[1 基于切片的故障定位技术 3](#_Toc471204898)

[1.1 故障定位的原理 3](#_Toc471204899)

[1.2 相关概念定义 4](#_Toc471204900)

[1.3 BPEL程序解析 5](#_Toc471204901)

[1.3.1 BPEL程序的数据依赖分析 5](#_Toc471204902)

[1.3.2 BPEL程序的变量分析 8](#_Toc471204903)

[1.3.3 BPEL程序解析算法 9](#_Toc471204904)

[1.3.4 小结 10](#_Toc471204905)

[1.4 后向切片算法 10](#_Toc471204906)

[1.5 BPEL故障定位示例 12](#_Toc471204907)

[1.6 本章小结 16](#_Toc471204908)

[2 基于切片的BPEL程序故障定位工具设计与实现 17](#_Toc471204909)

[2.1 需求分析 17](#_Toc471204910)

[2.2 工具的设计与实现 18](#_Toc471204911)

[2.2.1 体系结构 18](#_Toc471204912)

[2.2.2 工具实现 20](#_Toc471204913)

[2.3 工具演示 23](#_Toc471204914)

[2.4 本章小结 28](#_Toc471204915)

[3 基于切片的故障定位技术的实验评估 29](#_Toc471204916)

[3.1 实验对象与度量指标 29](#_Toc471204917)

[3.1.1 TravelAgency实例 29](#_Toc471204918)

[3.1.2 SmartShelf实例 30](#_Toc471204919)

[3.1.3 QuoteProcess实例 30](#_Toc471204920)

[3.1.4 度量指标 31](#_Toc471204921)

[3.2 实验设计 31](#_Toc471204922)

[3.3 故障版本生成 32](#_Toc471204923)

[3.4 实验结果 33](#_Toc471204924)

[3.4.1 故障定位方法 33](#_Toc471204925)

[3.4.2 故障定位准确率 34](#_Toc471204926)

[3.4.3 故障定位精度 36](#_Toc471204927)

[3.5 ODE引擎 42](#_Toc471204928)

[3.5.1 修改ODE 42](#_Toc471204929)

[3.5.2 BPEL文件部署 42](#_Toc471204930)

[3.6 本章小结 42](#_Toc471204931)

[4 结论 44](#_Toc471204932)

[参考文献 45](#_Toc471204933)

1. 基于切片的故障定位技术

本章主要介绍基于切片的故障定位技术的具体实现，介绍了基于切片的故障定位技术的原理，分析BPEL程序的数据流，解析BPEL程序，提出后向切片算法，最后通过一个实例验证该故障定位算法。

* 1. 故障定位的原理

本文介绍的故障定位技术结合谓词切换技术和程序切片技术对故障进行定位。通过谓词切换技术找到谓词选择部分的错误，找到程序的关键谓词。通过对关键谓词进行后向数据切片，找到数据依赖部分的错误。通过两种故障定位技术的结合极大的缩小了程序切片集。

在准备工作中，需要对BPEL程序植入不同类型的故障，生成多个故障版本。对每个故障版本故障定位之前，需要找到至少一个测试用例*T*能够检测出该程序存在故障，即为预计输出与实际输出不一致。

对每个故障版本，执行能发现错误的测试用例*T*，通过引擎获取执行轨迹，记为*P*(*T*)，然后在执行轨迹中获取所有的谓词集合，对谓词集合进行谓词切换，查找关键谓词。

* 找到关键谓词记为*K*，对*K*后向数据切片，得到切片集合*C*。
* 找不到关键谓词，为了保证故障定位的效率，需要对错误输出结果后向数据切片进行故障定位。

根据基于切片的故障定位的原理，本文绘制了基于切片的BPEL程序故障定位的原理图，如图1-1所示，该原理图主要包含四个部分，分别为BPEL解析、BPEL执行、谓词切换和切片分析。BPEL程序的解析的目的是解析BPEL程序的结构，获取活动中变量的定义使用信息。BPEL执行的目的是获取执行轨迹，得到实际输出结果。谓词切换的目的是找到关键谓词。切片分析的目的是通过关键谓词或错误输出语句应用后向切片算法找到可能出现故障的集合。

本文主要对原理图中的BPEL程序解析，切片分析两部分主要分析。BPEL的执行和谓词切换在课题组前期工作中已介绍。

图 1-1 基于切片的故障定位方法原理图

* 1. 相关概念定义

在介绍故障定位算法之前，首先介绍BPEL程序切片的基本概念。

**定义1.1（控制流图）**：BPEL程序控制流图是一个有向图，定义为*G*=(*N*,*A*,*s*,*e*)，*N*为结点集，即为BPEL程序中的原子活动；*A*为边集，边表示结点间的控制流向；*s*对应BPEL程序的开始结点；*e*对应BEPL程序的终止结点。

