分为 5 大模块：

(1) BPEL 程序解析 Parser 类

该类主要对输入的 BPEL 程序使用 DOM4J 技术进行解析，找到可变异位置等信息。

- **Parser 类：**用于解析 BPEL 文件的 document 对象，获取可变异位置信息列表。
- **ManageDataInfo 类：**用于存储可变异位置等基本信息，包括变异算子名称、变异节点等。

(2) 变异体生成 MutantProduction 类

该类用于根据 BPEL 程序的解析获得的可变异位置和变异算子列表，生成一阶或二阶变异体，方法如下：

- **GenerateFacade 类：**外观类，提供调用一阶和二阶变异体生成子系统接口，减少客户端和变异体生成子系统之间的耦合。

- **FirstGen 类：**完成相应变异算子的一阶变异体生成，并记录变异体变异位置，使用的变异算子等信息。
- **SecondGen 类：**完成相应变异算子的二阶变异体生成，并记录产生二阶变异体的基本信息。

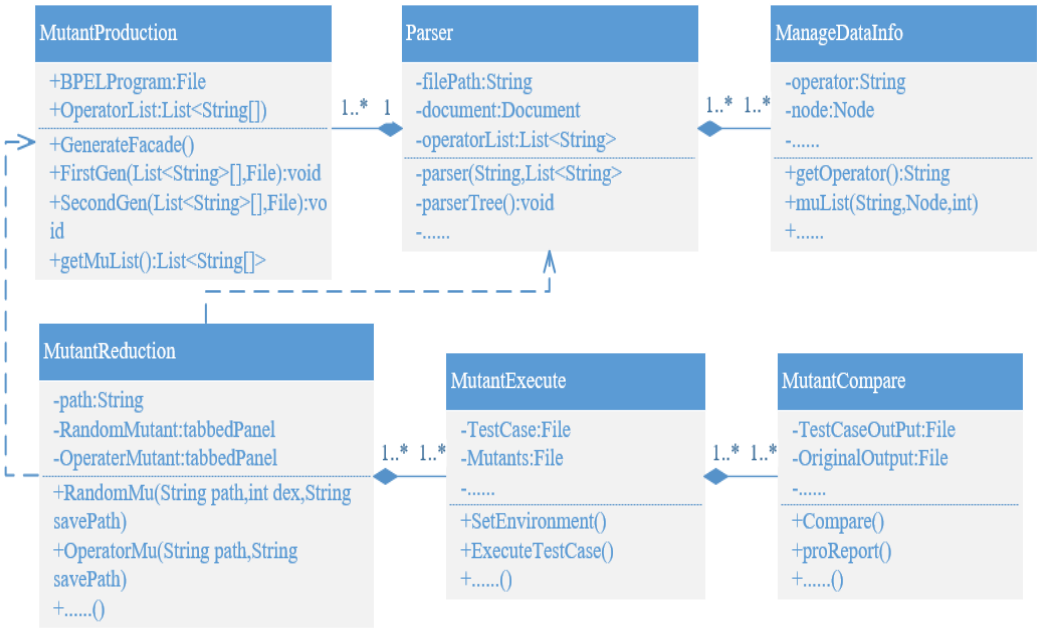

图 4-3 工具类图

(3) 变异体精简 MutationReduction 类

该类用于实现变异测试优化技术的业务逻辑，方法如下：

- **RandomMu()：**通过读入变异体程序所在路径信息及需要精简的比例参数，使用 Random 函数完成变异体的随机选择，得到变异体子集。最后，将选取的精简变异体子集存放在指定的文件夹中。
- **OperatorMu()：**该函数用于根据 BPEL 程序解析结果，获得的可以应用变异算子子集；然后，识别出具有包含关系的变异算子的剔除操作，获得精简的变异算子子集，保存在 List 中。调用变异体生成类，存入变异算子 List，完成相应一阶变异体的生成。

(4) 变异体执行 MutantExecute 类

根据某优化技术选取的变异体子集设计测试用例，并将测试用例在所有变异体上执行，方法如下：

- **SetEnvironment()：**该方法用于配置 BPEL 执行环境，通过解析基于 XML 文件的配置信息，获取 BPEL 服务的端口号 endpoint 和操作名称 operation name 等操作。

- **ExecuteTestCase()：**该方法用于读取 txt 格式的测试用例文件，对第一行的用例信息，进行解析和识别，提取出输入参数个数和类别，并将用例保存在 list 数据结构中，依次向 BPEL 程序发送消息，自动执行变异体，将输出结果保存到以变异体名字加上“_result”后缀命名的文件中。

(5) 结果统计 MutantCompare 类

对输入的原始程序和变异体测试结果进行统计分析，方法如下：

- **Compare():** 该方法通过读入原始程序和变异体测试结果文件，对变异体程序的每条测试用例结果提取和解析，与原始程序的输出结果进行对比，输出变异体被测杀情况。
- **proReport()：**该方法用于根据结果统计情况，输出此次变异测试的对象名称，变异体数量，变异得分等信息。

4.3 系统演示

μBPEL 工具界面如图 4-4 所示，主要由四个部分组成：菜单栏、文件选择区域、具体功能区域和日志区域。其中，菜单部分提供工具的重启、Tomcat 的运行、帮助三个菜单项；文件选择区域用来显示待测 BPEL 程序信息；具体功能区域是工具的主要操作点，包括变异体生成、变异体优化、测试用例生成、测试用例执行及结果分析五个部分；日志区域提供三种日志的显示：[GenerationLog]显示变异体生成时的日志；[ServerLog]显示 Tomcat 运行时输出日志；[LogInfo]显示与用户操作行为相关的日志。

采用 SupplyCustomer[2]实例来演示 μBPEL 工具的使用，包括变异体生成，变异体优化、测试用例执行及结果分析部分。SupplyCustomer 是一个项目订单管理的 BPEL 实例，涉及 6 个 Web 服务，其执行流程如图 4-5 所示。SupplyCustomer 接收客户的两个输入参数，分别是订单的货物信息和地址，系统验证之后向客户反馈信息。

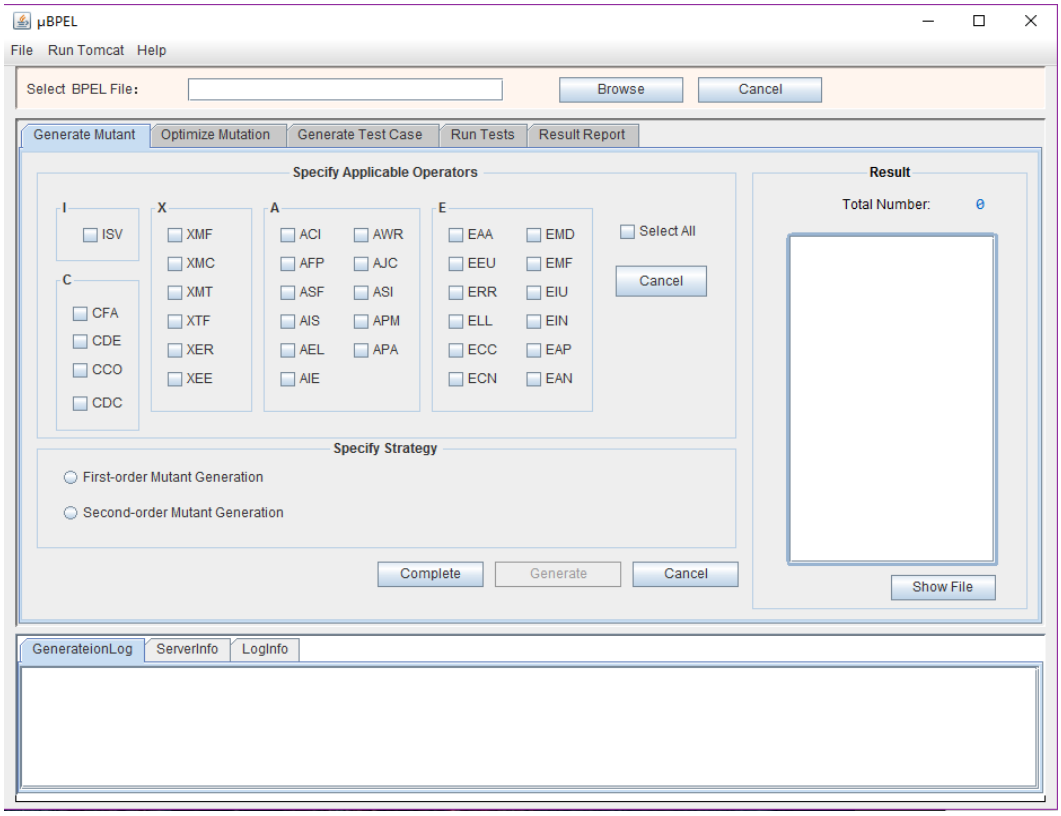

图 4-4 工具主界面

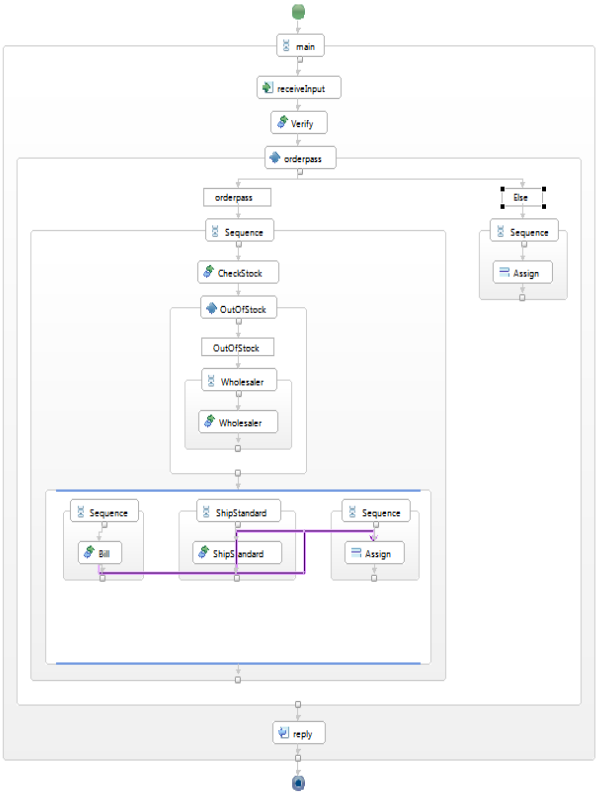

图 4-5 SupplyCustomer 实例执行流程

(1) 变异体生成过程演示

在 BPEL 程序的变异测试前，需要选入待测程序并开启 Tomcat。开启 Tomcat 是用来支持 BPEL 程序的运行。用户可以在 Configuration.xml 配置文件中自定义 Tomcat 的文件路径。当 Tomcat 成功开启后，会在界面上显示“Open Tomcat Successfully”提示语。完成以上步骤，可以进行变异测试。

首先选择[Generate Mutant]标签，进入变异体生成界面。在[Specify Applicable Operators]框中选择需要应用的变异算子（或直接点击[Select All]选择所有变异算子）。其中，鼠标指向变异算子时，界面给出相应的变异算子的转换规则描述。通过[Specify Strategy]框，可以对变异体的种类进行选择，即生成一阶变异体还是二阶变异体。若选择[First-order Mutant Generation]按钮，并完成变异算子和策略的选择后，点击[Complete]按钮，系统会将所配置的选项加载进去。点击[Generate]按钮，系统会完成相应变异算子的一阶变异体生成。变异体生成结果显示在[Result]框中。如图 4-6 所示，SupplyCustomer 共生成 73 个一阶变异体，变异体由所应用的变异算子名称加“_”和数字命名。若在[Specify Strategy]框中，选择[Second-Order Mutant Generation]按钮，其他步骤同一阶变异体生成过程，系统将会生成二阶变异体。图 4-7 展示该实例的二阶变异体生成情况。

