|  |
| --- |
|  |
| 面向BPEL程序的变异测试技术与支持工具研究 |
| 技术报告 |
| 第1版 |
| **王巧玲** |
| **2014/06/30** |

|  |
| --- |
|  |

[1 课题背景 1](#_Toc397415444)

[2 相关标准和技术 2](#_Toc397415445)

[2.1 BPEL 2](#_Toc397415446)

[2.2 变异测试 2](#_Toc397415447)

[2.3 面向BPEL2.0的变异算子 3](#_Toc397415448)

[2.4 变异算子之间的包含关系 5](#_Toc397415449)

[3 面向BPEL程序的变异体自动生成系统 7](#_Toc397415450)

[3.1 需求分析 7](#_Toc397415451)

[3.2 系统结构 7](#_Toc397415452)

[3.3 系统设计的关键问题 8](#_Toc397415453)

[3.3.1 变异算子管理模块 8](#_Toc397415454)

[3.3.2 BPEL程序解析模块 9](#_Toc397415455)

[3.3.3变异体生成模块 9](#_Toc397415456)

[3.3.4 XML读写模块 9](#_Toc397415457)

[3.4 系统实现 9](#_Toc397415458)

[3.4.1 About界面 9](#_Toc397415459)

[3.4.2 Generate界面 10](#_Toc397415460)

[3.4.2 File界面 11](#_Toc397415461)

[3.4.2 Help界面 12](#_Toc397415462)

[4 面向BPEL2.0变异算子关系研究 14](#_Toc397415463)

[4.1 实例SmartShelf的变异测试 14](#_Toc397415464)

[4.1.1 待测程序 14](#_Toc397415465)

[4.1.2 测试用例设计 15](#_Toc397415466)

[4.1.3 变异体生成 15](#_Toc397415467)

[4.1.4 测试结果 16](#_Toc397415468)

[4.2 实例SupplyChain的变异测试 18](#_Toc397415469)

[4.2.1 待测程序 18](#_Toc397415470)

[4.2.2 测试用例设计 19](#_Toc397415471)

[4.2.3 变异体生成 19](#_Toc397415472)

[4.2.4 测试结果 20](#_Toc397415473)

[4.3 实例SupplyCustomer的变异测试 21](#_Toc397415474)

[4.3.1 待测程序 21](#_Toc397415475)

[4.3.2 测试用例设计 22](#_Toc397415476)

[4.3.3 变异体生成 22](#_Toc397415477)

[4.3.4 测试结果 22](#_Toc397415478)

[4.4实例loan\_approval的变异测试 24](#_Toc397415479)

[4.4.1 待测程序 24](#_Toc397415480)

[4.4.2 测试用例设计 24](#_Toc397415481)

[4.4.3 变异体生成 24](#_Toc397415482)

[4.4.4 测试结果 25](#_Toc397415483)

[4.5实例CarEstimate的变异测试 26](#_Toc397415484)

[4.5.1 待测程序 26](#_Toc397415485)

[4.5.2 变异体生成 27](#_Toc397415486)

[4.5.3 测试结果 27](#_Toc397415487)

[4.6实例TravelAgency的变异测试 28](#_Toc397415488)

[4.6.1 待测程序 29](#_Toc397415489)

[4.6.2 变异体生成 29](#_Toc397415490)

[4.6.3 测试结果 30](#_Toc397415491)

[4.7面向BPEL2.0变异算子关系总结 32](#_Toc397415492)

[5总结与未来工作 33](#_Toc397415493)

[6参考文献 34](#_Toc397415494)

# 1 课题背景

近年来，网络已经在日常生活中被普遍使用，企业为了能够在网络上提供服务，必须要将信息系统的整合到一起。因为这一趋势，难以整合的传统软件逐渐无法满足需求，必须进行改变或替换，而目前逐渐开始被采用的Web服务则被视为一种适用于分散式环境的软件组件。Web服务是解决应用程序之间相互通信的一项技术。严格地说，Web服务是描述一系列操作的接口，它使用标准的、规范的XML描述接口[1]。这一描述中包括与服务进行交互所需要的全部细节，包括服务位置、消息格式和传输协议。而对外的接口中则隐藏了服务实现的细节，仅仅提供一系列可执行的操作，这些操作独立于软、硬件平台和编写服务所采用的编程语言。Web服务既可以单独使用，也可与其他的Web服务一起，实现更加复杂的业务功能。当前有许多研究探讨如何能够有效地组合Web服务[2]，提供更丰富的功能集，其中有一个被广泛采用的技术，就是通过业务流程执行语言来对Web服务之间的互动关系进行描述，如BPEL [3] （Business Process Execution Language），它可以将多个相互交互的Web服务编排在一起形成业务流程，并作为一种更为复杂的Web服务对外提供。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，在实践中，软件测试是一种保证软件可靠性的重要手段 [4,5]。

在诸多软件测试技术中，变异测试[6]技术是其中一种行之有效的测试方法。该方法将着重点放在软件中可能会出现的错误上，与传统的强调整个软件的测试方法相比，具有更强的针对性和高效性。变异测试关键的一步就是利用变异算子生成变异体，由Estero-Botaro等人提出了面向BPEL 2.0的26种变异算子[9,10,11]，这些变异算子将会产生大量的变异体，由于该项工作繁琐耗时，课题组提出开发面向BPEL2.0的变异体自动生成系统，对变异体的生成节约了资源，但是生成的大量变异体需要消耗的资源存储，对于这些变异体的测试会消耗大量的资源，在本课题中，将会对BPEL2.0的变异体进行精简，使得BPEL2.0的变异测试能更加快捷的完成。

# 2 相关标准和技术

这一节中将对课题所涉及的相关标准和技术进行简要介绍。主要涉及四个部分，一是BPEL语言，二是变异测试，三是面向BPEL2.0的变异算子，四是变异算子之间的包含关系说明。

## 2.1 BPEL

Web服务实现了对业务逻辑的封装，并以独立于操作系统和开发平台的方式向外提供调用接口。但是，单独的Web服务往往功能有限，无法满足实际的业务需求。因而，将多个独立的Web服务组合起来完成更大的业务需求的开发方式，成为构建SOA软件的一种主流方式。

BPEL是一种可执行的服务组装语言。它从工作流的角度描述了如何将多个Web服务组合成复杂的业务流程。采用BPEL描述的组合服务（即BPEL程序）能够以Web服务的形式向外提供功能。通过这种方式，BPEL程序可以参与到更高层次的业务流程当中，从而实现服务复用。

BPEL程序中的基本成分是活动（Activity）。活动又分为基本活动和结构性活动。一个基本活动表示业务流程中的一个步骤。基本活动包括assign、invoke、receive、reply、throw、wait、empty等。结构性活动用来协调和控制流程的执行过程包括sequence、switch、while、flow、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。与传统的程序设计语言类似的是，BPEL程序包括顺序、循环、分支等控制结构；不同的是，BPEL还包含并行的控制结构，以及同步和异步两种Web服务调用方式。另外，BPEL程序采用XML表示形式，语句格式与传统的编程语言具有较大区别。最后，BPEL程序中集成的Web服务可在运行时刻进行绑定。

BPEL程序的上述特性，不仅给软件测试带来了新的挑战，同时也是新型软件测试技术与工具开发的依据。

## 2.2 变异测试

变异测试（也称变异分析）[6]是一种基于故障的软件测试技术，该种测试方法通过将错误植入程序中来对程序进行测试。为了提高软件的可靠性，应尽可能地检测并消除软件中潜藏的各种故障。人们发现“一般情况下，实施充分的测试也是不可能的，而只能实施相对充分的测试”。换言之，针对某些种类的故障进行相对充分的测试是可能的，这导致了基于故障的软件测试技术的出现。

变异测试的基本思想是：将变异算子应用于待测程序从而植入错误，得到错误版本的程序集合，这种错误版本的程序被称为变异体。变异算子是对程序设计语言中典型错误的抽象和概括。每个变异体中包含了对原始程序进行的合乎语法的微小改动，从而模拟开发人员编程时所犯的错误。如果一个测试用例使得变异体的执行行为与待测程序不同，这个变异体就称为“被杀死”。

变异测试过程从解析原始程序生成变异体开始，然后用户向系统中增加测试用例，这个过程可以是人工的也可以是自动的。首先，使用这些测试用例测试原始程序，如果发现错误，则修改原来的程序，变异测试过程重新启动。如果原始程序的测试结果正确，则开始测试程序的变异体。如果在同一测试用例下，一个变异体的输出结果与原始程序不同，那么整个变异体被“杀死”，在系统中被认为已经死亡，不再参与测试的其余步骤。