**定义1.2 （执行轨迹）**：设*P*是BPEL程序*T*对输入变量*x*的执行轨迹，记为*P*=<*n*1,*n*2,…,*n*q>，其中*n*1=*s*，*n*q=*k*且*n*i∈*N*(*1*<=*i*<*q*)。

结点*X*在执行轨迹的第*p*个位置处可表示为(*X*，*p*)，记为*X*p。

**定义1.3（切片准则）**：程序*T*的动态切片准则表示为一个三元组*C*=(*x*，*X*q，*V*)，其中*x*为程序*T*的输入；*X*q是执行轨迹*P*中位置为*q*所表示的结点；*V*是程序*T*的变量子集。

**定义1.4（结点引用集）**：设结点*X*p为执行轨迹中的任意结点。*v*为结点*X*p中引用的变量，则结点*X*p的引用集表示为：

*U*(*X*p)={*v*|*v*是在结点*X*p中被引用的变量}。

**定义1.5（结点定义集）**：设结点*X*p为执行轨迹中的任意结点。*v*为结点*X*p中定义的变量，则结点*X*p的定义集表示为：

*D*(*X*p)={*v*|*v*是在结点*X*p中被定义的变量}。

**定义1.6（变量最后定义结点）**：给定执行轨迹*P*和执行位置*r*，假如变量*v*满足以下条件，则称结点*X*p是变量*v*的最后定义结点。

（1）*vD*(*X*p)；

（2）∀*k*(*p*<*k*<*r*)和*Y*满足*P*(*k*)=*Y*，*vD*(*Y*k)。

**定义1.7（定义-使用对）**：在BPEL中的任一结点*X*p中，定义某一变量*v*i，定义*v*i变量使用到的变量为*v*j，称变量*v*i和变量*v*j之间的关系为“定义-使用对”，表示为<*v*i，*v*j>。结点*X*p中包含的“定义-使用对”集合记为*US*(*X*p)={<*v*i，*v*j>…}。

**定义1.8（定义-引用关系）**：设某一变量*v*满足如下条件，则称结点*X*p与结点*Y*t具有定义-引用关系，记为*DU*(*X*p，*Y*t)，其中(1<=*p*<*t*)。

（1）*vU*(*Y*t)；

（2）*X*p是在*Y*t中变量*v*的最后定义。

* 1. BPEL程序解析

本节主要实现BPEL程序变量的解析。由于BPEL程序的数据流不同于传统的数据流，因而首先介绍BPEL程序中的数据依赖关系，然后分析BPEL程序的变量，最后提出BPEL程序解析算法。

* + 1. BPEL程序的数据依赖分析

BPEL程序中的变量主要用于流程执行过程中存储临时的数据，作为发送和接收信息的容器。变量通过“variable”属性定义自身的名称和类型。其定义格式为：

* <variable name=“变量名”messageType=“命名空间：消息名称”/>
* <variable name=“变量名”type=“命名空间：类型”/>

第一个为WSDL消息，第二个为XML schema 自定义类型或XML scheme简单类型。

当变量类型对应复杂的WSDL消息时，因为消息的定义位置对应着不同的WSDL文档，而且每条WSDL消息又含有一组分部，从而数据依赖关系不同于传统编程语言的数据依赖关系，比较复杂，需要单独进行分析。

如图1-2所示，该流程定义了“input”，“addRequest”，“subRequest”三个变量，根据“variable”中的类型属性可以得到变量对应的类型，分别为加法服务、减法服务和自身作为服务。对应的WSDL文档中定义的消息为“caculatorprocessRequestMessage”、“addRequest”、“subRequest”。如图1-3所示，减法服务的WSDL中的“subRequest”为复杂类型的变量且含有两个分部。如图1-4所示，加法服务的WSDL中的“addRequest”为复杂类型的变量并含有两个分部。如图1-5所示，服务组合的WSDL中的“caculatorprocessRequestMessage”为复杂类型的变量，并含有三个分部。

在BPEL程序的赋值语句中，可以通过“part”属性对某一变量的某些分部分别进行赋值。当前对于BPEL程序的数据依赖分析只分析BPEL程序，将“variable”属性作为一个数据传递的基本单位，对应于传统的数据流中的变量，其分析粒度过大。即无论是一个变量被定义（使用），还是包含的某一“part”被定义（使用），都视为对该变量的定义（使用），不符合变量的定义使用要求。

根据图1-2所示的BPEL业务流程，对其中节点1、节点2和节点3分别进行分析。节点1对“addRequest”进行了赋值，随后节点2又对“addRequest”进行了赋值。依照变量的定义使用分析方法，变量“addRequest”在节点1定义后没有使用就在节点2中重新定义。通过观察发现，因为节点1是对“addRequest”变量中“a1”进行定义，节点2是对“addRequest”变量中的“a2”进行定义，因而节点2中的赋值根本没有影响节点1中的定义。节点3是对“addRequest”变量的整体使用。

节点3

节点2

节点1

<process name="caculatorprocess"

targetNamespace=http://eclipse.org/bpel/sample

xmlns:tns=”http://eclipse.org/bpel/sample”

xmlns:ns1="http://add.example.ws"

xmlns:ns2="http://sub.example.ws">

<variables>

<variable name="input " messageType="tns:caculatorprocessRequestMessage "/>

<variable name="addRequest" messageType="ns1:addRequest"/>

<variablename="subRequest"messageType="ns2:subRequest"/>

</variables>

<sequence name=”main”>

<assign>

<copy>

<from variable=”input” part=”d1”/>

<to variable=”addRequest” part=”a1”/>

</copy>

</assign>

<assign>

<copy>

<from variable=”input” part=”d2”/>

<to variable=”addRequest” part=”a2”/>

</copy>

</assign>

<invoke outputVariable=”addResponse” inputVariable=”addRequest” …>

</sequence>

图 1-2 服务组合的部分BPEL文档

<definitions targetNamespace=”http://sub.example.ws” ..>

<message name=”subRequest”>

<part name=”a1” type=”xsd:double”>

<part name=”a2” type=”xsd:double”>

</message>

图 1-3 减法服务的部分WSDL文档

<definitions targetNamespace=”http://add.example.ws” ..>

<message name=” addRequest”>

<part name=”a1” type=”xsd:double”>

<part name=”a2” type=”xsd:double”>

</message>

图 1-4 加法服务的部分WSDL文档

<definitions targetNamespace=” http://eclipse.org/bpel/sample” ..>

<message name=” caculatorprocessRequestMessage”>

<part name=”d1” type=”xsd:double”>

<part name=”d2” type=”xsd:double”>

<part name=”type” type=”xsd:string”>

</message>

图 3-5 服务组合的部分WSDL文档

上述分析表明，仅仅依赖BPEL程序来寻找变量信息，是达不到精度要求的，这就需要与WSDL文档相结合。如果变量为WSDL消息类型，如图1-6所示，变量“addRequest”为WSDL消息类型，首先根据“messageType”命名空间别名“ns1”在“process”元素中找到定义该变量的WSDL文档。根据消息名称“addRequest”可以找到对应WSDL文档中的消息类型定义。可知该变量包含两个分部，分别为“a1”和“a2”，这两个变量为不可再分的单元变量，称之为原子变量。