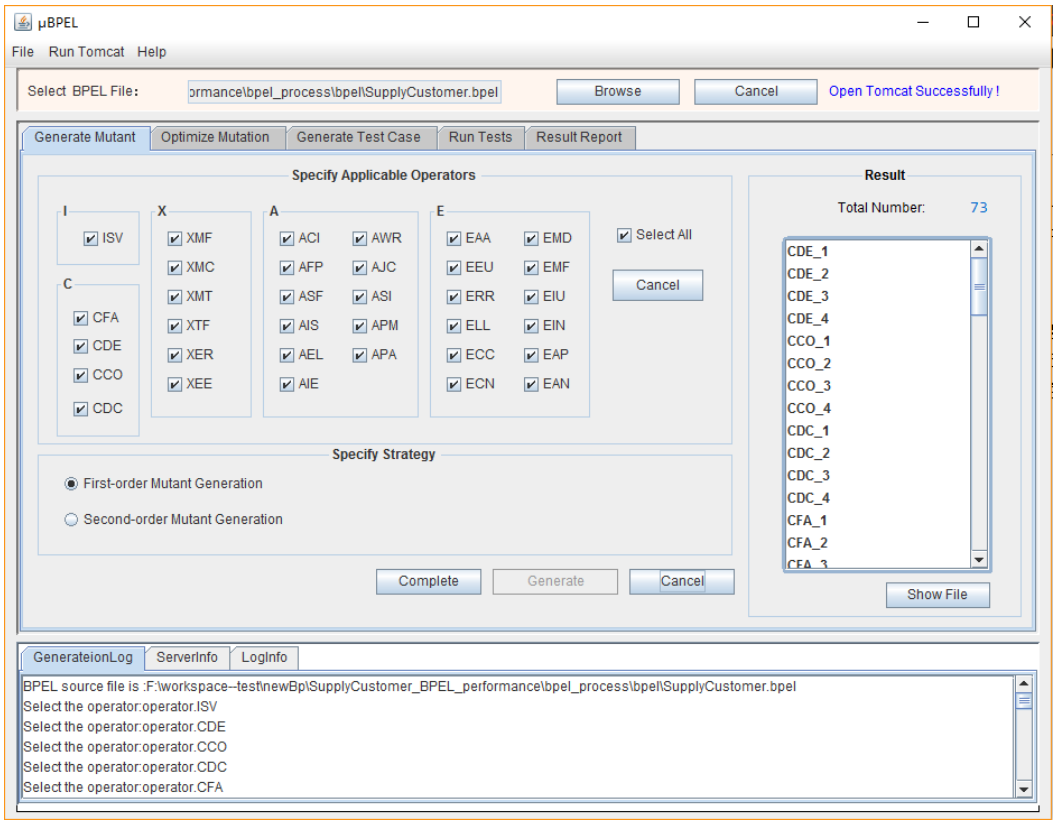

图 4-6 一阶变异体生成界面

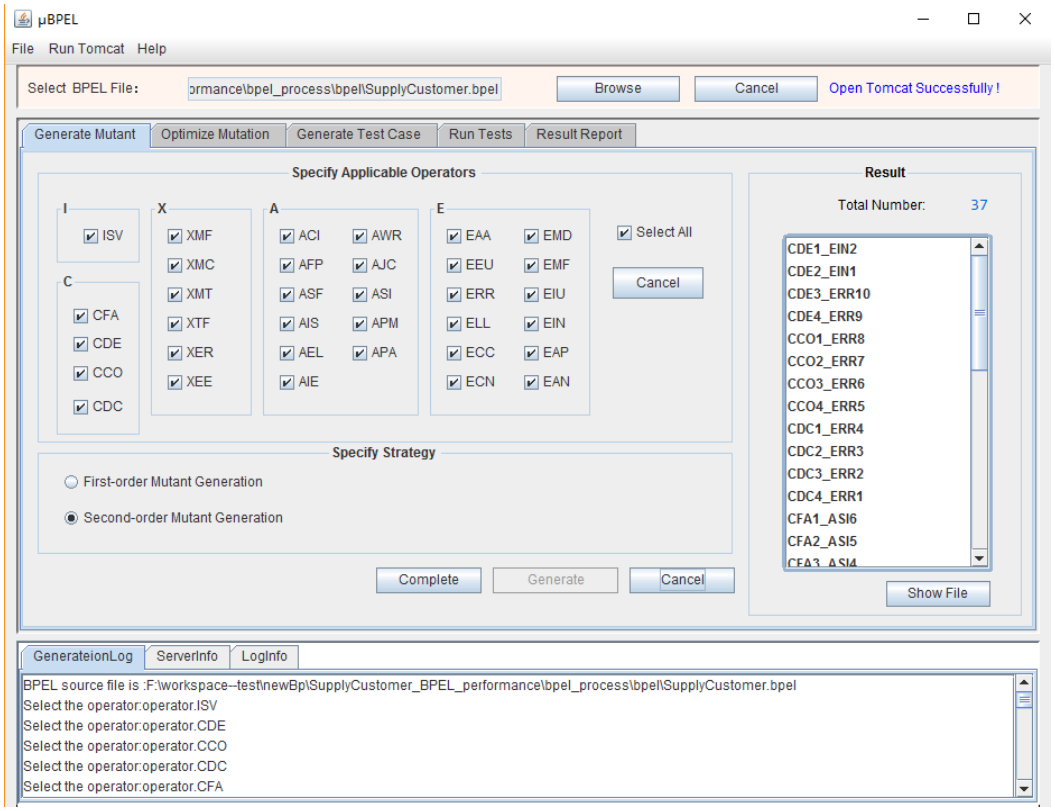

图 4-7 二阶变异体生成界面

为了方便查看变异体与原始程序的不同之处，系统提供一个[Mutants Display]界面对此功能进行支持。在界面中点击所要查看的变异体程序，界面底部的框中会给出该变异体变异的位置和内容，如图 4-8 所示。

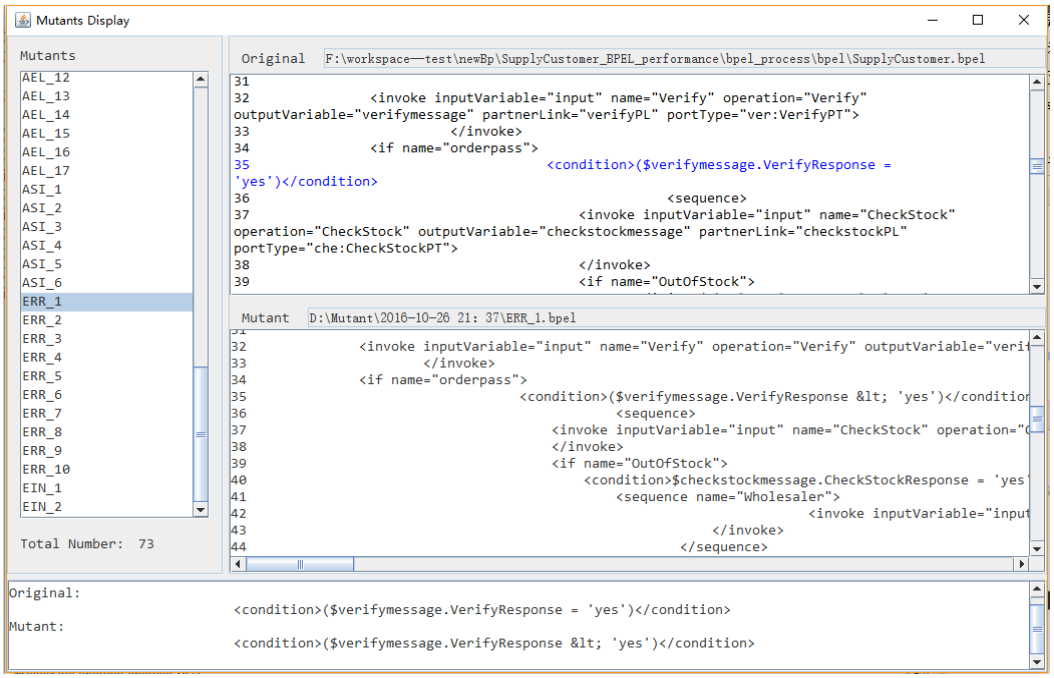

图 4-8 文件对比界面

(2) 变异体优化演示

通过点击[Optimize Mutation]菜单项，进入变异体优化界面。该模块提供两种变异体优化技术：一种是变异体随机选择优化技术，一种是基于包含关系的变异测试优化技术（MuSR）。具体如下：

1）变异体随机选择优化技术演示

点开[R-M]标签页，进入变异体随机选择优化界面。首先用户输入待优化的变异体所在文件夹的路径，系统会解析出该变异体集合的数目和所包含的变异体，结果[Result]中显示。然后，根据优化目标，输入约简比例，如图 4-9 所示。最后，点击[Generate]按钮，完成随机选择优化过程。本实例中，输入实例的一阶变异体所在文件夹路径和 10%的选取比例；μBPEL 工具从变异体集合中，随机的抽取 10%的变异体，得到精简的变异体集合，相关信息显示在[Result]中，如图 4-10 所示。