一个测试用例集合能杀死所有的变异体，那么该测试用例集合相对于这些变异体是充分的，本文中采用两个指标来度量测试用例集合的充分性。

第一个度量指标为变异得分(Mutation Score)[12]，其定义为：



其中，P为被测程序，T为测试数据集合，M为变异体的数量，D为杀死的变异体数量，E为等价的变异体数量。变异得分定义了测试用例集合T能够杀死非等价变异体的比例，度量了测试用例集合的充分性，其得分越高测试的越有效。

第二个度量指标是故障检测率(Fault Discovery Rate, FDR)[12],其定义为：



其中，m为变异体，T为测试用例集合，Nf为集合T中能够杀死m变异体的数量，Nt为测试用例集合T的总数量，Ni为非法测试用例数量。故障检测率定义了测试用例集合T能够杀死m变异体的比例，其检测率越高说明测试用例T杀死m变异体的几率越大。

近年来，人们普遍认为变异测试具有很强的故障检测能力[7]。与此同时，变异测试被广泛用来评估测试用例集完备性和某种测试技术有效性[8]。变异测试还可以对其他的测试方法进行评估，并借此评估结果可以提升测试方法的能力。

## 2.3 面向BPEL2.0的变异算子

针对BPEL特点，Estero-Botaro等人提出了面向BPEL 2.0的26种变异算子[9,10,11]，如表2-1所示。在设计变异算子时假设BPEL2.0设计人员采用了某种工具进行BPEL2.0编写，因此省略了手工编写BPEL2.0代码时可能的错误类型。按照变异常算子所作用的元素的类型划分为4类：

• 标识符替换变异算子（Identifier replacement operators）

• 表达式变异算子（Expression operators）

• 活动变异算子（Activity operators）

• 异常与事件变异算子（Exception and event operators）。

一些变异算子在每一类的内部进行定义，这样的算子由三个大写字母表示：第一个是类别标识符，后两个字母定义类内的算子。这些定义过的算子模型系统的模拟了程序员在实现一个BPEL流程时常犯的错误。

表2-1 BPEL2.0的变异算子

|  |  |  |
| --- | --- | --- |
| **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(<， >，<=， >=， =， !=)用另一个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 | |
| **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” |
| **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 | |

## 2.4 变异算子之间的包含关系

BPEL2.0有26个变异算子，这些变异算子将会产生大量的变异体，虽然本课题中会开发面向BPEL的变异体自动生成系统，对变异体的生成节约了资源，但是生成的大量变异体还是需要大量的资源存储，对于这些变异体的测试会消耗大量的资源，在本课题中，将会对BPEL2.0的变异体进行精简，使得BPEL2.0的变异测试能更加快捷的完成。

变异算子A应用到BPEL程序X上产生变异体集合M，如果给定的测试用例集合T能将变异体集合M全部杀死，那么变异算子A对于X是较易发现的，它的存在对于X的变异测试工作贡献较小，在本课题中，希望对这样的变异体进行精简，力求找到更难检测的变异体，同时检验变异体之间是否存在一定的包含关系。

本课题精简变异体的一个思想是：将变异算子提取出来，并分为A、B两组，由A组变异算子生产的变异体集合定为MA，由B组变异算子生产的变异体集合定为MB，对于任意的测试用例集合T，在能杀死MB的基础上，MA都能杀死，则MB较MA更难检测，A的变异算子可以精简。

在精简变异体的时候，需要注意的是等价变异体的存在。在每一变异体上都执行完所有的测试用例后，剩余的活变异体可以划分为两大类，一类变异体为可以杀死的变异体，但由于目前的测试数据不够完备而不能将其杀死，可以通过生成专门的测试用例来将其杀死。另一类是等价变异体。等价变异体是指该变异体在功能上等价于原始程序，也即等价变异体的输出结果总是一直和原始程序的输出结果相同，因此，根本就不可能测试用例将原始程序和等价变异体鉴别开来。等价变异体无助于提高变异分数，如果一直驻留在测试过程中将耗费巨大计算开销，因而应当今早将其检测出来，一旦识别为等价变异体剩余测试用例就不必再对其进行测试了，从而可以避免不必要的计算资源的浪费。因此，等价变异体的检测就成了变异测试中的一个重要部分。

在BPEL的变异算子中，AFP、ASF、AIS、ASI[10]可能产生等价变异体。

AFP算子将forEach活动由顺序改为并行，修改parallel属性由 “no”改为 “yes”，变异体需要更多的时间来完成活动，可能产生等价变异体。

ASF算子用flow活动替代sequence活动，如果flow活动与sequence活动的结构相同，可能产生等价变异体。

AIS算子将一个域的isolated属性改为“no”，这意味着这些变量的访问不会被保护，以防止产生没有预想到的结果，可能产生等价变异体。

ASI算子用一个sequence活动交换两个活动。如果它们之间没有时间依赖关系，它会生成等价变异体。

我们在对BPEL进行变异测试的时候，会有一部分的变异体无法杀死，这些变异的存在给测试用例的充分性评估有一定的影响，在本课题中，等价变异体的存在为探索变异体之间的包含关系带来了巨大的挑战。

# 3 面向BPEL程序的变异体自动生成系统

本章介绍我们开发的面向BPEL程序的变异体自动生成系统，将会从需求分析、系统结构、系统设计的关键问题、系统实现四个方面介绍。

## 3.1 需求分析

变异测试的一个重要组成部分是变异体的生成，BPEL的26种变异算子如果生成变异体的话，工作量非常庞大，为了更快更有效的对BPEL进行变异测试，提高BPEL的可靠性，本课题将设计并实现一个BPEL变异体自动生成平台MuBPEL。该系统提供的功能如下：

1. **BPEL流程解析：**能够读取并解析BEPL文件，找到相应变异点。
2. **变异算子管理：**通过变异点能够连接到变异算子的处理。
3. **变异体生成：**能够生成变异体。
4. **XML文件读写：**能够通过读取XML文件来完成对BPEL文件的操作。

## 3.2 系统结构

目前已经存在针对不同的编程语言的变异生成系统，本文借鉴面向Java程序的变异生成系统MuJava[13]，设计与实现了面向BPEL的变异生成系统MuBPEL。面向BPEL的变异生成系统MuBPEL在BPEL解析的基础上，找到可能的变异点并应用相应的变异算子，从而生成变异体。

图3-1示意了MuBPEL的体系结构。系统包含四个模块，即变异算子管理模块、BPEL流程解析模块、变异体生成模块以及XML读写模块。BPEL解析模块首先读入一个待测BPEL流程并对其进行解析得到该BPEL中可能存在的变异点位置，生成相应的变异算子模块；解析完成后，变异算子模块中记录了可变异的算子实例集合；变异体生成模块分别调用算子的转换规则从而产生变异体。BPEL流程解析模块和变异体生成模块依赖XML读写模块所提供的通用XML读写功能，从而实现BPEL流程的读取和变异体的输出。

待测BPEL流程图

待测BPEL流程图

变异算子管理模块

多个

变异体

变异体生成模块

XML

读写模块

Input

Output

面向BPEL的变异体自动生成系统

数据流向

依赖关系

**图3-1面向BPEL的变异体自动生成系统**

## 3.3 系统设计的关键问题

本节将会对3.2中的四个模块设计进行介绍。

### 3.3.1 变异算子管理模块

在设计与实现变异算子管理模块时，用AbstractOperator抽象类表示变异算子的父类，具有一个抽象的handle方法，由子类根据具体转换规则实现，从而产生变异体。我们分别设计ISV、EAA、EEU、ERR、ELL、ECC等26个变异算子类，继承自AbstractOperator类，每个算子类具有属性Document和Element，用来记录BPEL流程以及其中变异点的位置，并根据每个变异算子具体的转换规则实现抽象父类中的handle方法。

由于26种变异算子数目较多，实现起来比较繁琐，我们在开发的时候利用了设计模式的方法，将这26种变异算子的实现方法进行封装，这样可以实现代码的复用性，同时可使系统更易维护和修改。在本系统中算子可复用性表现为：

1. **继承Auto1类：**EAA、ERR、ELL、ECC，都是运算符之间的相互转换。
2. **继承Auto2类：**EMD、EMF，都是对时间进行改变。
3. **继承Auto3类：**AEL、AIE，都是对活动进行删除。
4. **继承Auto4类：**ACI、AFP、AIS，都是将活动属性由“yes”改为“no”。
5. **继承Auto5类：**AJC、APM、APA、XMF、XER、XEE、AMC、XMT，都是对某些元素或者处理器的定义进行删除。