<definitions targetNamespace=”http://add.example.ws” ..>

<message name=”add Request”>

<part name=”a1”type=”xsd:double”>

<part name=”a2” type=”xsd:double”>

</message>

<process name="caculatorprocess"

targetNamespace=http://eclipse.org/bpel/sample

xmlns:ns1="http://add.example.ws" >

<variables>

<variable name="addRequest"

messageType="ns1:addRequest"/>

</variables>

图1-6 BPEL程序和WSDL之间的关联关系

* + 1. BPEL程序的变量分析

由于BPEL程序的数据流传递不同于传统的程序语言，因而对BPEL程序中结点中变量的定义使用关系要进行特殊分析。下述分析BPEL程序中变量的定义使用位置。

**定义1.9（变量定义结点）**设*v*为BPEL中的一个变量，若其在结点*X*中被赋值。则称*X*为*v*的定义结点，记为*def*(*v*，*X*)。在BPEL中，结点中变量的定义位置为：

* **Receive活动和Invoke活动。**

Receive活动中的“variable”属性和Invoke活动中的“outputVariable”属性用于存储外部服务发来的消息。由于这两个变量为复杂的数据变量，所以所有的分部均为变量的定义点。其中的“variable”为Receive活动的输入变量属性，“outputVariable”属性为Invoke活动中的输出属性。

* **Assign活动。**

变量位于Assign活动中的“to”元素中，*v*相当于赋值表达式的左部，即为变量或者变量的某一部分被定义。

**定义1.10（变量使用结点）**设*v*为BPEL中的一个变量，若其在结点*X*中被引用，则称*X*为变量*v*的使用结点，记为*use*(v，*X*)。

变量的使用结点又分为*c-use*（用于计算的普通结点）和*p-use*（谓词结点）。

* ***c-use*（用于计算的普通结点），普通结点中使用到的变量位置为：**
* **Invoke活动和Reply活动。**

Invoke活动中的“inputVariable”属性和Reply活动中的“variable”属性作为变量的使用位置，变量被发送到外部服务。Invoke的“inputVariable”为活动的输入变量，Reply的“variable”为活动的输出变量。这两种类型的变量都为复杂的数据变量，所有的分部均被使用。

* **Assign活动。**

变量位于“Assign”活动中的“from”元素中，*v*相当于赋值表达式的右部，变量或变量的某一部分被使用。

* **p-use在谓词语句中使用，谓词语句中使用的变量位置为：**
* **Switch活动**，在Switch活动中的“case”语句，*v*用于分支的布尔表达式中。
* **While活动**，在While活动中作为“condition”属性的属性值。
* **If活动**，在If活动中作为“condition”属性的属性值。

由于BPEL程序的“Assign”活动，包含多对<copy></copy>标签，且每对标签之间互不影响，则可知该结点类型包含多个“定义-使用对”。如果一个结点中只有使用变量，而没有定义变量，则设其定义变量设为空。

* + 1. BPEL程序解析算法

通过上述分析，在故障定位前需要对BPEL程序进行解析，得到每个结点的“定义-使用对集合”，并进行存储。

首先解析BPEL程序，获得根活动，然后获取根活动的下一层活动，依照活动的类型采取不同的策略，该解析算法(Parse)表示如下：

**步骤1：**如果活动是基本活动，则根据活动名称进一步的处理。

1. 如果为Assign活动，分别解析每一个“copy”标签，然后解析标签下的“from”属性变量为*v*1，“to”属性变量为*v*2，并将其放入结点“定义-使用对”，表示为<*v*2，*v*1>。最后将多个“定义-使用对”添加到结点的“定义-使用对集合”中；
2. 如果为Invoke活动，解析“inputVariable”属性变量为*v*1，解析“outputVariable”属性变量为*v*2，将其放入结点的“定义-使用对”表示为<*v*2，*v*1>；
3. 如果为Reply活动，解析“variable”属性变量*v*，将其放入结点“定义-使用对”表示为<null，*v*>；
4. 如果为Receive活动，解析“variable”属性变量*v*，将其放入结点“定义-使用对”表示为<*v*，null>。

**步骤2：**如果活动是结构型活动，则继续进行判断；

1. 如果活动是顺序型活动、并行型活动，则获取当前活动的下一层活动的BPEL片段递归调用Parse算法；
2. 如果为Switch活动，则解析“case”语句中的变量*v*并将其放入“定义-使用对”表示为<null，*v*>。“case”语句可以包含多个使用到的变量，则包含多个“定义-使用对”。然后取得当前活动的下一层活动的BPEL片段递归调用Parse算法；
3. 如果为While活动或If活动，则解析“condition”语句中的变量*v*并将其放入“定义-使用对”表示为<null，*v*>。“condition”语句包含多个使用到的变量，则包含多个“定义-使用对”。然后取得当前活动的下一层活动的BPEL片段递归调用Parse算法。

因为BPEL程序中的活动包含结构型活动，可以多层嵌套，但不管有多少层，一定包含基本活动。因而Parse算法不论循环递归多少次，肯定会跳出循环，不会无限执行。在上述算法中，为了简便和清晰，只保留结点的“定义-使用对”建立步骤。