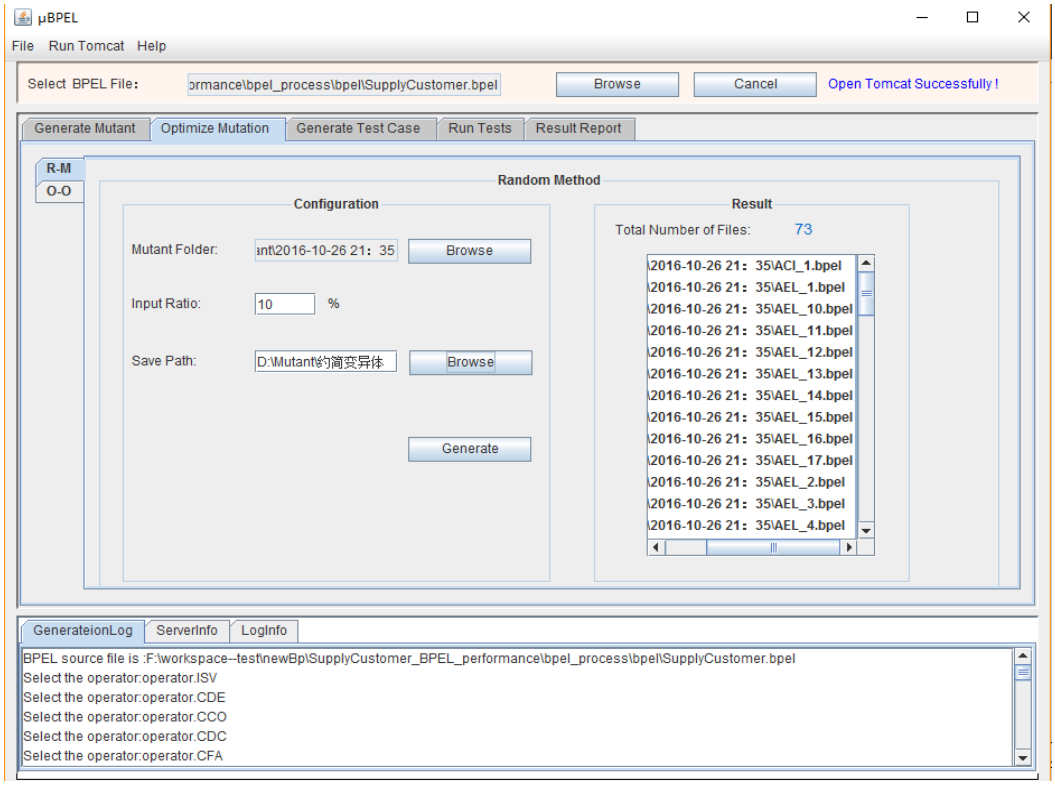

图 4-9 随机优化方法配置

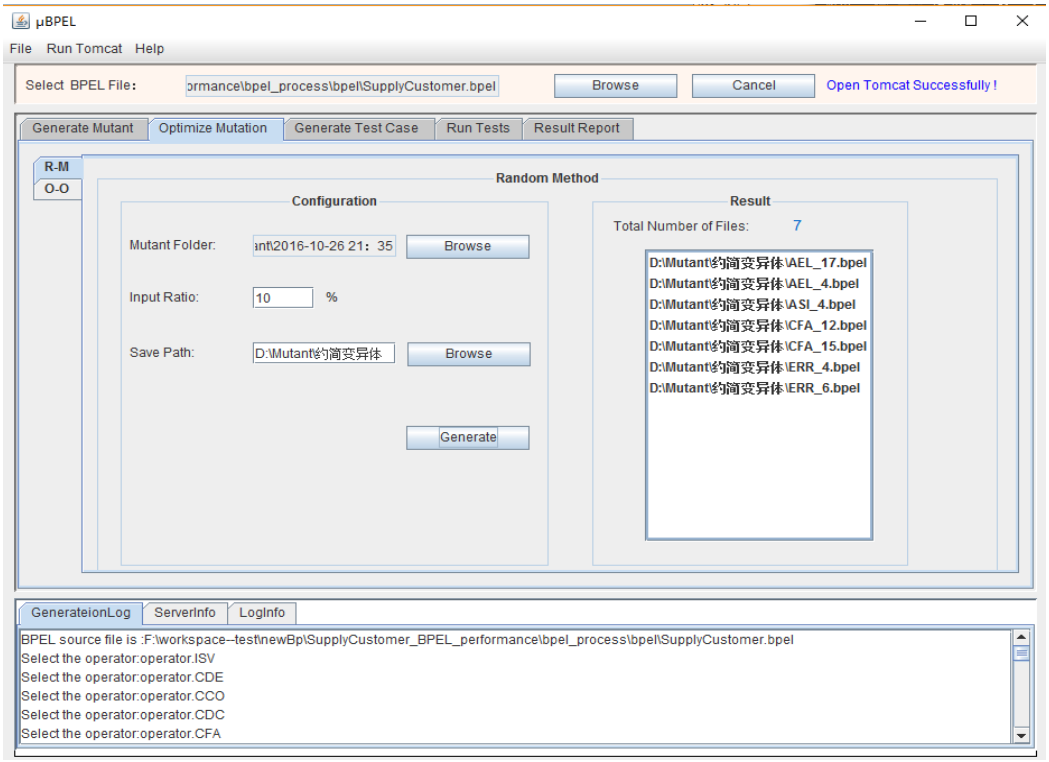

图 4-10 随机抽取变异体执行过程

2) 基于包含关系的变异测试优化技术演示

点开[O-O]标签页，进入界面。首先点击[Parse]按钮，μBPEL 会自动识别出程序可以应用的变异算子。如图所示 4-11 所示，SupplyCustomer 实例可以应用 11 种变异算子。点击[Optimize]按钮，基于 MuSR 技术，系统自动识别出这些被包含的变异算子，并将被包含的变异算子从变异算子集合中剔除（这里表现为取消选中）。在本实例中，*ASI*、*EIN*、*AIE*、*CFA*、*CCO*、*CDE* 这 6 个变异算子被踢除。点击[Generate]按钮，系统会对剩下的 5 种变异算子生成变异体，得到的变异体信息在[Result]文本框中显示，如图 4-12 所示，该实例仅生成 39 个变异体。

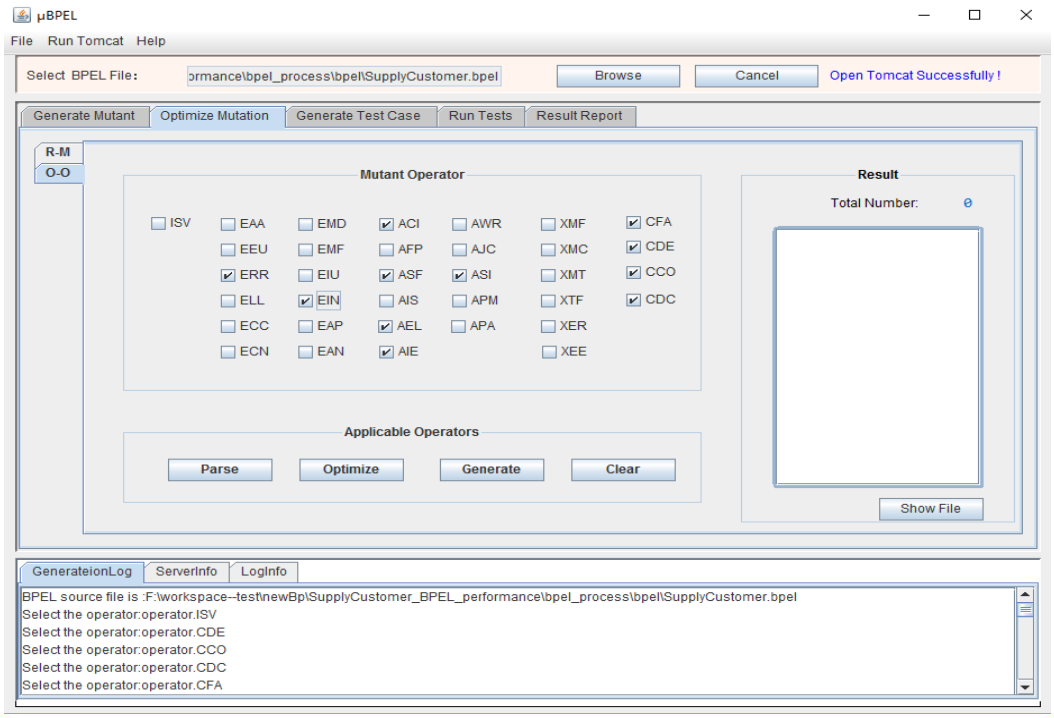

图 4-11 基于包含关系的变异体优化技术界面

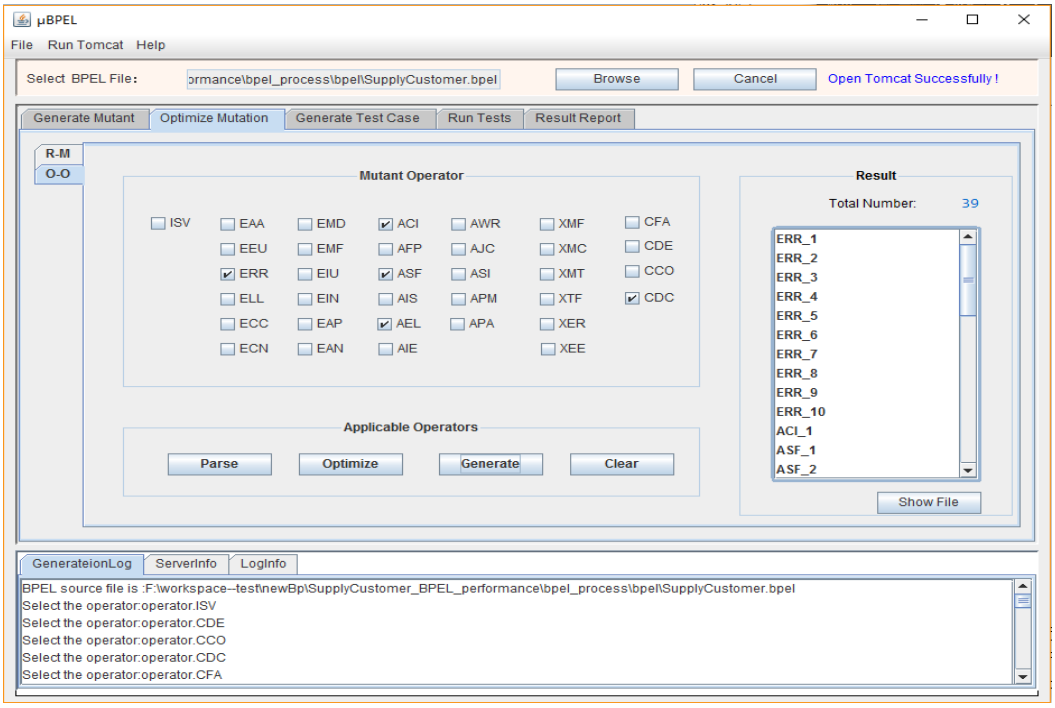

图 4-12 基于包含关系的变异体优化技术执行结果

(3) 测试执行演示

通过以上步骤，得到变异体集合和测试用例集合，可以进行变异测试。通过[Run Tests]标签页，进入执行测试界面。在对 BPEL 程序执行测试前，需

要先将其部署到 Tomcat 上，μBPEL 通过 build.xml 脚本来完成 BPEL 服务的部署。然后，输入待执行的 BPEL 程序和测试用例文件，完成基本测试环境的配置。点击[Run Test Cases]按钮，μBPEL 将会自动获取程序的输入端口等信息，执行测试用例并将测试结果保存在文件中，如图 4-13 所示。

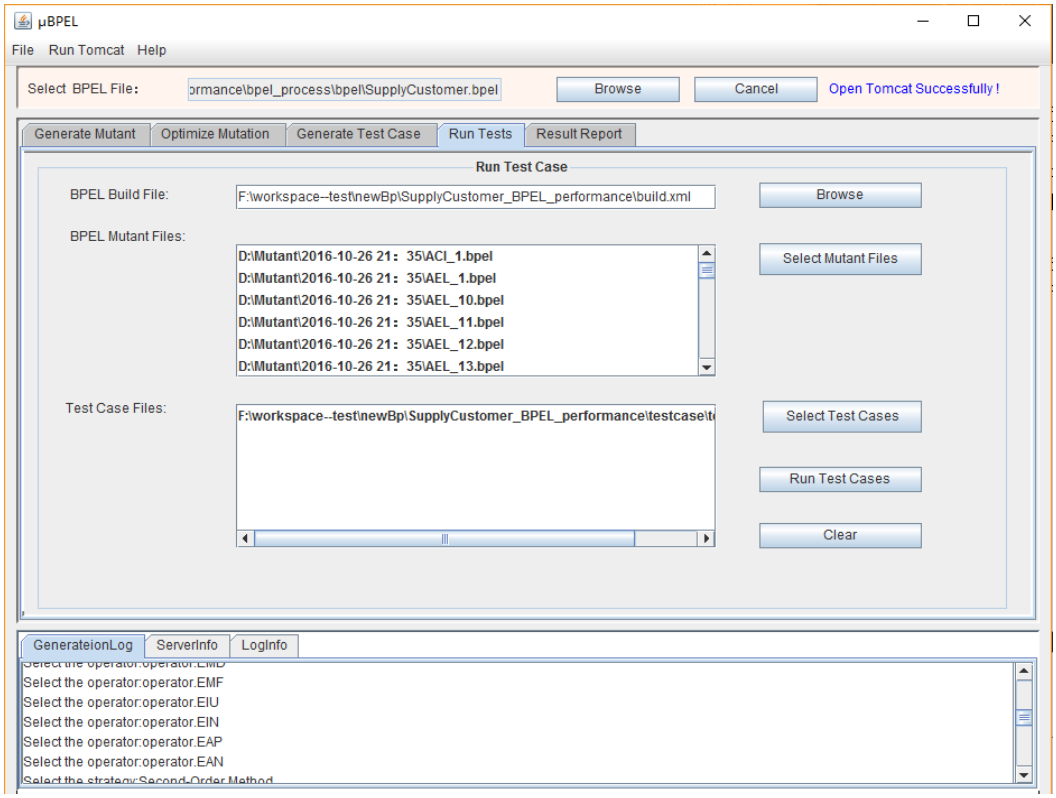

图 4-13 测试用例执行界面

(4) 测试结果分析演示

完成测试用例的执行，还需要对测试结果进行统计。通过点击[Result Report]菜单项，进入结果分析界面。点击[Browse]按钮选择原始程序测试结果和[Select Mutatnt Output]按钮选取所有变异体的测试用例执行结果，单击[Run]按钮对它们的测试结果进行对比，并输出结果。

系统统计变异体的故障检测率并显示在界面上。如图 4-14 所示，变异体的测试状态会在[Status]的项中显示，其中，“F”表示该条测试用例杀死变异体；“T”表示该条测试用例没有杀死变异体。[FinalStatus]状态栏显示该组测试用例集合是否杀死”变异体，其中“SUCCESS”表示变异体存活；“KILLED”表示变异体被“杀死”。[Result]栏中显示实例的变异测试报告。本实例中，被杀死的变异体数量共59个、总变异体数量共73个，最终的变异得分为81%。