### 3.3.2 BPEL程序解析模块

对待测BPEL流程进行解析，找出可以进行变异的元素位置，对于变异算子匹配的代码进行记录并生成变异算子列表。具体说来，BPEL流程解析模块中包含一个Parser类，主要用于对BPEL进行解析。Parser类首先调用XML读写模块中的XMLHelper类的openXMLFile方法，读取一个BPEL流程，之后开始解析。Parser中包含一个变异算子列表，在对BPEL流程进行解析的时候，每当找到一个可能出现错误的变异点，就实例化一个相应错误类型的变异算子类，加入到变异算子列表中。当对BPEL的解析结束后，就得到了针对该BPEL流程的全部变异算子。

### 3.3.3变异体生成模块

根据已得到的变异算子列表，调用算子的转换规则生成相应的变异体。变异生成模块包含一个Generator类，实现根据变异算子列表生成变异体的功能。Generator类调用每个变异算子类的handle方法，根据变异算子的转换规则生成变异体。然后调用XML读写模块，将变异体作为文件写到磁盘上，得到错误版本的BPEL流程。重复这个过程，就可以根据变异算子列表得到多个变异体。由IDGenerator类产生变异体标号，从0开始，使用静态变量id将会方便我们定为等价变异体。

### 3.3.4 XML读写模块

将BPEL流程的读取到内存中形成文档结构树，以及将内存中的BPEL树结构写入到磁盘中。

## 3.4 系统实现

基于Java及相关技术[5]，我们实现了面向BPEL 的自动变异生成系统MuBPEL，采用DOM4J实现BPEL流程的解析和读写、采用Swing实现工具的图形化界面。下面我们利用一个实例来演示系统的实现。

### 3.4.1 About界面

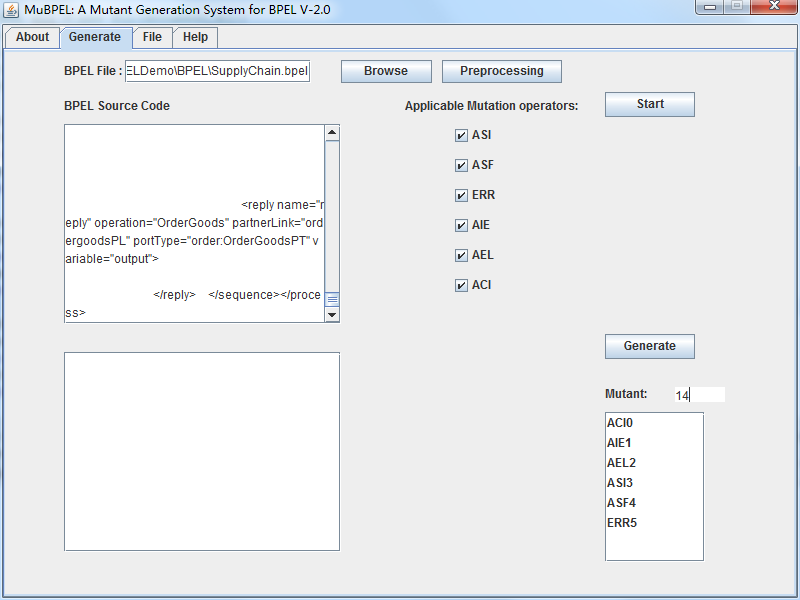
About界面主要是对BPEL、变异测试、设计模式、系统四个方面的介绍，如图3-2所示。



**图3-2 About界面**

### 3.4.2 Generate界面

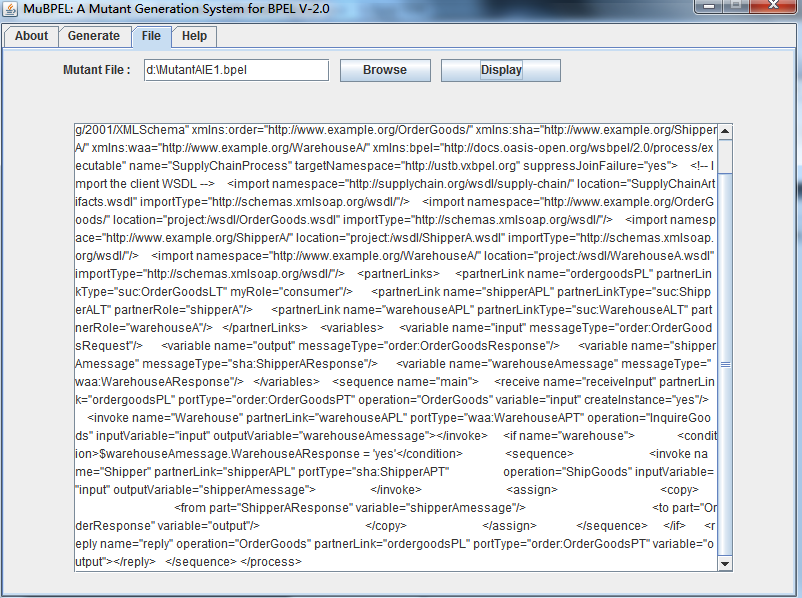
Generate界面是系统的主界面，如图3-3所示，点击Browse按钮实现选择BPEL源程序；Preprocessing按钮实现把选择的BPEL源程序显示在textArea1中；start按钮实现生成变异算子列表，并且显示在界面上；Generate按钮实现根据变异算子列表生成变异体；并且可以选择列表显示变异算子。



**图3-3 Generate界面**

### 3.4.2 File界面

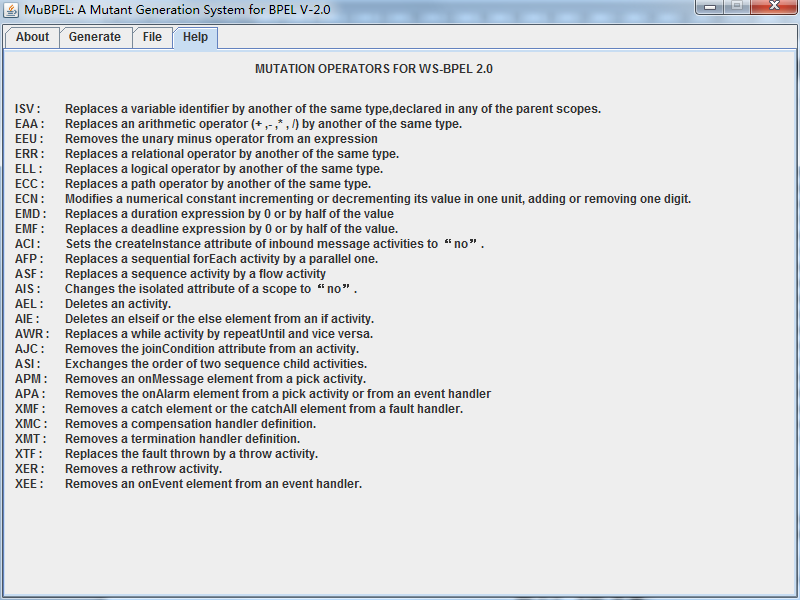
File界面选择生成变异体的文件路径，并且显示在界面上，如图3-4所示。



**图3-4 File界面**

### 3.4.2 Help界面

Help界面显示面向BPEL2.0的26种变异算子，如图3-5所示。



**图3-5 Help界面**

# 4 面向BPEL2.0变异算子关系研究

本章利用已开发的面向BPEL程序的变异体自动生成系统MuBPEL，对实例BPEL生成变异体，并对其进行变异测试，通过测试结果得出各变异算子的检验程度，对比各实例结果，找出面向BPEL2.0变异算子之间的包含关系。

## 4.1 实例SmartShelf的变异测试

实例研究对SmartShelf这个实例的原始程序和使用MuBPEL产生的变异体分别进行测试。

### 4.1.1 待测程序

SmartShelf例子是一个货架商品管理的BPEL例子，涉及14个Web服务，业务流程比较复杂。利用变异测试技术，找到变异算子之间的关系。图4-1描述了其执行流程。