* + 1. 小结

本节从BPEL程序的数据流入手，首先对比BPEL程序中的变量与传统程序变量的差异，然后对BEPL程序进行数据依赖分析。最后实现BPEL程序解析算法，实现BPEL程序结构的解析和变量的解析，为后向切片算法的前期准备工作。

* 1. 后向切片算法

在介绍后向切片搜索算法之前，需要前期一些预处理工作，主要由以下四项作为算法输入：

* + - ***P*（*T*），**执行*Ts*（能够发现错误的测试用例集）中的一个测试用例*T*，并通过引擎获取执行路径序列，记为*P*(*T*)=<*X*1，*X*2，*X*3…，*X*i-1，*X*i>，*X*i为执行轨迹中结点的名称，其中*i*为执行轨迹中的序号。
    - ***X*p，**根据执行轨迹*P*(*T*)中的谓词，通过谓词切换算法找到关键谓词*Xp*，其中1<*p*<*i*。
    - ***US*（*X*），**通过对BPEL程序解析，得到执行轨迹*P*(*T*)中每个结点*X*的“定义-使用对集合”*US*(*X*)={<*v*i，*v*j>，…}。
    - **WSDL**，描述BPEL程序中服务的相关文档。

**切片算法的过程为**：先在执行轨迹中寻找关键谓词*X*p，找到*X*p中的变量使用集合*V*a，后向搜索*X*p-1结点，判断*V*a集合中的每一个变量*v*是否在结点*X*p-1中定义。如果存在一个变量*v*在结点*X*p-1被定义，则在结点*X*p-1中找出定义*v*变量使用到的变量集合*V*b，将*V*b添加到*V*a集合中，并删除*V*a集合的*v*变量，继续后向搜索。重复上述过程，直到搜索到起始结点为止。

最后将这些结点与关键谓词的远近程度进行排序输出，即为程序*P*关于关键谓词*X*p的切片*C*。下述为故障定位算法的详细步骤。

**步骤1：**在执行轨迹中找到关键谓词*X*p，然后解析其中的“定义-使用对”，将“定义-使用对”中所有使用的变量*v*放入集合*U*中。

1. 在*v*放入集合*U*之前先判断该变量是否为复杂类型。如果变量*v*数据结构如图1-7所示，为复杂类型变量，包含三个分部*p*1、*p*2、*p*3。则需要将*v*替换为*p*1、*p*2、*p*3，然后将*p*1、*p*2、*p*3添加到集合*U*中。最后查找*p*1、*p*2、*p*3的最后定义位置。

图 1-7 *v*数据结构示意图

1. 如果变量*v*为简单数据类型，只需将变量*v*添加到集合*U*中。最后只需查找*v*的最后定义位置。

**步骤2：**搜索结点*X*p-1，搜索该结点的所有“定义-使用对”，其中一个“定义-使用对”为<*v*1，*v*2>，*v*1为结点的定义变量。由于变量*v*1可能为原子变量或复杂变量，在查找*p*1、*p*2、*p*3最后定义位置的过程中，并不能直接与当前搜索结点中的定义变量*v*1直接比较。比较过程如下：

1. 当*v*1为复杂的数据类型，如果*v*1数据结构如图1-8所示，将*v*1转换为原子变量*p*1、*p*2、*q*3和集合*U*中变量比较，最后找到*p*1、*p*2的定义位置。
2. 当*v*1为简单的数据类型，则直接和集合*U*中的变量进行比较。

图1-8 *v*1数据结构示意图

**步骤3：**找到集合*U*中变量*p*1、*p*2的定义位置后，删除集合*U*中对应的变量*p*1、*p*2。最后将定义变量*v*1使用到的变量*v*2放入到集合*U*中。同理，*v*2在放入集合*U*之前，需要对*v*2的变量类型进行判断。

**步骤4：**如果在当前搜索结点找到集合*U*中变量的定义位置，则将结点*X*p-1和搜索到的“定义-使用对”<*v*1，*v*2>添加到切片集合*C*中。

**步骤5：**对*p*进行减一操作，继续搜索，重复执行步骤2、3、4。如果*p*=0，即搜索到起始节点，停止搜索，*C*为对应的切片集合。

* 1. BPEL故障定位示例

以“SmartShelf”实例来对本文提出的故障定位方法验证。“SmartShelf”服务在接收到用户输入的参数后，根据参数的不同来执行不同分支，调用不同的服务。

该流程会根据用户输入商品的名称（name）和数量（amount）来查询其当前状态（status），根据当前状态判断该商品是否过期。如果商品过期则需要处理掉，然后返回“Expired produce has been replaced”，如果商品没有过期则返回“Goods are in good status”。

根据商品的名称查询货架上的位置和商品期望的位置进行对比。如果位置不同则表明商品摆放错误，需要重新整理该商品在货架上的位置并返回“the location is correct now”，若位置相同直接返回“location is right”。