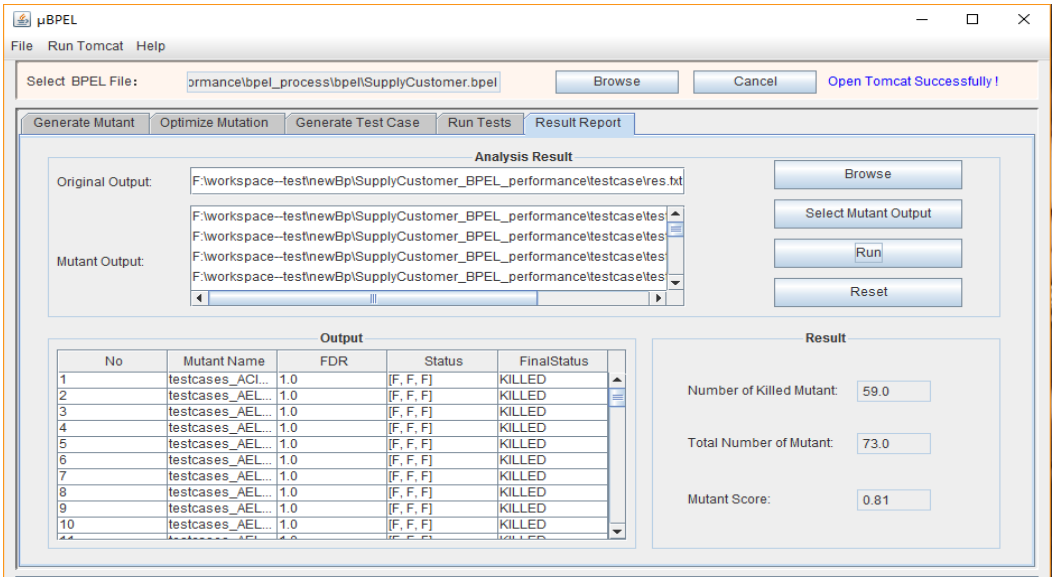

图 4-14 测试执行结果

4.4 小结

本章详细讨论了 BPEL 程序的变异测试集成化工具 µBPEL 的设计与实现。µBPEL 支持 BPEL 程序变异测试的全过程，同时实现了本文提出的部分变异测试优化技术，有助于测试人员对 BPEL 进行高效的变异测试。

5 经验研究

本章对课题组前期的经验研究进行扩充，以增强实验结果的可靠性；采用经验研究评估所提出的优化技术的有效性。

5.1 实验对象与度量指标

(1) 实验对象

本文实验对象共包括 6 个 BPEL 程序实例。SupplyChain[10]是一个供销链管理的实例。客户输入商品的名字和数量，零售商将会根据库存的状况反馈信息。SmartShelf[10]是一个商品货架管理的实例。用户输入货物的相关信息，例如货物名称、数量，系统处理这些信息并输出商品库存数量、货架位置和库存状态三个信息。SupplyCustomer[2]是项目订单管理的实例。用户输入订单的名称和地址，系统处理并反馈订单的查询结果。LoanApproval[2]是贷款审批的实例。用户输入个人信息及贷款金额，系统进行处理并反馈贷款审核结果。CarEstimate[41]是用于汽车评估的实例。用户发出评估请求，系统提供初步、简单、复杂评估方式，并返回最终评估结果。TravelAgency[42]是一个旅行社预订实例，组合旅店预订、订票、旅行社预订、和银行结算服务。用户提供预订旅客的信息和人数，系统进行处理并反馈结果。表 5-1 给出这些实例的具体信息，包括实例的基本功能描述（Basic functionality）、组装的服务数目（No. of services composed）和实例的规模（Size）。文中，使用 P1 至 P6 依次代表 SupplyChain、SmartShelf、SupplyCustomer、LoanApproval、CarEstimate 和 TravelAgency 实例程序。

表 5-1 BPEL 的实例程序基本信息

Program	Basic functionality	No. of services composed	Size (LOC)
P1	Management of supply chains	2	50
P2	Management of commodity shelves	14	194
P3	Management of project orders	5	122
P4	Examination of loan applications	3	120
P5	Assessment of car repairs	7	121
P6	Booking of travels	9	543

(2) 度量指标

使用如下三个指标来度量变异测试优化方法的有效性。

1) 变异得分（Mutation Score）

定义测试用例集杀死的非等价变体的比例[4]，用来衡量测试的充分程度。变异得分计算公式如下：

$$MS(P,TS)=\frac{N_k}{N_m-N_e} \tag{5-1}$$

其中，P 代表被测程序，TS 代表测试数据集，N_k 表示被杀死的变异体数量，N_m 表示变异体总数量，N_e 代表等价变异体的数量。变异得分越高，说明越多的非等价变异体被测试用例集“杀死”，测试用例集越有效。

2) 故障检测率

定义测试用例能够有多少比例杀死变异体，具体信息在文章 3.3 进行了介绍。

3) APFD（Average of the Percentage of Faults Detected）

一种基于错误的评价标准，多用于根据测试集的错误检测效率来衡量不同优先级技术的排序效果[43]。APFD 的计算公式如下：

$$APFD(TS,P)=1-\frac{\sum_{i=1}^{m} reveal(i,TS)}{mn}+\frac{1}{2n} \tag{5-2}$$

其中，TS 表示具有顺序的测试用例集，P 表示待测程序，n 表示测试用例的个数，m 表示被检测错误的总数，$reveal(i, TS)$表示最早检测出错误 i 所执行的测试用例的位置。APFD 量化了测试用例序列的效率和效能[44]。本文采用 APFD 来度量序列化的测试用例的故障检测效率。APFD 的数值越大，说明该测试用例集检测错误速率越快、效率越高。

5.2 扩充的实例研究

对课题组面向 BPEL 的变异测试的实验进行扩充，以增强相关研究成果的可靠性。

5.2.1 实验步骤

对实验对象进行变异测试，实验步骤如下：

(1) 选取一个实例程序 P，使用 μBPEL 为该程序生成一阶变异体；将这些一阶变异体组成集合，记为 M。

(2) 使用不同用例生成技术构造测试用例集合 TS。在实例程序 P 和一阶变异体集合 M 上运行测试用例集合 TS，记录测试结果。

(3) 分析测试结果，找出没有被测试用例集 *TS*“杀死”的一阶变异体，识别出其中的等价变异体。

(4) 使用 μBPEL 统计故障检测率 FDR 和变异得分 MS。

5.2.2 实验结果

(1) 变异体生成

使用 μBPEL 为实例自动化生成一阶变异体，结果如表 5-2 所示。

表 5-2 BPEL 实例的变异体生成

Program	No. of generated first-order mutants	No. of used operators
P1	34	11
P2	170	11
P3	73	11
P4	85	17
P5	54	5
P6	171	14

(2) 测试用例生成

课题组前期工作中主要使用了等价类划分[45]和边界值分析[45]两种技术生成测试用例。为了减少不同测试用例生成技术对变异算子的经验研究成果的影响，本文新增两种测试用例生成技术：面向场景测试技术[36]和随机测试技术[45]，来增加实验结果的通用性。面向场景的测试技术首先将 BPEL 程序转换为图模型结构，获取测试场景，最后根据不同场景产生用例。随机测试以输入和约束的范围生成用例。前期工作产生的三组测试用例集合分别命名为 T_x、T_y 和 T_z，新增的技术产生的测试用例集，命名为 T_u 和 T_w 。表 5-3 给出每个 BPEL 实例的测试用例集数量。需要指出的是，测试用例集的数量依赖于程序的规模和所使用的测试用例生成技术。对于小规模的程序所产生的用例集的数量相对较少。例如，SupplyChain 和 SmartShelf 实例分别由最少和最多数量的服务组装而成，它们所关联的用例数量也是最少和最多的；另一方面，不同的测试技术也会导致不同测试用例的产生。

表 5-3 BPEL 实例的测试用例集合

Program	$\|T_x\|$	$\|T_y\|$	$\|T_z\|$	$\|T_u\|$	$\|T_w\|$
P1	6	10	15	19	30
P2	22	35	50	74	90
P3	7	12	18	24	30
P4	12	18	25	31	40
P5	10	20	30	36	40
P6	4	7	10	14	20

(3) 测试结果

对实验结果进行统计，计算故障检测率(FDR)和变异得分(MS)，结果如表 5-4 和 5-5 所示。表 5-4 给出不同测试用例集合下，变异体的 FDR 值范围，其中 NA 表示由此变异算子产生的变异体都是等价变异体，无法统计 FDR 值。表 5-5 给出实例变异测试的变异得分。

表 5-4 6 个 BPEL 实例的故障检测率 FDR

(a) SupplyChain

Oper-ator	Value range of FDR				
	T_x	T_y	T_z	T_u	T_w
ACI	{100%,100%}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}
AEL	{50%,100%}	{40%,100%}	{40%,100%}	{42.1%100%}	{36.7%,100%}
AIE	{50%,50%}	{60%,60%}	{60%,60%}	{57.9%,57.9}	{63.3%,63.3}
ASI	{50%,100%}	{40%,100%}	{40%,100%}	{42.1%,100%}	{36.7%,100%}
ASF	{50%,100%}	{40%,100%}	{40%,100%}	{42.1%,100%}	{36.7%,100%}
ERR	{16.7%,100}	{10%,100%}	{6.7%,100%}	{5.3%,100%}	{3.3%,100%}
EIN	{100%,100%}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}
CFA	{50%,100%}	{40%,100%}	{40%,100%}	{42.1%100%}	{36.7%,100%}
CDE	{50%,50%}	{40%,60%}	{40%,60%}	{42.1%,57.9}	{36.7%,63.3}
CDC	{50%,50%}	{40%,60%}	{40%,60%}	{42.1%,57.9}	{36.7%,63.3}
CCO	{50%,50%}	{40%,60%}	{40%,60%}	{42.1%,57.9}	{36.7%,63.3}

(b) SmartShelf

Oper-ator	Value range of FDR				
	T_x	T_y	T_z	T_u	T_w
ACI	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}
AEL	{ 18.2%,100%}	{20%,100%}	{20%,100%}	{21.6%,100%}	{24.4%,100%}
AIE	{40.9%,81.9%}	{37.1%,80%}	{36%,80%}	{33.8%,82.4%}	{30%,75.6%}
ASI	{ 18.2%,100%}	{20%,100%}	{20%,100%}	{21.6%,100%}	{24.4%,100%}
ASF	{ 18.2%,100%}	{20%,100%}	{20%,100%}	{21.6%,100%}	{24.4%,100%}
ERR	{0%,100%}	{0%,100%}	{0%,100%}	{0%,100%}	{0%,100%}
EIN	{59.1%,100%}	{60%,100%}	{64%,100%}	{63.5%,100%}	{66.7%,100%}
CFA	{ 18.2%,100%}	{20%,100%}	{20%,100%}	{21.6%,100%}	{24.4%,100%}
CDE	{18.2%,81.9%}	{20%,80%}	{20%,80%}	{21.6%,78.4%}	{24.4%,75.6%}
CCO	{18.2%,81.9%}	{20%,80%}	{20%,80%}	{21.6%,78.4%}	{24.4%,75.6%}
CDC	{18.2%,81.9%}	{20%,80%}	{20%,80%}	{21.6%,78.4%}	{24.4%,75.6%}

(c) SupplyCustomer

Oper-ator	Value range of FDR				
	T_x	T_y	T_z	T_u	T_w
ACI	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}
ASF	{57.1%,100%}	{50%,100%}	{50%,100%}	{50%,100%}	{46.7%,100%}
AIE	{42.9%,42.9}	{50%,50%}	{50%,50%}	{50%,50%}	{53.3%,53.3}
ASI	{14.3%,100%}	{8.3%,100%}	{11.1%,100%}	{12.5%,100%}	{13.3%,100%}
AEL	{14.3%,100%}	{8.3%,100%}	{11.1%,100%}	{12.5%,100%}	{13.3%,100%}
ERR	{14.3%,100%}	{8.3%,100%}	{11.1%,100%}	{12.5%,100%}	{13.3%,100%}
EIN	{57.1%,100%}	{50%,100%}	{50%,100%}	{50%,100%}	{46.7%,100%}
CFA	{14.3%,100%}	{8.3%,100%}	{11.1%,100%}	{12.5%,100%}	{13.3%,100%}
CDE	{14.3%,85.7}	{8.3%,91.7%}	{11.1%,88.9}	{12.5%,87.5%}	{13.3%,86.7}
CCO	{14.3%,85.7}	{8.3%,91.7%}	{11.1%,88.9}	{12.5%,87.5%}	{13.3%,86.7}
CDC	{14.3%,85.7}	{8.3%,91.7%}	{11.1%,88.9}	{12.5%,87.5%}	{13.3%,86.7}