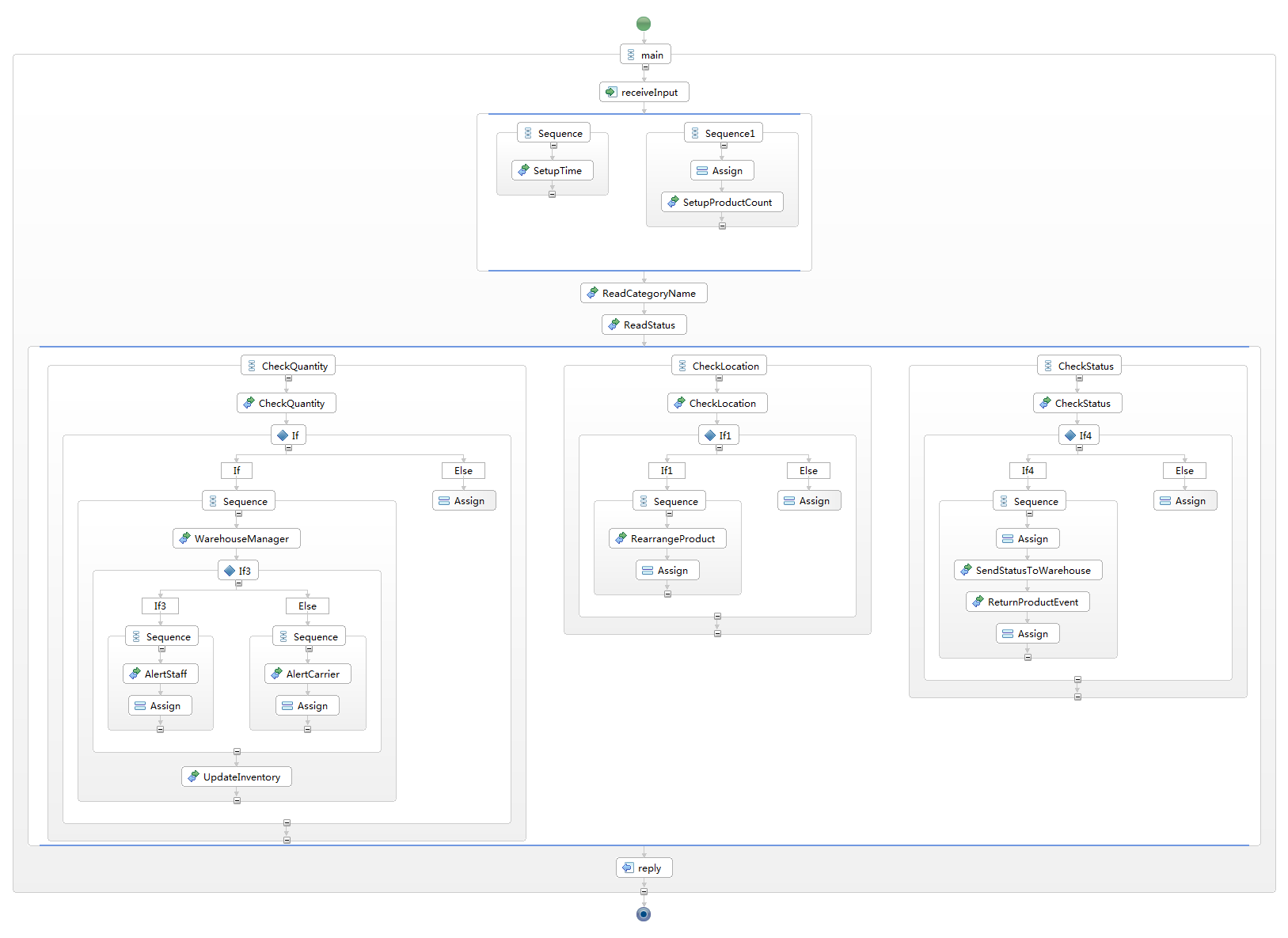


图 4-1 SmartShelf执行流程

SmartShelf货架商品管理的输入输出信息见表4-1所示，用户输入需要商品的名称和数量，管理系统查询商品信息后，给用户反馈商品的数量、位置和状态三条信息。

表4-1 SmartShelf的输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **变量** | **名称** | **类型** | **变量值** | **说明** |
| 输入变量 | name | string | ---- | 商品需求名称 |
| amount | int | ---- | 商品需求数量 |
| 输出变量 | quantity | string | Quantity is enough. | 货架商品数量可供应需求 |
| Warehouse Quantity is enough. | 仓库商品数量可供应需求 |
| Warehouse Quantity is not enough. | 仓库商品数量供不应求 |
| location | string | Location is ok. | 货架商品排放正确 |
| Rearrange is done. | 货架商品排放错误 |
| status | string | Status is ok. | 商品处于正常状态 |
| Status is not overdue. | 商品处于过期状态 |

### 4.1.2 测试用例设计

由图4-1与表4-1可以得出，SmartShelf一共有12条路径，为方便测试，我们为商品储存信息设计了一个数据库，具体信息见表4-2。

表4-2 SmartShelf的数据库

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **name** | **amount** | **Warehouse** | **status** | **location** | **备注** |
| 1 | cookie | 100 | 200 | 2015-05-09 | 1 | 正常商品 |
| 2 | milk | 100 | 0 | 2013-09-09 | 2 | 过期商品 |
| 3 | coca | 0 | 100 | 2015-04-01 | 2 | 商品售罄 |
| 4 | candy | 100 | 300 | 2015-07-15 | 5 | 货架错误 |

结合表4-1与表4-2，通过边界值与等价类划分的方法，设计了TXTYTZ三组测试用例集合，其中TX有22个测试用例，TY有35个测试用例，TZ有50个测试用例。

### 4.1.3 变异体生成

利用3.4设计的MuBPEL对SmartShelf生成变异体，共生成了60个变异体，

其中涉及到的变异算子有8种，如表4-3所示。

表4-3 SmartShelf的变异算子

|  |  |  |
| --- | --- | --- |
| **类型** | **个数** | **所占百分比** |
| ACI | 1 | 1.7% |
| AEL | 9 | 15% |
| AIE | 4 | 6.7% |
| ASI | 10 | 16.7% |
| ASF | 11 | 18.3% |
| ERR | 20 | 33.3% |
| EAA | 4 | 6.7% |
| EEU | 1 | 1.7% |

### 4.1.4 测试结果

利用4.1.2的TXTYTZ三组测试用例集合对SmartShelf原始程序和变异体进行变异测试，其中原始程序运行结果为预期测试结果，对比预期结果与实际结果，查看变异体是否被杀死，根据得到的数据，计算出测试用例集合的故障检测率，其结果如表4-3所示。

表4-4 SmartShelf故障检测率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **变异体** | **TX（22）** | **TY（35）** | **TZ（50）** |
| ACI | MutantACI1 | 100% | 100% | 100% |
| AEL | MutantAEL6 | 100% | 100% | 100% |
| MutantAEL23 | 18.2% | 20% | 28% |
| MutantAEL27 | 40.9% | 37.1% | 36% |
| MutantAEL29 | 40.9% | 40% | 36% |
| MutantAEL39 | 31.8% | 28.6% | 28% |
| MutantAEL41 | 68.2% | 71.4% | 72% |
| MutantAEL51 | 18.2% | 20% | 22% |
| MutantAEL52 | 18.2% | 20% | 22% |
| MutantAEL54 | 81.9% | 80% | 80% |
| AIE | MutantAIE24 | 40.9% | 37.1% | 36% |
| MutantAIE28 | 40.9% | 40% | 36% |
| MutantAIE40 | 68.2% | 71.4% | 72% |
| MutantAIE53 | 81.9% | 80% | 80% |
| ASI | MutantASI2 | 100% | 100% | 100% |
| MutantASI5 | 100% | 100% | 100% |
| MutantASI8 | 100% | 100% | 100% |
| MutantASI15 | 100% | 100% | 100% |
| MutantASI22 | 0 | 0 | 0 |
| MutantASI26 | 0 | 0 | 0 |
| MutantASI31 | 100% | 100% | 100% |
| MutantASI38 | 0 | 0 | 0 |
| MutantASI43 | 100% | 100% | 100% |
| MutantASI50 | 18.2% | 20% | 22% |
| ASF | MutantASF0 | 100% | 100% | 100% |
| MutantASF3 | 0 | 0 | 0 |
| MutantASF4 | 100% | 100% | 100% |
| MutantASF7 | 100% | 100% | 100% |
| MutantASF14 | 100% | 100% | 100% |
| MutantASF21 | 0 | 0 | 0 |
| MutantASF25 | 0 | 0 | 0 |
| MutantASF30 | 100% | 100% | 100% |
| MutantASF37 | 0 | 0 | 0 |
| MutantASF42 | 100% | 100% | 100% |
| MutantASF49 | 18.2% | 20% | 22% |
| ERR | MutantERR9 | 13.6% | 8.6% | 6% |
| MutantERR10 | 86.4% | 91.4% | 94% |
| MutantERR11 | 100% | 100% | 100% |
| MutantERR12 | 72.7% | 68.6% | 70% |
| MutantERR13 | 59.1% | 60% | 64% |
| MutantERR16 | 13.6% | 8.6% | 6% |
| MutantERR17 | 40.9% | 42.9% | 48% |
| MutantERR18 | 59.1% | 54.3% | 56% |
| MutantERR19 | 40.9% | 31.2% | 28% |
| MutantERR20 | 18.2% | 22.9% | 28% |
| MutantERR32 | 100% | 100% | 100% |
| MutantERR33 | 31.8% | 28.6% | 28% |
| MutantERR34 | 100% | 100% | 100% |
| MutantERR35 | 0 | 0 | 0 |
| MutantERR36 | 68.2% | 71.4% | 72% |
| MutantERR44 | 0 | 0 | 0 |
| MutantERR45 | 100% | 100% | 100% |
| MutantERR46 | 100% | 100% | 100% |
| MutantERR47 | 18.2% | 20% | 22% |
| MutantERR48 | 81.9% | 80% | 80% |
| EAA | MutantEAA55 | 63.6% | 62.9% | 68% |
| MutantEAA56 | 63.6% | 62.9% | 68% |
| MutantEAA57 | 63.6% | 62.9% | 68% |
| MutantEAA58 | 63.6% | 62.9% | 68% |
| EEU | MutantEEU59 | 63.6% | 62.9% | 68% |