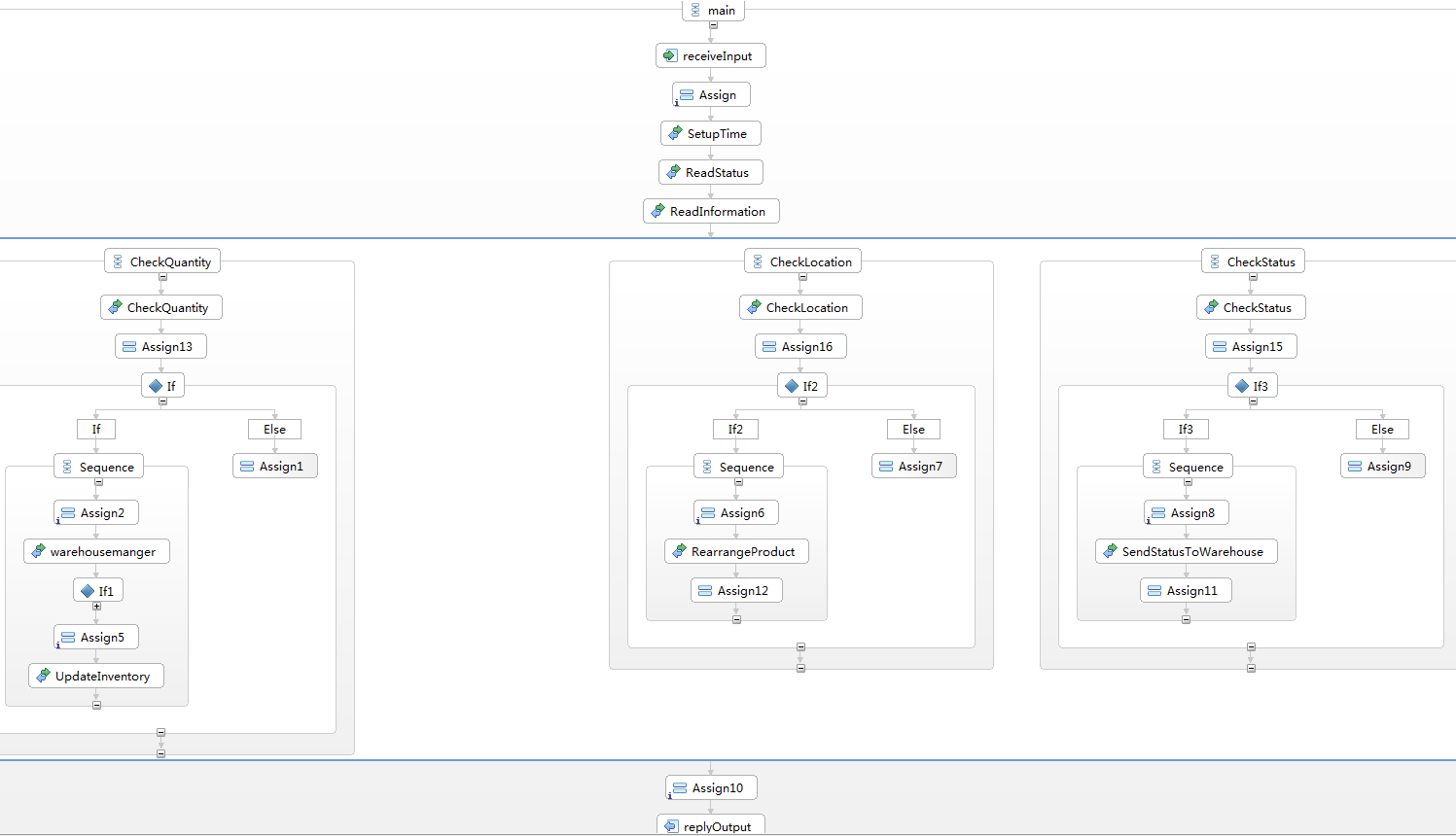


图1-9 SmartShelf的BPEL流程图

根据输入商品的数量与货架上实际的该商品数量对比，如果不足，则会与仓库中的该商品数量对比，若仓库货物大于用户输入的商品数量，则表示仓库充足，则将仓库的该商品搬运一部分到货架，并返回“Qulitity is sufficient”。若仓库不足，则会提醒工作人员进行购买并返回“warehouse is insufficient, Alert staff to purchase”。SmartShelf中的BPEL流程图如图1-9所示。

假设该BPEL程序中的结点If植入错误，If原始值为“$\_amount <$init\_amount”，将其改变为“$\_amount > $init\_amount”。然后应用后向切片算法来定位故障可能存在的位置。

**步骤1：**通过对BPEL文档和WSDL文档静态分析可以得到每个结点的“定义-使用对”。表1-1列出了每个结点中变量的“定义-使用对”。如表1-1所示，观察可以发现结点类型为“Invoke”的“定义-使用对”中使用到的变量都是为复杂类型的变量，而且都是对变量的整体赋值。一个“Assign”类型的结点通常都包含多个“定义-使用对”，并且“Assign”类型中结点的赋值都是对复杂类型的每一个分量进行赋值。

表1-1 SmartShelf程序中的“定义-使用对”

| **序号** | **类型** | **名称** | **定义-使用对** |
| --- | --- | --- | --- |
| 1 | Assign | Assign | (1)<ReadStatusPLRequest.amount-input.amount>  (2)<SetupTimePLRequest.name-input.name>  (3)<ReadInformationPLRequest.amount-input.amount>  (4)<CheckStatusRequest.amount-input.amount>  (5)<CheckQuantityRequest.amount-input.amount>  (6)<CheckStatusRequest.name-input.name>  (7)<CheckLocationPLRequest.name-input.name>  (8)<CheckLocationPLRequest.amount-input.amount>  (9)<ReadInformationPLRequest.name-input.name>  (10)<init\_amount.amount-input.amount>  (11)<ReadStatusPLRequest.name-input.name>  (12)<SetupTimePLRequest.amount-input.amount>  (13)<CheckQuantityRequest.name-input.name>  (14)<init\_name.amount-input.name> |
| 2 | Invoke | SetupTime | (15)<SetupTimePLResponse-SetupTimePLRequest> |
| 3 | Invoke | ReadStatus | (16)<ReadStatusPLResponse-ReadStatusPLRequest> |
| 4 | Invoke | ReadInformation | (17)<ReadInformationPLResponse-ReadInformationPLRequest> |
| 5 | Invoke | CheckQuantity | (18)<CheckQuantityResponse-CheckQuantityRequest> |
| 6 | Assign | Assign13 | (19)<\_amount-CheckQuantityResponse.checkQuantityReturn> |