(d) CarEstimate

Oper-ator	Value range of FDR				
	T_x	T_y	T_z	T_u	T_w
ACI	{100%,100}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}
AEL	{100%,100}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}
ASI	{100%,100}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}
ASF	{100%,100}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}
CFA	{100%,100}	{100%,100}	{100%,100}	{100%,100%}	{100%,100%}

(e) LoanApproval

Oper-ator	Value range of FDR				
	T_x	T_y	T_z	T_u	T_w
ISV	{8.3%,8.3%}	{11.1%,11.1}	{12%,12%}	{12.9%,12.9%}	{12.5%,12.5}
ACI	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}
AEL	{0%,100%}	{0%,100%}	{0%,100%}	{0%,100%}	{0%,100%}
ASI	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}	{100%,100%}
ASF	{8.3%,8.3%}	{11.1%,11.1}	{12%,12%}	{12.9%,12.9%}	{12.5%,12.5}
ECN	{8.3%,50%}	{5.6%,50%}	{4%,48%}	{3.2%,38.7%}	{2.5%,32.5%}
ERR	{8.3%,83.3%}	{5.6%,77.8%}	{4%,72%}	{3.2%,71%}	{2.5%,65%}
EIU	{25%,75%}	{22.2%,66.7}	{20%,50%}	{19.4%,58.1%}	{17.5%,52.5}
EIN	{8.3%,83.3%}	{11.1%,77.8}	{12%,72%}	{12.9%,71%}	{12.5%,65%}
EAN	{25%,75%}	{22.2%,66.7}	{20%,50%}	{19.4%,58.1%}	{17.5%,52.5}
ECC	NA	NA	NA	NA	NA
EAP	NA	NA	NA	NA	NA
XMF	{0%,0%}	{0%,0%}	{0%,0%}	{0%,0%}	{0%,0%}
CFA	{0%,100%}	{0%,100%}	{0%,100%}	{0%,100%}	{0%,100%}
CDE	{8.3%,75%}	{11.1%,66.7}	{12%,60%)	{12.9%,58.1%}	{12.5%,52.5}
CCO	{8.3%,75%}	{11.1%,66.7}	{12%,60%)	{12.9%,58.1%}	{12.5%,52.5}
CDC	{8.3%,75%}	{11.1%,66.7}	{12%,60%)	{12.9%,58.1%}	{12.5%,52.5}

(f) TravelAgency

Oper-ator	Value range of FDR				
	T_x	T_y	T_z	T_u	T_w
ACI	{100%,100}	{100%,100%}	{100%,100}	{100%,100}	{100%,100}
AEL	{50%,100%}	{42.9%,100%}	{50%,100%}	{50%,100%}	{45%,100%}
AIE	{50%,50%}	{57.1%,57.1}	{50%,50%}	{50%,50%}	{45%,45%}
ASI	{50%,100%}	{42.9%,100%}	{50%,100%}	{50%,100%}	{45%,100%}
ASF	{50%,100%}	{42.9%,100%}	{50%,100%}	{50%,100%}	{45%,100%}
ERR	{25%,100%}	{14.3%,,100}	{%10,100%}	{7.1%,100%}	{5%,100%}
ECN	{25%,50%}	{14.3%,57.1}	{10%,50%}	{7.1%,50%}	{5%,50%}
EAN	{50%,50%}	{42.9%,42.9}	{50%,50%}	{50%,50%}	{55%,55%}
EIU	{50%,50%}	{42.9%,42.9}	{50%,50%}	{50%,50%}	{55%,55%}
EIN	{100%,100}	{100%,100%}	{100%,100}	{100%,100}	{100%,100}
ECC	NA	NA	NA	NA	NA
EAP	NA	NA	NA	NA	NA
CFA	{50%,100%}	{42.9%,100%}	{50%,100%}	{50%,100%}	{45%,100%}
CDE	{50%,50%}	{42.9%,57.1}	{50%,50%}	{50%,50%}	{45%,55%}
CCO	{50%,50%}	{42.9%,57.1}	{50%,50%}	{50%,50%}	{45%,55%}
CDC	{50%,50%}	{42.9%,57.1}	{50%,50%}	{50%,50%}	{45%,55%}

表 5-5 6 个 BPEL 实例的变异得分 MS

Program	P1	P2	P3	P4	P5	P6
MS	100%	99.3%	100%	96.3%	100%	100%

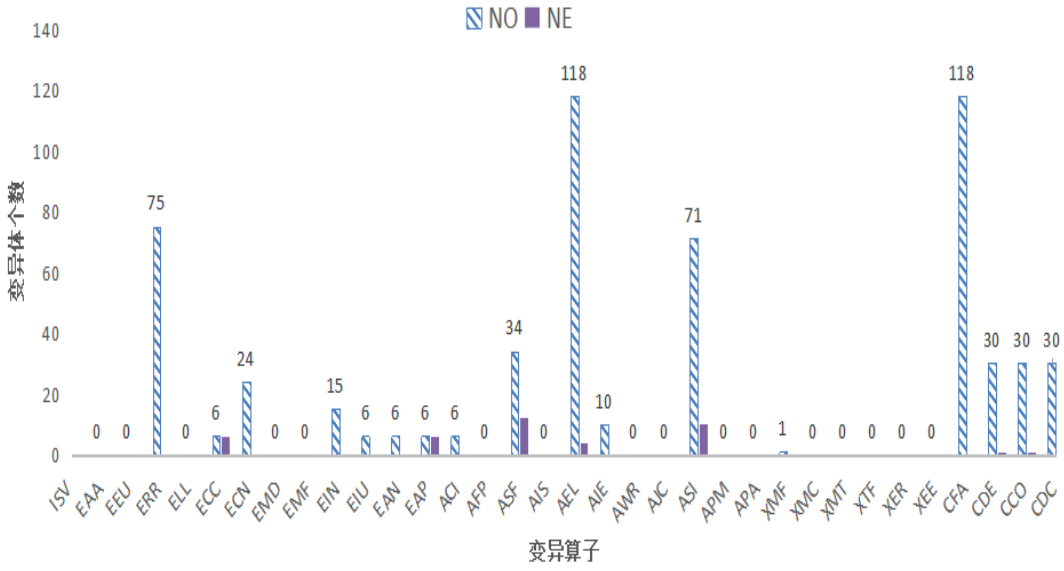

图 5-1 变异算子产生变异体情况

基于以上实验结果，统计不同变异算子产生变异体的情况，如图 5-1 所示。在图 5-1 中，*NO* 指由此变异算子产生的变异体数目，*NE* 指由此变异算子产生的等价变异体数目。通过分析这些数据，可以看出：

(1) EAA，EEU，ELL，EMD，EMF，AFP，AIS，AWR，AJC，APM, APA, XMC，XMT，XTF，XER，XEE 这 16 种变异算子在这 6 个 BPEL 实例中不适用。

(2) AEL，CFA，ERR，ASI 这四类变异算子相对于其他变异算子可产生较多变异体。

(3) ECC，EAP 这两种变异算子产生的变异体都是等价变异体；ASF，ASI，AEL 这三类变异算子会产生大量的等价变异体。

5.3 变异测试优化技术实例评估

采用经验研究方法验证和评估本文提出的优化技术的有效性。

5.3.1 MuSR 技术评估

在文章 3.2 节，提出了变异算子的包含关系规则，并总结出 7 组具有包含关系的变异算子。本部分实验运用包含关系规则验证这 7 组包含关系的正确性。同时，评估 MuSR 技术对于 BPEL 程序的变异测试优化的效果。

根据包含关系规则定义可知，若 $O_A \leftarrow O_B$，则说明对于变异算子 O_B 产生的所有变异体，都存在变异算子 O_A 产生的变异体，使得所有能将 O_A 产生的变异体“杀死”的测试用例，都能“杀死”O_B 产生的变异体。O_B 产生的变异体可以被 O_A 的变异体所替代，所以 O_B 的变异体可以被精简。

通过查询 5.2 节中实例的变异测试结果，统计 7 组具有包含关系的变异算子产生的变异体的 FDR 值，观察它们的测试情况。FDR 值给出测试用例能够有多少比例“杀死”变异体，可以间接反映出变异算子间的包含关系。若 *AEL*←*AIE*，根据包含关系定义，可知 *AEL* 变异体的 FDR 值应小于等于 *AIE* 变异体的 FDR 值。对每组变异算子之间的包含关系找一个 BPEL 实例进行验证，得到如表 5-6 至 5-11 所示结果。表 5-6 至 5-11 给出在五组测试用例集合下，这些具有包含关系的变异算子产生变异体的 FDR 值。其中，表 5-7 展示 SupplyCustomer 实例中存在 *AEL*←*AIE* 这组包含关系。在五组测试用例集合下，*AEL* 和 *AIE* 的变异体的 FDR 值相同，表明所有能将 *AEL* 的 MU_1^{AEL} 或 MU_2^{AEL} 变异体的“杀死”的测试用例，都能将 *AIE* 变异算子的 MU_1^{AIE} 变异体“杀死”。所以，*AIE* 和 *AEL* 满足包含规则，*AEL*←*AIE* 成立。同理，其余 6 组变异算子的包含关系也成立。

表 5-6 SupplyChain 的 ASI 和 ASF 算子情况

Operator	Mutant	Value of FDR				
		T_x	T_y	T_z	T_u	T_w
ASI	MU_1^{ASI}	100%	100%	100%	100%	100%
	MU_2^{ASI}	100%	100%	100%	100%	100%
	MU_3^{ASI}	100%	100%	100%	100%	100%
	MU_4^{ASI}	50%	40%	40%	42.1%	36%
ASF	MU_1^{AEL}	100%	100%	100%	100%	100%
	MU_2^{AEL}	50%	40%	40%	42.1%	36%

表 5-7 SupplyCustomer 的 AIE 和 AEL 算子情况

Operator	Mutant	Value of FDR				
		T_x	T_y	T_z	T_u	T_w
AIE	MU_1^{AIE}	42.9%	50%	50%	50%	53.3%
AEL	MU_1^{AEL}	42.9%	50%	50%	50%	53.3%
	MU_2^{AEL}	42.9%	50%	50%	50%	53.3%

表 5-8 SupplyChain 的 CCO、CDE 和 CDC 算子情况

Operator	Mutant	Value of FDR				
		T_x	T_y	T_z	T_u	T_w
CCO	MU_1^{CCO}	50%	40%	40%	42.1%	36.7%
	MU_2^{CCO}	50%	60%	60%	57.9%	63.3%
CDE	MU_1^{CDE}	50%	40%	40%	42.1%	36.7%
	MU_2^{CDE}	50%	60%	60%	57.9%	63.3%
CDC	MU_1^{CDC}	50%	40%	40%	42.1%	36.7%
	MU_2^{CDC}	50%	60%	60%	57.9%	63.3%

表 5-9 SupplyChain 的 CFA 和 AEL 算子情况

Operator	Mutant	Value of FDR				
		T_x	T_y	T_z	T_u	T_w
CFA	MU_1^{CFA}	100%	100%	100%	100%	100%
	MU_2^{CFA}	100%	100%	100%	100%	100%
	MU_3^{CFA}	50%	40%	40%	42.1%	36.7%
	MU_4^{CFA}	50%	40%	40%	42.1%	36.7%
	MU_5^{CFA}	50%	40%	40%	42.1%	36.7%
	MU_6^{CFA}	50%	60%	60%	57.9%	63.3%
	MU_7^{CFA}	100%	100%	100%	100%	100%
AEL	MU_1^{AEL}	100%	100%	100%	100%	100%
	MU_2^{AEL}	100%	100%	100%	100%	100%
	MU_3^{AEL}	50%	40%	40%	42.1%	36.7%
	MU_4^{AEL}	50%	40%	40%	42.1%	36.7%
	MU_5^{AEL}	50%	40%	40%	42.1%	36.7%
	MU_6^{AEL}	50%	60%	60%	57.9%	63.3%
	MU_7^{AEL}	100%	100%	100%	100%	100%

表 5-10 TravelAgency 的 EIN 和 ERR 算子情况

Operator	Mutant	Value of FDR				
		T_x	T_y	T_z	T_u	T_w
EIN	MU_1^{EIN}	100%	100%	100%	100%	100%
	MU_2^{EIN}	100%	100%	100%	100%	100%
	MU_3^{EIN}	100%	100%	100%	100%	100%
	MU_4^{EIN}	100%	100%	100%	100%	100%
ERR	MU_1^{ERR}	100%	100%	100%	100%	100%
	MU_2^{ERR}	100%	100%	100%	100%	100%
	MU_3^{ERR}	100%	100%	100%	100%	100%
	MU_4^{ERR}	100%	100%	100%	100%	100%