根据实验的数据，得出以下结论：

1. 对于ACI变异算子，MACI故障检测率为100%，说明此类型的变异体很容易被杀死。
2. 对于AEL和AIE变异算子，MAEL中的{MutantAEL27, MutantAEL 29, MutantAEL 41, MutantAEL 54}与MAIE的故障检测率相同，对于测试用例集合TXTYTZ，在能杀死MAEL的基础上MAIE都能杀死，即MAEL包含MAIE，相对于AEL算子AIE可以精简。
3. 对于ASI和ASF变异算子，MASI中的{MutantASI22, MutantASI26, MutantASI 38}与MASF的{MutantASF3, MutantASF 21, MutantASF 25, MutantASF 27}为等价变异体，其他的故障检测率相同，相对于ASI算子ASF可以精简。
4. 对于ERR变异算子，MERR故障检测率变化比较大，说明此类型的变异体被测杀的概率不稳定。
5. 对于EAA算子，MEAA故障检测率变化比较小，本例原型算术表达式为两变量只差与0的比较，故MEAA与MEEU的故障检测率都相同。

在这60个变异体中，有9个变异体的故障检测率为0，其中有7个等价变异体，分析测试结果数据，可以发现三组测试用例集的变异分数一样，如表4-4。

表4-4 SmartShelf实验变异得分

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TX（22）** | **TY（35）** | **TZ（50）** |
| 变异得分 | 96.2% | 96.2% | 96.2% |

## 4.2 实例SupplyChain的变异测试

实例研究对SupplyChain这个实例的原始程序和使用MuBPEL产生的变异体分别进行测试。

### 4.2.1 待测程序

SupplyChain例子是一个供应链管理的BPEL例子，涉及3个Web服务，图4-2描述了其执行流程。

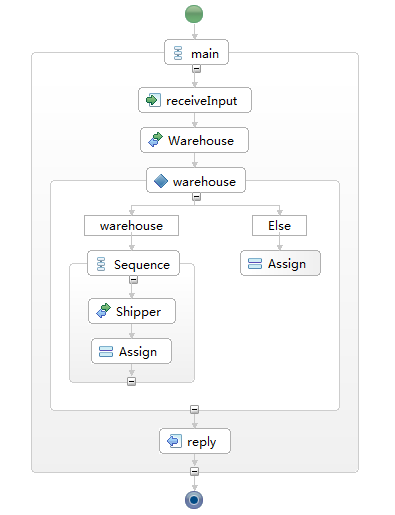


图 4-2 SupplyChain执行流程

供应链管理的输入输出信息见表4-5所示，客户输入需要商品的名称和数量，零售商根据订货单及供应商库存情况向用户反馈信息。

表4-5 SupplyChain的输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **变量** | **名称** | **类型** | **变量值** | **说明** |
| 输入变量 | name | string | ---- | 商品需求名称 |
| amount | int | ---- | 商品需求数量 |
| 输出变量 | OrderResponse | string | ---- | 供应商库存可供应需求，供应商发出可订货消息 |
| warehouseA ca not receive the bill. | 供应商库存供不应求 |

### 4.2.2 测试用例设计

由图4-2与表4-5可以得出，SupplyChain一共有2条路径，业务流程比较简单，库存的情况如表4-5所示。

表4-5 SupplyChain的数据库

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **name** | **Warehouse** | **备注** |
| 1 | cookie | 100 | 可供订货 |
| 2 | Coco | 10 | 不可订货 |
| 3 | milk | 0 | 商品售罄 |

根据库存情况，通过边界值与等价类划分的方法，设计了TXTYTZ三组测试用例集合，其中TX有6个测试用例，TY有10个测试用例，TZ有15个测试用例。

### 4.2.3 变异体生成

利用3.4设计的MuBPEL对SupplyChain生成变异体，共生成了12个变异体，

其中涉及到的变异算子有7种，如表4-6所示。

表4-6 SmartShelf的变异算子

|  |  |  |
| --- | --- | --- |
| **类型** | **个数** | **所占百分比** |
| ACI | 1 |  |
| AEL | 2 |  |
| AIE | 1 |  |
| ASI | 2 |  |
| ASF | 2 |  |
| ERR | 5 |  |
| ELL | 1 |  |

### 4.2.4 测试结果

利用4.2.2的TXTYTZ三组测试用例集合对SupplyChain原始程序和变异体进行变异测试，其中原始程序运行结果为预期测试结果，对比预期结果与实际结果，查看变异体是否被杀死，根据得到的数据，计算出测试用例集合的故障检测率，其结果如表4-7所示。

表4-7 SupplyChain故障检测率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **变异体** | **TX（6）** | **TY（10）** | **TZ（15）** |
| ACI | MutantACI1 | 100% | 100% | 100% |
| AEL | MutantAEL10 | 50% | 40% | 40% |
| MutantAEL12 | 50% | 60% | 60% |
| AIE | MutantAIE11 | 50% | 60% | 60% |
| ASI | MutantASI2 | 100% | 100% | 100% |
| MutantASI9 | 50% | 40% | 40% |
| ASF | MutantASF0 | 100% | 100% | 100% |
| MutantASF8 | 50% | 40% | 40% |
| ERR | MutantERR3 | 33.3% | 30% | 33.3% |
| MutantERR4 | 66.7% | 70% | 66.7% |
| MutantERR5 | 100% | 100% | 100% |
| MutantERR6 | 83.3% | 90% | 93.3% |
| MutantERR7 | 16.7% | 10% | 6.7% |
| ELL | MutantELL13 | 16.7% | 20% | 20% |

根据实验的数据，得出以下结论：

1. 对于ACI变异算子，MACI故障检测率为100%，说明此类型的变异体很容易被杀死。
2. 对于AEL和AIE变异算子，MAEL中的{MutantAEL12, }与MAIE的故障检测率相同，对于测试用例集合TXTYTZ，在能杀死MAEL的基础上MAIE都能杀死，即MAEL包含MAIE，相对于AEL算子AIE可以精简。
3. 对于ASI和ASF变异算子，MASI与MASF的故障检测率相同。
4. 对于ERR变异算子，MERR故障检测率变化比较大，说明此类型的变异体被测杀的概率不稳定。
5. 对于ELL变异算子，MELL故障检测率变化比较小，说明此类型的变异体被测杀的概率比较稳定。

用TXTYTZ三组测试用例可以杀死全部的变异体，其变异得分如表4-8所示。

表4-8 SupplyChain实验变异得分

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TX（6）** | **TY（10）** | **TZ（15）** |
| 变异得分 | 100% | 100% | 100% |