表1-1 SmartShelf程序中的“定义-使用对”（续）

| **序号** | **类型** | **名称** | **定义-使用对** |
| --- | --- | --- | --- |
| 7 | If | If | (20)<null-\_amount>  (21)<null-$init\_amount> |
| 8 | Assign | Assign2 | (22)<WarehouseManagerPLRequest.name-input.name>  (23)<WarehouseManagerPLRequest.amount-input.amount> |
| 9 | Invoke | WarehouseManager | (24)<WarehouseManagerPLResponse-WarehouseManagerPLRequest> |
| 10 | If | If1 | (25)<null-WarehouseManagerPLResponse.warehouseManagerReturn >  (26)<null-init\_amount> |
| 11 | Assign | Assign3 | (27)<SendOrderToStaffPLRequest.name-input.name>  (28)<SendOrderToStaffPLRequest.amount-input.amount> |
| 12 | Invoke | AlertStaff | (29)<SendOrderToStaffPLResponse-SendOrderToStaffPLRequest> |
| 13 | Assign | Assign17 | (30)<quantity-'Warehouse is insufficient! Alert the staff to purchase!'> |
| 14 | Assign | Assign4 | (31)<SendOrderToCarrierPLRequest.name-input.name>  (32)<SendOrderToCarrierPLRequest.amount-input.amount> |
| 15 | Invoke | AlertCarrier | (33)<SendOrderToCarrierPLResponse-SendOrderToCarrierPLReques> |
| 16 | Assign | Assign18 | (34)<quantity-'Quantity is sufficient now!'> |
| 17 | Assign | Assign5 | (35)<UpdateInventoryPLRequest.name-input.name>  (36)<UpdateInventoryPLRequest.amount-input.amount> |
| 18 | Invoke | UpdateInventory | (37)<UpdateInventoryPLResponse-UpdateInventoryPLRequest> |
| 19 | Assign | Assign1 | (38)<quantity-'Quantity is sufficient!'> |
| 20 | Invoke | CheckLocation | (39)<CheckLocationPLResponse-CheckLocationPLRequest> |
| 21 | Assign | Assign16 | (40)<\_location-CheckLocationPLResponse.checkLocationReturn> |
| 22 | If | If2 | (41)<null-\_location > |
| 23 | Assign | Assign6 | (42)<RearrangeProductPLRequest.name-input.name>  (43)<RearrangeProductPLRequest.amount-input.amount> |
| 24 | Invoke | RearrangeProduct | (44)<RearrangeProductPLResponse-RearrangeProductPLRequest> |
| 25 | Assign | Assign12 | (45)<location-'The location of has been corrected!'> |
| 26 | Assign | Assign7 | (46)<location-'Location is right!'> |
| 27 | Invoke | CheckStatus | (47)<CheckStatusResponse-CheckStatusRequest> |
| 28 | Assign | Assign15 | (48)<\_status-CheckStatusResponse1.checkStatusReturn> |
| 29 | If | If3 | (49)<null-$\_status > |
| 30 | Assign | Assign8 | (50)<SendStatusToWarehousePLRequest.name-input.name>  (51)<SendStatusToWarehousePLRequest.amount-input.amount> |
| 31 | Invoke | SendStatusToWarehouse | (52)<SendStatusToWarehousePLResponse-SendStatusToWarehousePLRequest> |

表1-1 SmartShelf程序中的“定义-使用对”（续）

| **序号** | **类型** | **名称** | **定义-使用对** |
| --- | --- | --- | --- |
| 32 | Assign | Assign11 | (53)<status-SendStatusToWarehousePLResponse.sendStatusToWarehouseReturn>  (54)<status-'status\_init'> |
| 33 | Assign | Assign9 | (55)<status-'Goods are in good status!'> |
| 34 | Assign | Assign10 | (56)<output.location-location>  (57)<output.status-status>  (58)<output.quantity-quantity> |

**步骤2：**执行能够发现错误的测试用例*T*，得到执行轨迹*P*(*T*)。执行轨迹中的每个值表示为结点的名称，执行轨迹如下。

*P*(*T*)=<“receiveInput”，“Assign”，“SetupTime”，“ReadStatus”，“ReadInformation”，“Flow”，“CheckQuantity”，“CheckLocation”，“CheckStatus”，“Assign16”，“If2”，“Assign7”，“Assign13”，“If”，“Assign2”，“WarehouseManager”，“If1”，“Assign3”，“AlertStaff”，“Assign17”，“Assign5”，“UpdateInventory”，“Assign15”，“Assign9”，“If3”，“Assign8”，“SendStatusToWarehouse”，“Assign10”>。

**步骤3：**结合表3-1，找到与执行轨迹对应的序列号。执行轨迹中的谓词为“If2”，“If”，“If1”，“If3”。对每一个谓词进行谓词切换，即为强行改变谓词的真假分支，最后发现只有对谓词“If”切换时能够得到与预期输出一致的结果。则定义谓词“If”为关键谓词。