表 5-11 LoanApproval 的 EAN 和 EIU 算子情况

Operator	Mutant	Value of FDR				
		T_x	T_y	T_z	T_u	T_w
EAN	MU_1^{EAN}	75%	66.7%	50%	58.1%	52.5%
	MU_2^{EAN}	25%	22.2%	20%	19.4%	17.5%
EIU	MU_1^{EIU}	75%	66.7%	50%	58.1%	52.5%
	MU_2^{EIU}	25%	22.2%	20%	19.4%	17.5%

统计 6 个 BPEL 实例中与这 7 组包含关系关联的变异体，依据 MuSR 技术可知，这些变异体可以被精简。结果如表 5-12 所示。其中，属于 *AEL←AIE* 的变异体有 11 个，属于 *ASF ← ASI* 的变异体有 61 个，属于 *AEL←CFA* 的变异体有 118 个，属于 *ERR ← EIN* 的变异体有 15 个，属于 *EIU ← EAN* 的变异体有 6 个，属于 *CDC ← CDE* 和 *CDC ←CCO* 的变异体各有 30 个，共有 271 个变异体可被精简。由包含规则的定义可知，这 271 个变异体所代表的错误完全可以被其它变异体所替代。精简这部分变异体，可以减少 46.2%比例的变异体开销。由此可见，MuSR 技术极大的减少了待测变异体的数目，提高了测试效率。

表 5-12 6 个 BPEL 程序中具有包含关系变异算子组合的总结

Program		P1	P2	P3	P4	P5	P6	Total
No. of mutants pairs	AEL←AIE	1	4	1	0	0	5	11
	ASF←ASI	4	19	6	1	15	16	61
	AEL←CFA	7	40	17	8	16	30	118
	ERR←EIN	1	4	2	4	0	4	15
	EIU←EAN	0	0	0	2	0	4	6
	CDC←CCO	2	8	4	8	0	8	30
	CDC←CDE	2	8	4	8	0	8	30
Total		17	83	34	31	31	75	271
Reduction		50%	48.8%	46.6%	36.5%	57.4%	43.8%	46.2%

5.3.2 MuSOM 技术评估

(1) 实验步骤

对 BPEL 实例的一阶变异体和二阶变异体进行变异测试，来评估 MuSOM 技术的有效性。图 5-2 给出具体的实验流程。

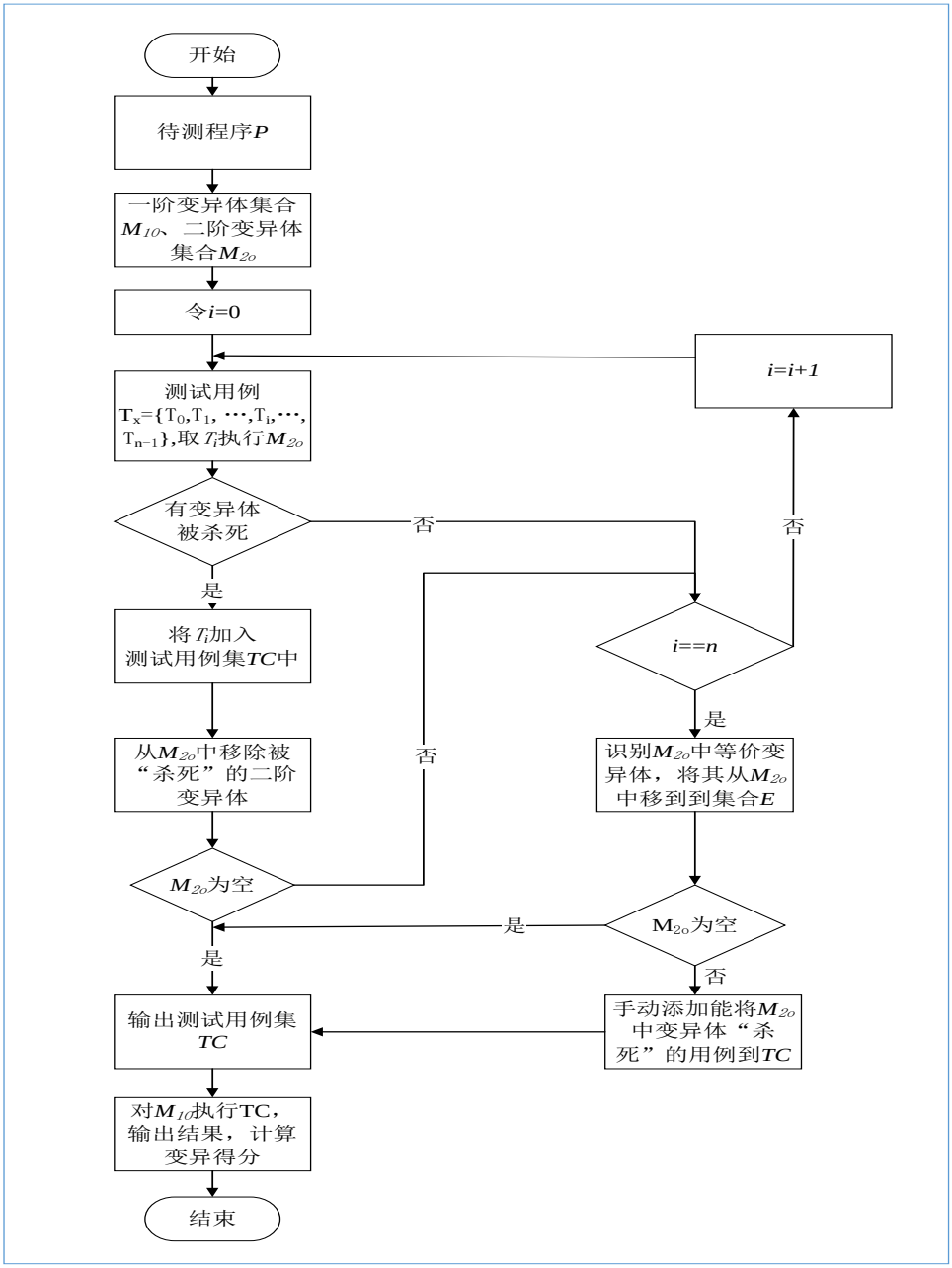

图 5-2 二阶变异体测试过程

实验过程描述如下：

1) 输入实例源程序 *P*，使用 μBPEL 工具来自动化生成变异体，得到一阶变异体集合，记为 M_{1o} 和二阶变异体集合，记为 M_{2o}；

2) 选取 5.2 节中的测试用例集 T_x，$T_x=\{T_0,T_1,\ldots,T_i,\ldots,T_{n-1}\}$；令 $n=|T_x|$，$i=0$；
3) 用 T_i 执行所有的二阶变异体和原始程序 P，将结果保存在文件中；比较二阶变异体和原始程序的输出结果，若二者不同，说明 T_i “杀死”该二阶变异体；否则，说明该变异体“存活”；
4) 若 M_{2o} 中存在被 T_i “杀死”的二阶变异体，则将 T_i 加入到测试用例集合 TC 中，并从 M_{2o} 中删除这些被 T_i “杀死”的二阶变异体；判断 M_{2o} 集合是否为空，如果不是，则执行第五步；否则，则执行第八步；
5) 判断 i 是否等于 n，如果不是，则令 $i=i+1$，返回第三步；否则，执行第六步；
6) 人工识别 M_{2o} 中的等价变异体，将这些等价变异体从 M_{2o} 中删除并加入到集合 E 中，执行下一步；
7) 判断 M_{2o} 集合是否为空，如果不是，则手动添加能将 M_{2o} 中变异体“杀死”的测试用例，并将这些测试用例添加到 TC；否则，执行第八步；
8) 用测试用例集合 TC，执行一阶变异体 M_{1o}，保存测试结果；
9) 使用 μBPEL 工具统计一阶变异体的变异得分 MS。

(2) 实验结果

按照上小节的实验步骤，对实例进行实验。实验结果如表 5-13 和图 5-3 所示。表 5-13 给出实验中产生的一阶变异体和二阶变异体情况。其中，P 指待测的原始程序；*Mutants* 列表示使用 μBPEL 工具生成的一阶变异体集合 M_{1o} 和二阶变异体集合 M_{2o}；NS 表示应用的变异算子数目；NM 表示产生的变异体程序数量；NE 表示等价变异体程序的数量。图 5-3 给出实例中的一阶变异体集合和二阶变异体集合的变异得分情况。

表 5-13 6 个 BPEL 程序的实验结果

P	Mutants	NS	NM	NE
P1	M_{1o}	11	34	0
	M_{2o}	11	17	0
P2	M_{1o}	11	170	13
	M_{2o}	11	85	0
P3	M_{1o}	11	73	5
	M_{2o}	11	37	0
P4	M_{1o}	17	85	7
	M_{2o}	17	43	0
P5	M_{1o}	5	54	7
	M_{2o}	5	27	0
P6	M_{1o}	14	171	16
	M_{2o}	14	86	0

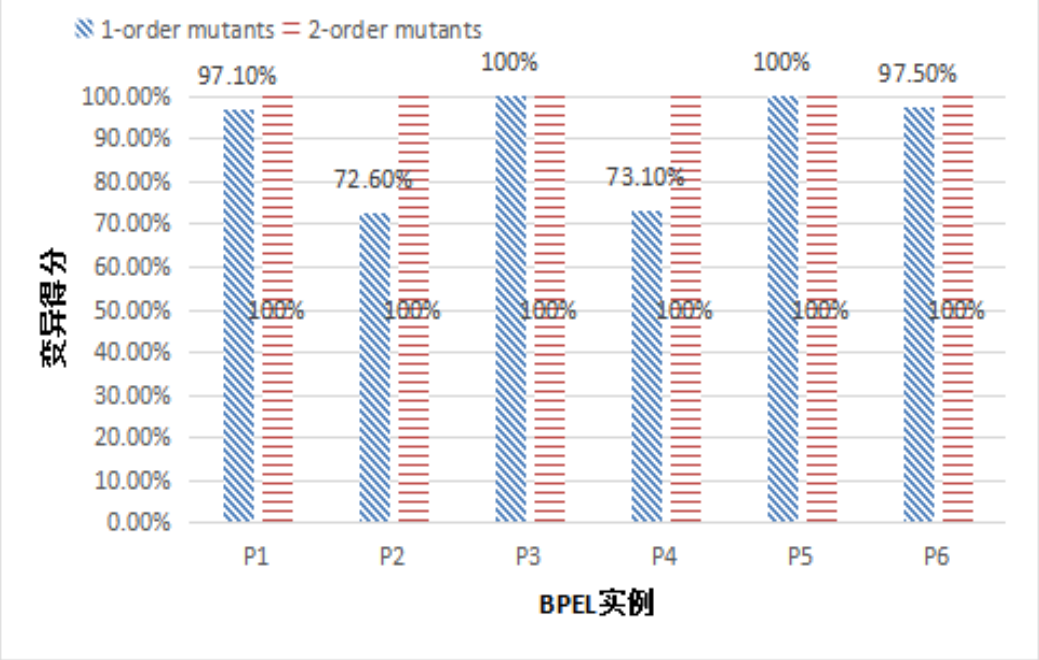

图 5-3 6 个 BPEL 程序的一阶与二阶变异体实验变异得分

通过实验结果，可以得出如下结论：

1) 在实验中，应用同样数目和种类的变异算子，M_{2o} 集合数量约为 M_{1o} 的一半，减少了约 50%的待测变异体。
2) M_{2o} 中的等价变异体数目（*NE*）总和为 0，而 M_{1o} 中存在 48 个等价变异体。可见，MuSOM 技术大幅度降低等价变异体的识别开销。
3) 对于测试用例集 *TC*，二阶变异体的变异得分均为 100%，而一阶变异体的变异得分不尽相同。其中，*P3* 和 *P5* 实例的变异得分为 100%；*P1* 和 *P6* 实例的变异得分分别是 97.1%和 97.5%，接近于 100%；而 *P2* 和 *P4* 实例的变异得分分别是 72.6%和 73.1%。可见，绝大多数一阶变异体可以被 *TC* “杀死”。