## 4.3 实例SupplyCustomer的变异测试

实例研究对SupplyCustomer这个实例的原始程序和使用MuBPEL产生的变异体分别进行测试。

### 4.3.1 待测程序

SupplyCustomer例子是一个项目订单管理的例子，涉及6个Web服务，图4-3描述了其执行流程。

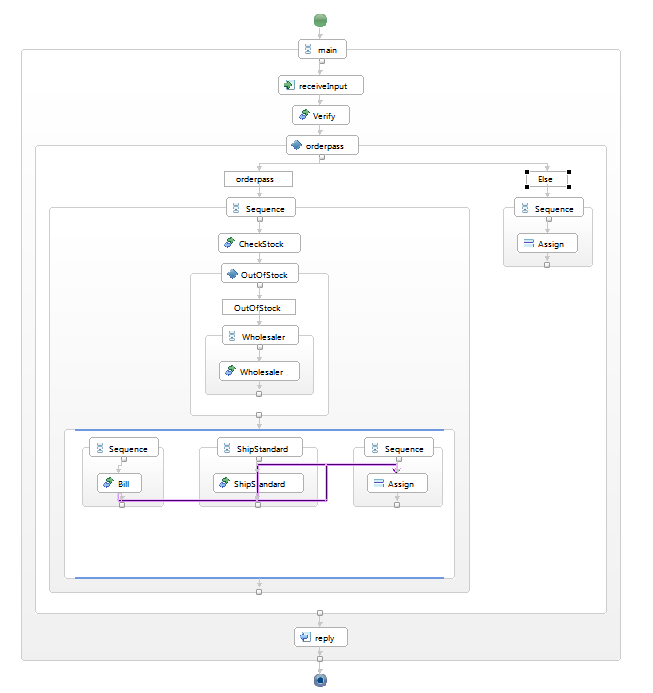


图 4-3 SupplyCustomer执行流程

项目订单管理的输入输出信息见表4-9所示，客户输入订单的信息和地址，系统通过验证之后向客户反馈信息。

表4-9 SupplyCustomer的输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **变量** | **名称** | **类型** | **变量值** | **说明** |
| 输入变量 | items | string | ---- | 项目订货信息 |
| address | string | ---- | 项目订货地址 |
| 输出变量 | SendOrderResponse | string | yes | 订单检查通过 |
| Order Checked Failure | 订单检查失败 |

### 4.3.2 测试用例设计

由图4-3可以得出，SupplyCustomer一共有3条路径，业务流程比较简单，我们采用表4-5的库存，根据库存情况，通过边界值与等价类划分的方法，设计了TXTYTZ三组测试用例集合，其中TX有7个测试用例，TY有12个测试用例，TZ有18个测试用例。

### 4.3.3 变异体生成

利用3.4设计的MuBPEL对SupplyCustomer生成变异体，共生成了25个变异体，其中涉及到的变异算子有8种，如表4-10所示。

表4-10 SmartShelf的变异算子

|  |  |  |
| --- | --- | --- |
| **类型** | **个数** | **所占百分比** |
| ACI | 1 | 4% |
| AEL | 2 | 8% |
| AIE | 1 | 4% |
| ASI | 2 | 8% |
| ASF | 7 | 28% |
| ERR | 10 | 40% |
| ELL | 1 | 8% |
| AJC | 1 | 8% |

### 4.3.4 测试结果

利用4.3.2的TXTYTZ三组测试用例集合对SupplyCustomer原始程序和变异体进行变异测试，其中原始程序运行结果为预期测试结果，对比预期结果与实际结果，查看变异体是否被杀死，根据得到的数据，计算出测试用例集合的故障检测率，其结果如表4-11所示。

表4-11 SupplyCustomer故障检测率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **变异体** | **TX（7）** | **TY（12）** | **TZ（18）** |
| ACI | MutantACI1 | 100% | 100% | 100% |
| AEL | MutantAEL19 | 42.9% | 33.3% | 33.3% |
| MutantAEL22 | 57.1% | 66.7% | 66.7% |
| AIE | MutantAIE20 | 57.1% | 66.7% | 66.7% |
| ASI | MutantASI2 | 100% | 100% | 100% |
| MutantASI9 | 85.7% | 83.3% | 83.3% |
| ASF | MutantASF0 | 100% | 100% | 100% |
| MutantASF8 | 85.7% | 83.3% | 83.3% |
| MutantASF15 | 0 | 0 | 0 |
| MutantASF16 | 0 | 0 | 0 |
| MutantASF17 | 0 | 0 | 0 |
| MutantASF18 | 0 | 0 | 0 |
| MutantASF21 | 0 | 0 | 0 |
| ERR | MutantERR3 | 42.9% | 33.3% | 33.3% |
| MutantERR4 | 42.9% | 33.3% | 33.3% |
| MutantERR5 | 0 | 0 | 0 |
| MutantERR6 | 42.9% | 33.3% | 33.3% |
| MutantERR7 | 0 | 0 | 0 |
| MutantERR10 | 85.7% | 83.3% | 83.3% |
| MutantERR11 | 85.7% | 83.3% | 83.3% |
| MutantERR12 | 42.9% | 50% | 50% |
| MutantERR13 | 42.9% | 33.3% | 33.3% |
| MutantERR14 | 0 | 0 | 0 |
| ELL | MutantELL23 | 0 | 0 | 0 |
| AJC | MutantAJC24 | 0 | 0 | 0 |

根据实验的数据，得出以下结论：

1. 对于ACI变异算子，MACI故障检测率为100%，说明此类型的变异体很容易被杀死。
2. 对于AEL和AIE变异算子，MAEL中的{MutantAEL22}与MAIE的故障检测率相同，对于测试用例集合TXTYTZ，在能杀死MAEL的基础上MAIE都能杀死，即MAEL包含MAIE，相对于AEL算子AIE可以精简。
3. MASF的{MutantASF15, MutantASF16, MutantASF17, MutantASF18, MutantASF21}为等价变异体
4. MASF的{MutantASF0, MutantASF8}与MASI的故障检测率相同，相对于ASI算子ASF可以精简。
5. 对于ERR变异算子，MERR故障检测率变化比较大，说明此类型的变异体被测杀的概率不稳定。
6. 对于ELL和AJC变异算子，故障检测率为0 ，说明此类型的变异体不易被检测。

在这25个变异体中，有10个变异体的故障检测率为0，其中有5个等价变异体，分析测试结果数据，可以发现三组测试用例集的变异分数一样，如表4-12。

表4-12 SupplyCustomer实验变异得分

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TX（7）** | **TY（12）** | **TZ（18）** |
| 变异得分 | 75% | 75% | 75% |

## 4.4实例loan\_approval的变异测试

实例研究对loan\_approval这个实例的原始程序和使用MuBPEL产生的变异体分别进行测试。

### 4.4.1 待测程序

loan\_approval例子是一个贷款审批的例子，涉及6个Web服务，图4-4描述了其执行流程。客户输入自己的个人信息和贷款数目，贷款审批执行审批流程后，返回贷款成功或贷款失败。

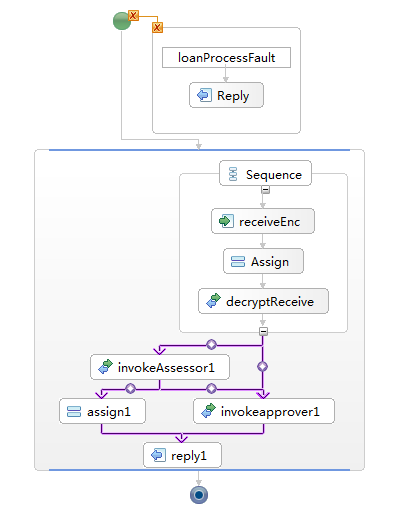


图 4-4 loan\_approval 执行流程

### 4.4.2 测试用例设计

由图4-4可以得出，loan\_approval一共有3条路径，通过银行人员信息信息情况，设计了TXTYTZ三组测试用例集合，其中TX有12个测试用例，TY有18个测试用例，TZ有25个测试用例。

### 4.4.3 变异体生成

利用3.4设计的MuBPEL对loan\_approval生成变异体，共生成了24个变异体，其中涉及到的变异算子有7种，如表4-13所示。

表4-13 loan\_approval的变异算子

|  |  |  |
| --- | --- | --- |
| **类型** | **个数** | **所占百分比** |
| ACI | 1 | 4.2% |
| AEL | 2 | 8.3% |
| ASI | 1 | 4.2% |
| ASF | 1 | 4.2% |
| ECN | 8 | 33.3% |
| ERR | 10 | 41.7% |
| XMF | 1 | 4.2% |

### 4.4.4 测试结果

利用4.4.2的TXTYTZ三组测试用例集合对loan\_approval原始程序和变异体进行变异测试，其中原始程序运行结果为预期测试结果，对比预期结果与实际结果，查看变异体是否被杀死，根据得到的数据，计算出测试用例集合的故障检测率，其结果如表4-14所示。