**步骤4：**对关键谓词“If”应用后向切片算法，最终找到切片集合。首先搜索“If”结点是否存在故障，本例中由于故障位置直接在“If”结点，所以能够直接发现故障。

1. 如果关键谓词中不存在故障，则后向切片分析。“If”结点的值为“$\_amount < $init\_amount”，包含变量“\_amount”和“init\_amount”。在执行轨迹中找到结点“If”，将“\_amount”和“init\_amount”放入*U*集合中，然后开始寻找*U*集合中变量最后定义的位置。查找每个结点的“定义-使用对”中定义的变量是否与*U*集合中变量相等。
2. 搜索到“Assign13”，包含19号“定义-使用对”，找到*U*集合中变量“\_amount”的定义位置，定义该变量使用到变量“CheckQuantityResponse.checkQuantityReturn”，因其为原子变量，直接加入到使用集合*U*中，并将“\_amount”从*U*集合中移出。当前*U*集合为{CheckQuantityResponse.checkQuantityReturn，init\_amount}。
3. 搜索“CheckQuantity”结点，包含18号“定义-使用对”，找到*U*集合中变量“CheckQuantityResponse.checkQuantityReturn”的定义位置，定义该变量使用到变量“CheckQuantityRequest”，将“CheckQuantityResponse.checkQuantityReturn”从集合*U*中移除，并将“CheckQuantityRequest”加入到集合*U*中。
4. “CheckQuantityRequest”为复杂类型的变量，不能直接放入*U*集合中，需要查找WSDL文档将其所有分部查出，将所有分部加入到*U*集合中。“CheckQuantityRequest”包含两个分部，分别为“CheckQuantityRequest.amount”和“CheckQuantityRequest.name”，将这两个原子变量添加到*U*集合中。当前*U*集合为{CheckQuantityRequest.amount，CheckQuantityRequest.name，init\_amount}。
5. 继续后向搜索到“Assign”结点，包含5号，13号，10号“定义-使用对”。找到*U*集合中所有变量的定义位置，将定义这些变量使用到的变量“input.amount”，“input.name”放入*U*集合中，并将“CheckQuantityRequest.amount”，“CheckQuantityRequest.name”，“init\_amount”从*U*集合中移除。后向搜索到“receiveInput”，为输入结点，搜索过程结束。
6. 最终找出“If”，“Assign13”，“CheckQuantity”，“Assign”是与故障相关的结点，具体可以细化到故障结点中的相关变量。
7. **Assign13**

<CheckQuantityResponse.checkQuantityReturn-\_amount>

1. **CheckQuantity**

<CheckQuantityRequest-CheckQuantityResponse>

1. **Assign**

<input.amount-init\_amount.amount>

<Input.amount-CheckQuantityRequest.amount>

<Input.name-CheckQuantityRequest.name>

* 1. 本章小结

本章介绍了基于切片的故障定位技术的原理，并设计了故障定位的框架，然后分析BPEL程序的数据流，随后提出BPEL程序解析算法和后向切片算法，最后通过“SmartShelf”实验示例演示了如何将该故障定位技术应用于BPEL程序的故障定位中。

1. 基于切片的BPEL程序故障定位工具设计与实现

本章讨论基于切片的BPEL程序故障定位工具的分析、设计与实现。主要分为需求分析、支持工具的设计与实现。最后用一个程序演示该工具的使用。

* 1. 需求分析

支持工具的主要功能包括所需文件的导入、BPEL程序的解析、测试用例的执行、谓词的切换、切片的分析。支持平台的功能需求如图2-1所示（UML用例图）。



图2-1 UML用例图

1. **文件的导入**：该用例包含BPEL文件的导入，测试用例的导入和期望输出的导入三个部分。

* BPEL文件用例：用户导入待测BEPL程序。
* 测试用例用例：用户导入针对该BPEL程序能发现错误的测试用例的文件。
* 预期输出结果用例：用户导入期望输出结果的文件。

1. **BPEL程序解析**：该用例包括BPEL解析。

* 解析BPEL：该用例解析用户导入的BPEL程序，解析BPEL中的每个结点，并分析变量的定义使用关系。

1. **用例执行：**该用例主要包括测试用例的执行、执行轨迹的收集、实际输出的获取。

* 发送消息：该用例向BPEL程序发送能发现错误的测试用例。
* 收集输出：该用例记录执行完测试用例后的返回结果。
* 收集路径：该用例获取执行测试用例后的执行轨迹。
* 结果对比：该用例将当前测试用例的输出结果和期望输出结果对比。

1. **谓词切换**：该用例依据收集的执行轨迹中的全部谓词，进行谓词切换，得到关键谓词。该用例包括收集谓词、获取关键谓词两个部分。

* 收集谓词：该用例获取当前测试用例的执行轨迹，找到其中的谓词结点。
* 获取关键谓词：对执行轨迹中的谓词分别切换，直到找到关键谓词。

1. **故障定位**：根据关键谓词或程序的输出结点，结合WSDL文档，对其数据切片分析，定位出故障集合。该用例包含切片分析和解析WSDL。

* 解析WSDL：根据BPEL中的变量信息，索引到对应的WSDL文件，并分析变量的具体类型。
* 切片分析：对关键谓词或输出节点后向切片分析，找出最可能发生故障的语句集合。
  1. 工具的设计与实现

根据上述需求分析，设计了面向BPEL程序的故障定位支持工具BPEL\_SLLocator。本节讨论BPEL\_SLLocator的体系结构以及各组件的设计。

* + 1. 体系结构

支持工具由BPEL程序解析模块、执行模块、谓词切换模块、故障分析模块组成，图2-2所示为支持工具“BPEL\_SLLocator”的体系结构示意图。



图2-2 BPEL\_SLLocator系统结构图

“BPEL\_SLLocator”的功能模块如下划分：BPEL程序解析、测试用例执行（BPEL程序部署、测试用例消息的发送、执行轨迹的获取、消息结果的对比）、谓词切换、故障定位。

(1) BPEL程序解析模块中主要实现对BPEL程序静态解析的功能，解析出每个结点的名称、类型和变量的“定义-使用对”信息。

(2) 测试用例执行模块包括待测BPEL程序部署功能、测试用例消息发送功能、执行轨迹获取功能、返回消息收集、消息结果的对比功能。通过对Apache ODE服务部署文件夹下添加新的BPEL服务文件，可以实现服务的自动部署。通过对测试用例文件的解析，来获取BPEL服务的输入参数。使用 “AXIS2”包对输入参数进行“SOAP”消息格式封装，发送消息。执行测试用例后，BPEL流程将实际的输出结果以“SOAP”格式呈现给用户，记录返回结果与程序的预期输出结果进行比较。