综上所述，使用二阶变异体进行测试，相对于一阶变异体，可以减少近 50%的变异体测试开销和 8%的等价变异体的识别开销。同时，二阶变异体并没有大幅度降低衡量测试用例集故障检测的能力。

5.3.3 MuPri 技术评估

根据 5.2 节的实例的变异测试结果，计算变异算子的质量 Q_o，为变异算子进行优先级排序，得到表 5-14 结果。其中，“-”表示实例中该变异算子不适用；*Average* 是对 6 个实例中的变异算子 Q_o 值取得的平均值。根据这个值的大小为变异算子排序并分配优先级，优先级显示在 *Rank* 列中。Rank 中的

数值越小，表示该变异算子的优先级越高。优先级高的变异算子表示其产生的变异体越难被杀死，在检验测试用例有效性方面效果更好。需要说明的是，由于在这 6 个实例程序中，只有 16 个变异算子可适用，所以，仅对这 16 个变异算子进行了排序。

表 5-14 变异算子排序

Operator	Q_o						Average	Rank
	P1	P2	P3	P4	P5	P6		
XMF	-	-	-	100.0%	-	-	100.0%	1
ISV	-	-	-	88.6%	-	-	88.6%	2
ECN	-	-	-	82.9%	-	71.9%	77.3%	3
CDE	50.0%	54.7%	50.0%	75.8%	-	50.0%	56.1%	4
CCO	50.0%	54.7%	50.0%	75.8%	-	50.0%	56.1%	5
CDC	50.0%	54.7%	50.0%	75.8%	-	50.0%	56.1%	6
EIU	-	-	-	59.4%	-	50.0%	54.7%	7
EAN	-	-	-	59.4%	-	50.0%	54.7%	8
ERR	40.0%	49.7%	58.0%	73.2%	-	40.0%	52.2%	9
AIE	41.8%	54.5%	50.8%	-	-	50.0%	49.3%	10
ASF	29.1%	11.3%	24.6%	88.6%	0.0%	42.9%	32.7%	11
AEL	30.9%	32.0%	48.1%	30.5%	0.0%	32.3%	29.0%	12
CFA	30.9%	32.0%	27.9%	12.5%	0.0%	32.3%	22.6%	13
EIN	0.0%	9.3%	24.6%	57.7%	-	0.0%	18.3%	14
ASI	14.6%	14.5%	22.9%	0.0%	0.0%	18.7%	11.7%	15
ACI	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	16

使用一组实验来验证 MuPri 技术得到的变异算子优先级的有效性。主要思路是使用 MuPri 技术给测试用例集进行排序；计算排序后用例集的 APFD 值，来反映用例排序的效果；从而评估 MuPri 技术有效性。

(1) 实验步骤

图 5-4 给出具体的实验流程，实验步骤描述如下：

1) 首先，按照第 3.3.2 节的一般过程步骤，根据 MuPri 技术得到面向 BPEL 程序 *P* 的、变异得分为 100%且具有一定顺序的测试用例队列 *TS*；相应步骤对应图 5-4 的第一部分；
2) 令 *i=1*，*sum=0*；
3) 使用 μBPEL 工具生成程序 *P* 的变异体集合 $R(P)=\{R_1,..., R_i, ...R_m\}$，*m* 为变异体的总数；
4) 依次取出 *TS* 中的测试用例执行 R_i，记录第一个将变异体“杀死”的测试用例在 *TS* 中的位置 *s*；
5) 令 *sum=sum+s*；判断 *i* 是否等于 *m*，如果不是，则令 *i=i+1*，返回上

一步；否则，执行下一步；

6) 计算测试用例集 *TS* 的 APFD 的值。

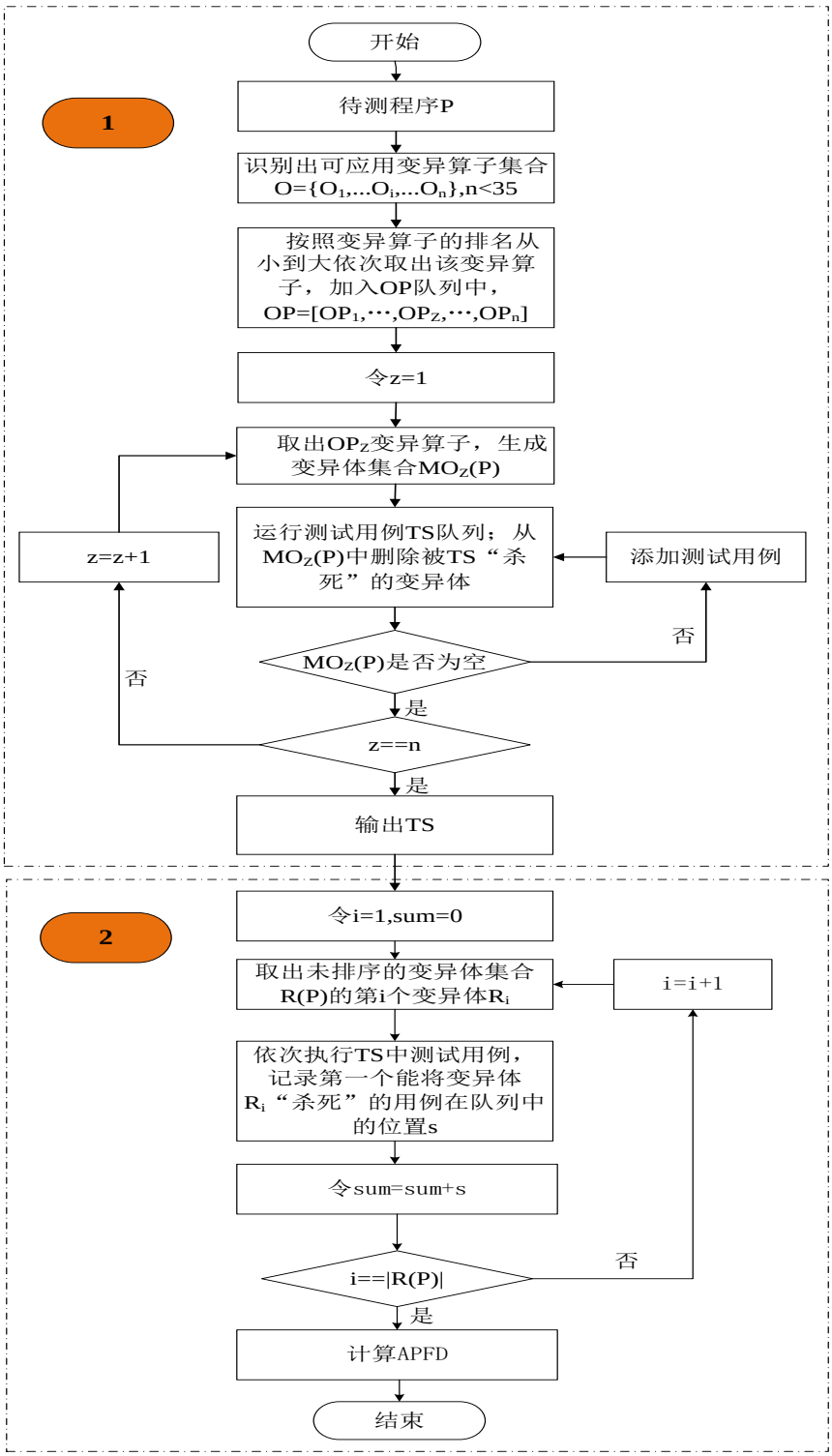

图 5-4 实验流程

(2) 实验结果

以 SupplyChain 实例为例阐述具体实验过程。对于 SupplyChain 实例，由

表 5-2 可知，该实例可以应用 11 种变异算子。由表 5-14 得到变异算子优先级序列为：CDE->CCO->CDC->ERR->AIE->ASF->AEL->CFA->EIN->ASI->ACI。根据这个序列，按照图 5-4 实验流程的第一部分，得到排序后的测试用例集是 *TS:T1->T2->T3*。这部分在 3.3.3 节已经介绍，这里不再详细讨论。

接下来，按照图 5-4 实验流程的第二部分进行实验。使用测试用例 *TS* 去执行没有排序的变异体集合，统计 APFD。结果如表 5-15 所示，其中对勾“√”表示第一个将变异体“杀死”的测试用例，*s* 表示用例的在 *TS* 中的位置，“×”表示该测试用例不能“杀死”变异体，“~”表示这个测试用例没有执行。根据 APFD 计算公式，得出这个测试用例集 *TS* 的 APFD 值是 74.5%。

表 5-15 变异体测试结果

Mutant		**Test Case**			**s**
		T1	T2	T3	
ERR	1	√	~	~	1
	2	×	×	√	3
	3	√	~	~	1
	4	×	√	~	2
	5	√	~	~	1
EIN	1	√	~	~	1
ACI	1	√	~	~	1
ASF	1	√	~	~	1
	2	√	~	~	1
AEL	1	√	~	~	1
	2	√	~	~	1
	3	√	~	~	1
	4	√	~	~	1
	5	√	~	~	1
	6	×	√	~	2
	7	√	~	~	1
AIE	1	×	√	~	2
ASI	1	√	~	~	1
	2	√	~	~	1
	3	√	~	~	1
	4	√	~	~	1
CFA	1	√	~	~	1
	2	√	~	~	1
	3	√	~	~	1
	4	√	~	~	1
	5	√	~	~	1
	6	×	√	~	2
	7	√	~	~	1
CDE	1	√	~	~	1
	2	×	√	~	2
CCO	1	√	~	~	1
	2	×	√	~	2
CDC	1	√	~	~	1
	2	×	√	~	2

对比实验不使用 MuPri 技术来为变异算子进行排序，仅按照表 2-2 中变异算子的顺序，为其生成变异体。然后按照 5-2 流程，得到的测试用例集顺序为 *TS:T1->T3>T2*，APFD 值为 69.6%。其余 5 个 BPEL 实例，采取与 SupplyChain 实例同样的实验过程，完成这两部分实验。统计实验数据，得到如表 5-16 所示的实验结果。

表 5-16 6 个 BPEL 实例实验的 APFD 值

Program name	P1	P2	P3	P4	P5	P6
Priority-method	74.5%	80.9%	94.1%	71.5%	50%	70.2%
No-Priority-method	69.6%	71.9%	94.1%	52.8%	50%	61.3%

表 5-16 中，“Priority-method”代表使用 MuPri 优化技术的实验，“No-Priority-method”代表对比实验。从结果可以看出，使用 MuPri 优化技术，得到的测试用例集合的 APFD 值都大于或等于对比实验的 APFD 值。由 APFD 的定义可知，APFD 值越大说明该测试用例集检测错误速率越快。所以，实验结果表明，使用 MuPri 优化技术来为测试用例集进行排序，将会提高测试用例集的检测错误效率，意味着该技术为变异算子排列的顺序也是有效的。

5.4 小结

本章使用 6 个 BPEL 实例并设计了三组不同的实验，对于第三章提出的变异测试优化技术逐一进行了验证与评估。从实验结果来看，MuSR 技术通过分析变异算子的操作，剔除具有包含关系的变异算子，减少将近一半的变异体生成；MuSOM 技术通过组合一阶变异体来生成二阶变异体，使得变异体数目减半，并且测试的变异得分也没有大幅度降低；使用 MuPri 技术为这 6 个实例中可适用的 16 种变异算子进行了优先级顺序，实验表明此顺序可以有效地为测试用例集排序，提高测试用例集的检测错误效率。

6 工作总结与展望

变异测试广泛用来评估测试技术的有效性和测试用例的完备性。然而，变异测试中大量变异体带来较高的测试开销，造成测试效率低而难以应用到实际中等问题。BPEL 是一种新型的服务组装语言，带来一些新的测试挑战。本文在课题组面向 BPEL 程序的变异测试研究的基础上，研究了面向 BPEL 程序的变异测试优化技术，取得的主要成果总结如下：

(1) 提出了基于包含关系、变异算子优先级和二阶变异体的三种面向 BPEL 程序的变异测试优化技术，极大的减少 BPEL 程序的变异测试开销。

(2) 开发了面向 BPEL 程序的变异测试集成化支持工具 μBPEL，支持变异测试的全过程，同时还支持本文提出的优化技术，方便测试人员对于 BPEL 程序不同需求的变异测试。

(3) 采用 6 个 BPEL 程序实例验证并评估提出的三种面向 BPEL 程序的变异测试优化技术的有效性。

工作不足及未来展望：

(1) 设计并检验了所提出的变异测试优化技术，需要进一步与其他相关的优化技术进行比较，评估优化技术的精简效率。

(2) 使用了 6 个 BPEL 程序进行实例验证，然而所选取的程序适用的变异算子种类较少，需要开发其他的实例程序进行评估优化技术的有效性。

参考文献

[1] 骆翔宇, 谭征, 苏开乐. 一种基于认知模型检测的 Web 服务组合验证方法 [J]. 计算机学报, 2011, 34(6): 1041-1061.