表4-14 loan\_approval故障检测率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **变异体** | **TX（12）** | **TY（18）** | **TZ（25）** |
| ACI | MutantACI2 | 66.7% | 66.7% | 68% |
| AEL | MutantAEL4 | 66.7% | 66.7% | 68% |
| MutantAEL23 | 58.3% | 55.6% | 52% |
| ASI | MutantASI3 | 66.7% | 66.7% | 68% |
| ASF | MutantASF1 | 66.7% | 66.7% | 68% |
| ECN | MutantECN10 | 8.3% | 5.6% | 4% |
| MutantECN11 | 16.7% | 11.1% | 8% |
| MutantECN12 | 8.3% | 11.1% | 12% |
| MutantECN13 | 33.3% | 33.3% | 32% |
| MutantECN19 | 16.7% | 11.1% | 8% |
| MutantECN20 | 8.3% | 5.6% | 4% |
| MutantECN21 | 16.7% | 22.2% | 20% |
| MutantECN22 | 8.3% | 11.1% | 12% |
| ERR | MutantERR5 | 8.3% | 5.6% | 4% |
| MutantERR6 | 58.3% | 55.6% | 56% |
| MutantERR7 | 41.7% | 44.4% | 48% |
| MutantERR8 | 33.3% | 33.3% | 32% |
| MutantERR9 | 25% | 22.2% | 24% |
| MutantERR14 | 8.3% | 5.6% | 4% |
| MutantERR15 | 33.3% | 38.9% | 44% |
| MutantERR16 | 41.7% | 44.4% | 48% |
| MutantERR17 | 25% | 27.8% | 32% |
| MutantERR18 | 16.7% | 16.7% | 16% |
| XMF | MutantXMF0 | 33.3% | 33.3% | 32% |

根据实验的数据，得出以下结论：

1. 对于ACI、 XMF、AEL变异算子，故障检测率有一定的变化,说明此类型的变异体被测杀的概率不稳定。
2. MASF与MASI的故障检测率相同。
3. 对于ERR和ECN变异算子，他们的故障检测率变化比较大，MECN的故障检测率小于MERR的故障检测率，测试过程中发现：能够杀死MECN的测试用例都能杀死MERR，因此，在带有常量的关系表达式中，ERR相对于ECN是可以精简的。

用TXTYTZ三组测试用例可以杀死全部的变异体，其变异得分如表4-15所示。

表4-15 loan\_approval实验变异得分

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TX（12）** | **TY（18）** | **TZ（25）** |
| 变异得分 | 100% | 100% | 100% |

## 4.5实例CarEstimate的变异测试

实例研究对CarEstimate这个实例的原始程序和使用MuBPEL产生的变异体分别进行测试。

### 4.5.1 待测程序

CarEstimate例子是一个汽车修复评估系统，涉及8个Web服务，图4-5描述了其执行流程。顾客提供评估请求，系统首先进行初步评估，然后根据请求对汽车进行简单评估或复杂评估，最后调用最终评估将评估结果返回给顾客。

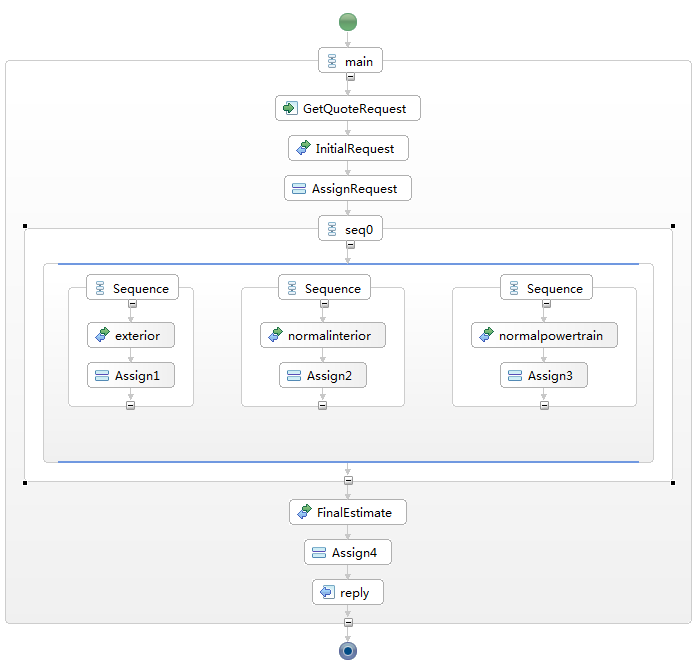


图 4-5 CarEstimate 执行流程

### 4.5.2 变异体生成

利用3.4设计的MuBPEL对CarEstimate生成变异体，共生成了15个变异体，其中涉及到的变异算子有4种，如表4-16所示。

表4-16 CarEstimate的变异算子

|  |  |  |
| --- | --- | --- |
| **类型** | **个数** | **所占百分比** |
| ACI | 1 |  |
| AEL | 5 |  |
| ASI | 4 |  |
| ASF | 5 |  |
| EMD | 2 |  |
| EMF | 2 |  |
| APM | 1 |  |
| APA | 1 |  |

### 4.5.3 测试结果

由图4-5可以得出，CarEstimate一共有3条路径，设计了TXTYTZ三组测试用例集合，其中TX有10个测试用例，TY有20个测试用例，TZ有30个测试用例。利用TXTYTZ三组测试用例集合对CarEstimate原始程序和变异体进行变异测试，其中原始程序运行结果为预期测试结果，对比预期结果与实际结果，查看变异体是否被杀死，根据得到的数据，计算出测试用例集合的故障检测率，其结果如表4-17所示。

表4-17 CarEstimate故障检测率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **变异体** | **TX（10）** | **TY（20）** | **TZ（30）** |
| ACI | MutantACI2 | 100% | 100% | 100% |
| AEL | MutantAEL3 | 100% | 100% | 100% |
| MutantAEL7 | 20% | 20% | 20% |
| MutantAEL10 | 40% | 45% | 33.3% |
| MutantAEL13 | 40% | 35% | 46.7% |
| MutantAEL14 | 100% | 100% | 100% |
| ASI | MutantASI1 | 100% | 100% | 100% |
| MutantASI6 | 20% | 20% | 20% |
| MutantASI9 | 40% | 45% | 33.3% |
| MutantASI12 | 40% | 35% | 46.7% |
| ASF | MutantASF0 | 100% | 100% | 100% |
| MutantASF4 | 0 | 0 | 0 |
| MutantASF5 | 20% | 20% | 20% |
| MutantASF8 | 40% | 45% | 33.3% |
| MutantASF11 | 40% | 35% | 46.7% |

根据实验的数据，得出以下结论：

1. 对于ACI变异算子，MACI故障检测率为100%，说明此类型的变异体很容易被杀死。
2. MAEL的故障检测率有一定的变化，说明此类型的变异体被测杀的概率不稳定。
3. 对于ASI和ASF变异算子，MASF中的{MutantASF4}为等价变异体，其他的故障检测率相同，相对于ASI算子ASF可以精简。

在15个变异体中有一个为等价变异体，其他变异体用TXTYTZ三组测试用例都可以杀死，其变异得分如表4-18所示。

表4-18 CarEstimate实验变异得分

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TX（10）** | **TY（20）** | **TZ（30）** |
| 变异得分 | 100% | 100% | 100% |

## 4.6实例TravelAgency的变异测试

实例研究对TravelAgency这个实例的原始程序和使用MuBPEL产生的变异体分别进行测试。

### 4.6.1 待测程序

TravelAgency例子是一个旅行社预订系统，涉及9个Web服务，该系统由旅行社预订、旅店预订、飞机预订和银行结算组成，图4-6描述了其执行流程。客户提供个人信息和人数，系统根据人数来选择预订服务，最后将预订信息与结算账单返回非客户。

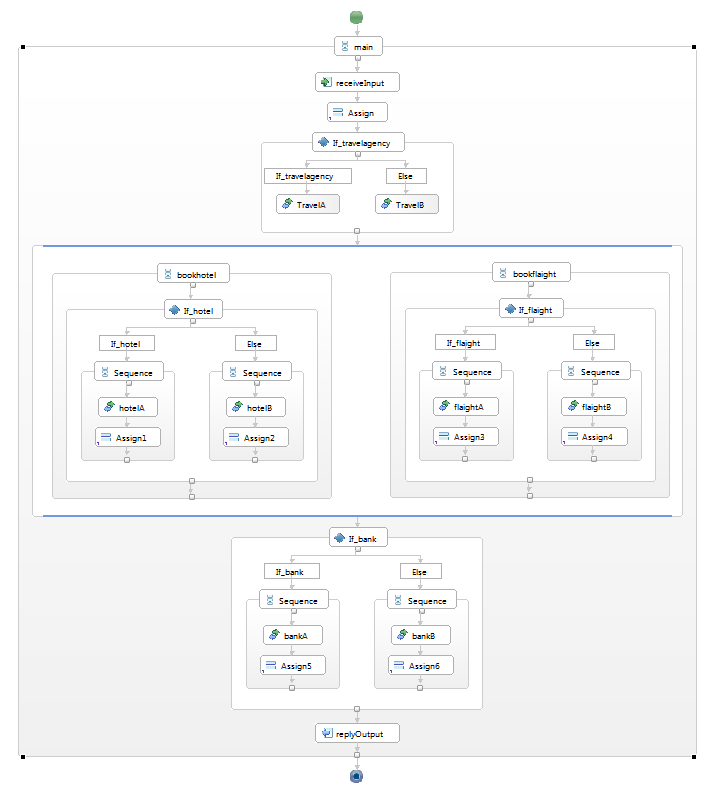


图 4-6 CarEstimate 执行流程

### 4.6.2 变异体生成

利用3.4设计的MuBPEL对TravelAgency生成变异体，共生成了43个变异体，其中涉及到的变异算子有6种，如表4-19所示。

表4-19 TravelAgency的变异算子

|  |  |  |
| --- | --- | --- |
| **类型** | **个数** | **所占百分比** |
| ACI | 1 | 2.3% |
| AEL | 7 | 16.3% |
| AIE | 5 | 11.6% |
| ASI | 4 | 9.3% |
| ASF | 6 | 14.0% |
| ERR | 20 | 46.1% |