[2] OASIS Standard. Web services business process execution language version 2.0[EB/OL].http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2012.

[3] 宋波, 李妙妍. 面向Web服务的BPEL的研究与实现 [J]. 计算机工程与实现, 2007, 28(9): 2212-2214.

[4] DeMillo, R. A., Lipton, R. J., Sayward, F. G. Hints on test data selection: Help for the practicing programmer [J]. IEEE Computer, 1978, 11(4): 34-41.

[5] Estero-Botaro, A., Palomo-Lozano, F., and Medina-Bulo, I. Mutation Operators for WS-BPEL 2.0 [C]//Proceedings of the International Conferen -ce on Software and Systems Engineering and their Applications (ICSSEA'08), 2008: 1-7.

[6] Estero-Botaro, A., Palomo-Lozano, F., Medina-Bulo, I. Quantitative Evaluation of Mutation Operators for WS-BPEL Compositions [C]// Proceedings of 3rd International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), IEEE Computer Society, 2010: 142-150.

[7] 王巧玲. 面向 BPEL 程序的变异测试技术与支持工具研究 [D]. 北京科技大学, 2015.

[8] Sun. C.-A., Pan. L., Wang. Q. L., et al. An Empirical Study on Mutation Testing of WS-BPEL Programs [J]. The Computer Journal, 2016, in press (DOI:10.1093/comjnl/bxw076).

[9] Zhang, L., Hou, S.-S., Hu, J.-J., Xie, T., and Mei, H. Is operator-based mutant selection superior to random mutant selection? [C]//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), 2010: 435-444.

[10] Sun, C.-A., Zhai, Y. M., Zhang, Z. BPELDebugger: An effective BPEL -specific fault localization framework [J]. Information and Software Technology, 2013, 55(12): 2140-2153.

[11] 陈翔，顾庆. 变异测试：原理、优化和应用 [J]. 计算机科学与探索,2012, 6(12): 1057-1075.

[12] Jia, Y., Harman, M. An analysis and survey of the development of mutation

testing [J]. IEEE Transactions on Software Engineering, 2011, 37(5): 649-678.

[13] Fraser, G., Zeller, A. Mutation-driven generation of unit tests and oracles [C]//Proceedings of the19th International Symposium on Software Testing and Analysis (ISSTA'10), 2010: 147-158.

[14] Appelt, D., Nguyen, C. D., Briand, L. C., et al. Automated testing for SQL injection vulnerabilities: an input mutation approach [C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM, 2014: 259-269.

[15] Gopinath, R., Jensen, C., Groce, A. Code coverage for suite evaluation by developers [C]//Proceedings of the 36th International Conference on Software Engineering (ICSE'14), 2014: 72-82.

[16] Inozemtseva, L., Holmes, R. Coverage is not strongly correlated with test suite effectiveness [C]//Proceedings of the 36th International Conference on Software Engineering (ICSE'14), 2014: 435-445.

[17] Chen, T. Y., Kuo, F.-C., Liu, H., Wong, W. E. Code coverage of adaptive random testing [J]. IEEE Transactions on Reliability, 2013, 62(1): 226-237.

[18] Liu, H., Kuo, F.-C., Towey, D., Chen, T. Y. How effectively does metamorphic testing alleviate the oracle problem? [J]. IEEE Transactions on Software Engineering, 2014, 40(1): 4-22.

[19] Wright, C. J., Kapfhammer, G. M., and McMinn, P. The impact of equivalent, redundant and quasi mutants on database schema mutation analysis [C]//Proceedings of the 14th International Conference on Quality Software (QSIC'14), 2014:57-66.

[20] Zhang, L., Marinov, D., Zhang, L., and Khurshid, S. Regression mutation testing [C]//Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA'12), 2012: 331-341.

[21] Sun, C.-A., Zhao, Y., Pan, L., Liu, H., and Chen, T. Y. Automated Testing of WS-BPEL Service Compositions: a Scenario-oriented Approach [J]. IEEE Transactions on Services Computing, 2015, in press (DOI: 10.1109/TSC.2015.2466572).

[22] Sun, C.-A., Wang, G., Cai, K.-Y., Chen, T. Y. Distribution-aware mutation analysis [C]//Proceedings of 9th IEEE International Workshop on Software Cybernetics (IWSC'12), 2012: 170-175.

[23] Just, R., Jalali, D., Nozemtseva, L., Ernst, I M. D., Holmes, R., Fraser, G. Are mutants a valid substitute for real faults in software testing? [C] //Proceedings of the Symposium on the Foundations of Software Engineering (FSE2014), 2014: 654-665.

[24] García-Domínguez, A., Medina-Bulo, I. MuBPEL [EB/OL]. https://neptuno.uca.es/redmine/projects/sources-fm/wiki/MuBPEL.html, 2015.

[25] Mathur, A. P., Wong, W. E. An empirical comparison of data flow and mutation-based test adequacy criteria [J]. Software Testing, Verification and Reliability, 1994, 4(1):9-31.

[26] Hussain, S. Mutation clustering [D]. London, UK: King's College, 2008.

[27] King, K. N., Offutt, A. J. A FORTRAN language system for mutation based software testing [J]. Software Testing, Verification and Reliability, 1991, 21(7): 685-718.

[28] Offutt, A. J., Rothermel, G., Zapf, C. An experimental evaluation of selective mutation [C]//Proceedings of the 15th International Conference on Software Engineering (ICSE '93), 1993: 100-107.

[29] Langdon, W. B., Harnab, N., Jia, Y. Efficient multi-objective higher order mutation testing with genetic programming [J]. Journal of Systems and Software, 2010, 83(12): 2416-2430.

[30] 薛飞飞. 基于控制流的变异体精简技术与支持工具研究 [D]. 北京科技大学, 2016.

[31] Sun, C.-A., Xue, F. F., Liu, H., Zhang, X. Y. A Path-aware Approach to Mutant Reduction in Mutation Testing, Information and Software Technology, Elsevier, 2016, 81(1): 65-81.

[32] Krauser, E. W., Mathur, A. P., Rego, V. J. High performance software testing on SIMD machines [J]. IEEE Transactions on Software Engineering, 1991, 17(5): 403-423.

[33] 牟小玲. 基于扩展着色 Petri 网的服务组合测试研究 [D]. 西南大学, 2012.

[34] LEE, S., OFFUTT, J. Generating test cases for XML-based Web component interactions using mutation analysis [C]//Proceedings of the 12th Internation-al Symposium on Software Reliability Engineering (ISSRE '01), IEEE Computer Society, 2001: 200-209.

[35] Boonyakulsrirung, P., Suwannasart, T. A weak mutation testing framework for WS-BPEL [C]// Proceedings of the 8th International Joint Conference on Computer Science and Software Engineering (JCSSE'11), IEEE, 2011: 313 -318.

[36] 赵彦. 基于场景的 BPEL 测试用例自动生成技术与工具研究 [D]. 北京科技大学, 2014.

[37] Domínguez-Jiménez, J. J., Estero-Botaro, A., García-Domínguez, A., et al. GAmera: an automatic mutant generation system for WS-BPEL compositions [C]//Proceedings of the 7th European Conference on Web

Services (ECOWS'09), IEEE, 2009: 97-106.

[38] Boonyakulsrirung, P., Suwannasart, T. WeMuTe - a weak mutation testing tool for WS-BPEL [C]//Proceedings of The International Multi Conference of Engineers and Computer Scientists (IMECS'12), 2012: 810-815.

[39] Polo, M., Piattini, M., Garc á-Rodr guez, I. Decreasing the cost of mutation testing with second-order mutants [J]. Software Testing, Verification and Reliability, 2009, 19(2): 111-131.

[40] dom4j 2.0 [EB/OL]. http://dom4j.sourceforge.net/, 2010.

[41] Active VOS. Car repair estimation [EB/OL]. http://www.activevos.com/developers/sample-apps, 2012.

[42] Sun, C.-A., Khoury, E., Aiello, M. Transaction management in service-oriented systems: Requirements and a proposal [J]. IEEE Transactions on Services Computing, 2011, 4: 167-180.

[43] Elbaum, S., Malishevs, A. G., Rothermel, G. Test case prioritization: A family of empirical studies [J]. IEEE Transactions on Software Engineering, 2002, 28(10): 159-182.

[44] 陈梦云. 基于圈复杂度和调用次数的测试用例排序方法 [D]. 上海师范大学, 2015.

[45] Myers, G. J. The Art of Software Testing, second edition [M]. John Wiley and Sons, 2004.

作者简历及在学研究成果

一、 作者入学前简历

起止年月	学习或工作单位	备注
2010 年 09 月至 2014 年 06 月	在北京科技大学计算机科学与技术专业攻读学士学位	

二、 在学期间从事的科研工作

[1] 面向 SOA 软件的蜕变测试技术研究, 国家自然科学基金（面上项目）（61370061），主要参与人员.

[2] 高效新型的 SOA 软件测试技术与工具研究, 北京市优秀人才培养项目（2012D009006000002），主要研究人员.

三、 在学期间所获的科研奖励

[1] μBPEL：一个面向 BPEL 服务组装程序的变异测试系统, 全国软件原型竞赛二等奖, 中国计算机学会软件工程专业委员会与中国计算机学会系统软件专业委员会，2015.11.

[2] 研究生国家奖学金，北京科技大学，2016.10.

四、 在学期间发表的论文

[1] Sun, C.-A., Pan, L., Wang, Q. L., et al. An empirical study on mutation testing of WS-BPEL programs [J]. The Computer Journal, 2016.（已发表. SCI 刊源）

[2] Sun, C.-A., Zhao, Y., Pan, L., et al. Automated testing of WS-BPEL service compositions-A scenario - oriented approach [J]. IEEE Transactions on Services Computing, 2015.（已发表. SCI 刊源）

[3] Sun, C.-A., Zhao, Y., Pan, L., et al. A transformation-based approach to testing concurrent programs using UML activity diagrams [J]. Software: Practice and Experience, 2016, 46(4): 551-576.（已发表. SCI 刊源. SCI 检索号 000372324600005）

独创性说明

本人郑重声明：所呈交的论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得北京科技大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解北京科技大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵循此规定）

签名：_____ 导师签名：_____ 日期：_____

学位论文数据集

关键词*	密级*	中图分类号*	UDC	论文资助
服务组装，BPEL，变异测试，测试工具，包含关系	公开	TP311	004.41	
学位授予单位名称*		**学位授予单位代码***	**学位类别***	**学位级别***
北京科技大学		10008	工学	硕士
论文题名*		**并列题名**		**论文语种***
BPEL 程序的变异测试优化技术与集成化支持工具研究				中文
作者姓名*	潘琳		**学号***	G20148605
培养单位名称*		**培养单位代码***	**培养单位地址**	**邮编**
北京科技大学		10008	北京市海淀区学院路 30 号	100083
学科专业*		**研究方向***	**学制***	**学位授予年***
计算机技术		软件测试	2.5 年	2017
论文提交日期*	2016 年 12 月 21 日			
导师姓名*	孙昌爱		**职称***	教授
评阅人	**答辩委员会主席***		**答辩委员会成员**	
王昭顺 施小丁	王昭顺		罗熊、王卫苹	
电子版论文提交格式 文本（√） 图像（ ） 视频（ ） 音频（ ） 多媒体（ ） 其他（ ） **推荐格式：**application/msword；application/pdf				
电子版论文出版（发布）者		**电子版论文出版（发布）地**		**权限声明**
论文总页数*	59 页			
共 33 项，其中带*为必填数据，为 22 项。				