### 4.6.3 测试结果

由于旅行社预订系统中个服务之间的绑定，TravelAgency的路径只有2条，设计了TXTYTZ三组测试用例集合，其中TX有4个测试用例，TY有7个测试用例，TZ有10个测试用例。利用TXTYTZ三组测试用例集合对TravelAgency原始程序和变异体进行变异测试，其中原始程序运行结果为预期测试结果，对比预期结果与实际结果，查看变异体是否被杀死，根据得到的数据，计算出测试用例集合的故障检测率，其结果如表4-20所示。

表4-20TravelAgency故障检测率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **变异体** | **TX（4）** | **TY（7）** | **TZ（10）** |
| ACI | MutantACI2 | 100% | 100% | 100% |
| AEL | MutantAEL9 | 50% | 57.1% | 50% |
| MutantAEL17 | 50% | 42.9% | 50% |
| MutantAEL21 | 50% | 57.1% | 50% |
| MutantAEL28 | 50% | 42.9% | 50% |
| MutantAEL32 | 50% | 57.1% | 50% |
| MutantAEL38 | 50% | 42.9% | 50% |
| MutantAEL42 | 50% | 57.1% | 50% |
| AIE | MutantAIE8 | 50% | 57.1% | 50% |
| MutantAIE16 | 50% | 42.9% | 50% |
| MutantAIE20 | 50% | 57.1% | 50% |
| MutantAIE31 | 50% | 57.1% | 50% |
| MutantAIE41 | 50% | 57.1% | 50% |
| ASI | MutantASI1 | 100% | 100% | 100% |
| MutantASI19 | 50% | 42.9% | 50% |
| MutantASI30 | 50% | 42.9% | 50% |
| MutantASI40 | 50% | 42.9% | 50% |
| ASF | MutantASF0 | 100% | 100% | 100% |
| MutantASF10 | 0 | 0 | 0 |
| MutantASF18 | 50% | 42.9% | 50% |
| MutantASF22 | 0 | 0 | 0 |
| MutantASF29 | 50% | 42.9% | 50% |
| MutantASF39 | 50% | 42.9% | 50% |
| ERR | MutantERR3 | 25% | 14.3% | 10% |
| MutantERR4 | 75% | 57.1% | 60% |
| MutantERR5 | 25% | 42.9% | 40% |
| MutantERR6 | 25% | 85.7% | 90% |
| MutantERR7 | 100% | 100% | 100% |
| MutantERR11 | 25% | 14.3% | 10% |
| MutantERR12 | 75% | 57.1% | 60% |
| MutantERR13 | 25% | 42.9% | 40% |
| MutantERR14 | 25% | 85.7% | 90% |
| MutantERR15 | 100% | 100% | 100% |
| MutantERR23 | 25% | 14.3% | 10% |
| MutantERR24 | 75% | 57.1% | 60% |
| MutantERR25 | 25% | 42.9% | 40% |
| MutantERR26 | 25% | 85.7% | 90% |
| MutantERR27 | 100% | 100% | 100% |
| MutantERR33 | 25% | 14.3% | 10% |
| MutantERR34 | 75% | 57.1% | 60% |
| MutantERR35 | 25% | 42.9% | 40% |
| MutantERR36 | 25% | 85.7% | 90% |
| MutantERR37 | 100% | 100% | 100% |

根据测试结果及实验的数据，得出以下结论：

1. 对于ACI变异算子，MACI故障检测率为100%，说明此类型的变异体很容易被杀死。
2. 对于AEL和AIE变异算子，MAEL中的{MutantAEL9, MutantAEL 17, MutantAEL 21, MutantAEL 32，MutantAEL 42}与MAIE的故障检测率相同，对于测试用例集合TXTYTZ，在能杀死MAEL的基础上MAIE都能杀死，即MAEL包含MAIE，相对于AEL算子AIE可以精简。
3. 对于ASI和ASF变异算子， MASF的{MutantASF10, MutantASF22}为等价变异体，其他的故障检测率相同，相对于ASI算子ASF可以精简。
4. 对于ERR变异算子，MERR故障检测率变化比较大，说明此类型的变异体被测杀的概率不稳定。

在43个变异体中有一个为等价变异体，其他变异体用TXTYTZ三组测试用例都可以杀死，其变异得分如表4-21所示。

表4-21 TravelAgency实验变异得分

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TX（10）** | **TY（20）** | **TZ（30）** |
| 变异得分 | 100% | 100% | 100% |

## 4.7面向BPEL2.0变异算子关系总结

# 5总结与未来工作

变异测试对BPEL的可靠性保证是一种重要手段，但是面向BPEL2.0有26种变异算子，这些变异算子将会产生大量的变异体，这些对于变异测试都会带来很大的工作量。在生成变异体阶段我们开发了面向BPEL2.0的变异体自动生成系统MuBPEL，它将会对给定的BPEL原程序自动的生成变异体集M，我们利用测试集合T对其进行测试，通过故障检测率来观察变异算子的检测程度。

我们通过对SmartShelf实例的变异测试结果得出：相对于AEL算子AIE可以精简；相对于ASI算子ASF可以精简。即在SmartShelf中，在AEL算子和ASI算子都能检测的情况下，AIE与ASF也能检测出来。

在未来的工作中，我们将对更多的BPEL实例进行变异测试，找到它们可以精简的变异算子，并证明这些算子对其它BPEL程序的同样可以精简。

# 6参考文献

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(6).
2. Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition[J]. IEEE Internet Computing, 2004, pp: 51-59.
3. OASIS, 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, 2007.
4. 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(5).
5. 孙喜龙.基于BPEL的Web服务组合测试研究,硕士学位论文.
6. Richard A.DeMillo, Richard J. Lipton and Frederick G.Sayward. Hints on test data selection: Help for the practicing programmer [J]. IEEE Computer, 1978(1), pp: 31-41.
7. Jia Yue, Harman Mark. An analysis and survey of the development of mutatio testing. IEEE Transactions on Software Engineering, 2011, 37(5): 649-678.
8. Sun Chang-ai, Wang Guan, Cai Kai-Yuan, Chen Tsong-Yueh. Distribution-aware Mutation Analysis. Proceedings of 9th IEEE International Workshop on Software Cybernetics (IWSC 2012), collocated with 36th Annual IEEE International Computer Software and Application Conference (COMPSAC 2012), IEEE Computer Society, July 16-July 20, 2012, Izmir, Turkey.
9. Boubeta-Puig Juan, Medina-Bulo Inmaculada, and García-Domínguez Antonio. Analogies and Differences between Mutation Operators for WS-BPEL 2.0 and Other Languages. Proceedings of 6th International Workshop on Mutation Analysis, 2011: 398-407.
10. Estero-Botaro Antonia, Palomo-Lozano Francisco, and Medina-Bulo Inmaculada. Mutation Operators for WS-BPEL 2.0. Proceedings of ICSSEA 2008-2 Estero Palomo Medina, 2008: 1-7.
11. Estero-Botaro Antonia, Palomo-Lozano Francisco, and Medina-Bulo Inmaculada. Quantitative Evaluation of Mutation Operators for WS-BPEL Compositions. Proceedings of Third International Conference on Software Testing, Verification, and Validation Workshops, 2010: 142-150.
12. Richard A.DeMillo, Richard J. Lipton and Frederick G.Sayward. Hints on test data selection: Help for the practicing programmer [J]. IEEE Computer, 1978(1), pp: 31-41.
13. Ma Yu-Seung, Offutt Jeff, and Kwon Yong-Rae. MμJava: A Mutation System for Java. Proceedings of the 28th international conference on Software engineering (ICSE’ 06), 2006: 827-830.