(3) 切换谓词模块，需要将收集到的执行轨迹中的谓词依次切换，通过对谓词语句添加“not()”，即为强行改变谓词表达式的布尔值来改变执行分支。每次切换后会生成新的BPEL程序，需要重新执行步骤2，直到找到关键谓词。

(4) 当获取到关键谓词后需要对其后向数据切片分析，最终定位出可能出现故障的模块位置。后向搜索过程中需要解析BPEL程序对应的WSDL文档，对BPEL程序中的变量类型进行判断，如果为复杂类型的变量，则解析出对应的原子变量。

* + 1. 工具实现

支持工具采用Java语言、Eclipse开发环境进行开发。下面介绍一下工具中使用到的技术。

1. 工具采用Java Swing作为图形化界面开发技术，实现与用户交互界面的设计。
2. BPEL程序结点解析采用DOM4J作为XML读写技术，使用该技术从BPEL的XML文档中读取结点信息和重新写入结点信息。
3. BPEL的部署采用Web服务器Tomcat对BPEL程序进行部署，采用Apache ODE引擎执行BPEL程序。

对支持工具的体系结构中的模块划分详细设计。图2-3是系统设计的包关系图。

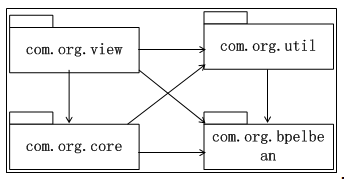


图 2-3 包关系图

下面分别介绍各个包的详细设计。

1. **系统图形界面包“com.org.view”**

图2-4为界面组件包“com.org.view”包的类图，它提供了工具的图形化界面，负责实现工具与用户交互的工作。

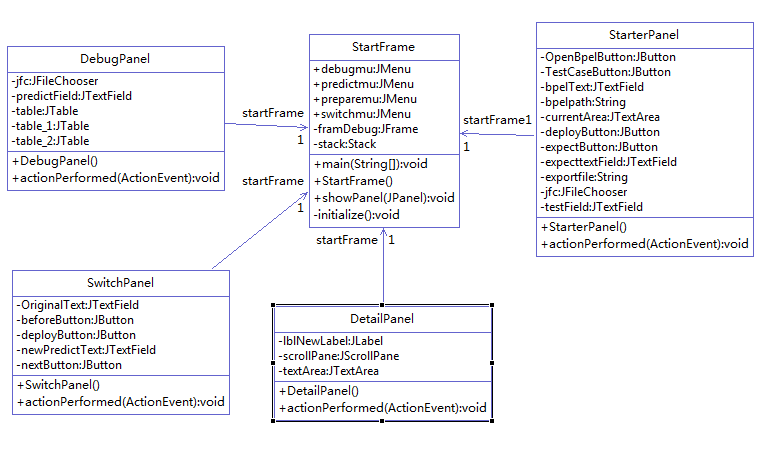


图 2-4 界面组件包类图

* **“StartFrame”类**为整个工具的入口，提供“main”方法，通过实例化该类来启动工具。“StartFrame”类继承自“Swing”中的“JFrame”类，是软件的核心窗体，它定义了软件运行窗体的标题和大小。
* **“showPanel”方法**，该方法实现继承自“Jpanel”类的对象在窗体StartFrame中显示的功能。其中“stack”变量为“StartFrame”类的全局变量，负责实现界面中的“上一步”和“下一步”的操作。当“下一步”操作时，需要重新绘制Jpanel对象，并且将被重新绘制的对象保存入“stack”中。当进行“上一步”操作时都会将“stack”顶端中的对象移除。

“StartPanel”、“DetailPanel”、“SwitchPanel”、“DebugPanel”四个类都继承自“JPanel”类，实现各自界面显示。每个类中都在其构造方法中实现界面的初始化和界面上控件的初始化操作。四个类都实现了ActionListener接口来监听各个按钮。

* **“StartPanel”类**实现故障定位的准备工作，完成各个资源的输入包含BPEL程序的输入、BPEL程序的部署、测试用例消息发送、输出结果对比等界面操作。
* **“DetailPanel”类**实现对获取的执行轨迹的显示、执行轨迹中谓词的显示。
* **“SwitchPanel”类**实现谓词的切换、切换后重新部署、重新对比。
* **“DebugPanel”类**实现最终定位结点的显示和对BPEL程序静态解析后所有的“定义-使用对”的显示。

1. **辅助工具类包“com.org.util”**

图2-5所示为支持平台的工具包组件“com.org.util”，它为整个软件提供必需的工具组件。

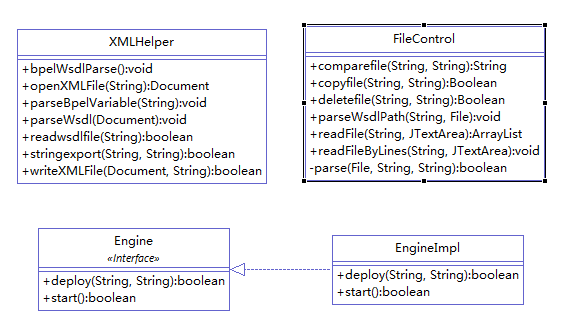


图 2-5 辅助工具包类图

* **“XMLHelper”**类封装了对WSDL文档的基本操作，包含WSDL文档的解析（bpelWsdlParse方法）、读入（readwsdlfile方法）、写出（writeXmlFile方法）等相关操作。
* **“FileControl”类**封装了对普通txt文件的基本操作，对文件的解析（parse方法）、比较（comparefile方法）、复制（copyfile方法）、删除（deletefile方法）等相关操作。
* **“EngineImpl”类**则是实现了对BPEL程序的部署（deploy方法）和测试用例消息的发送（start方法）相关操作。

1. **核心搜索包“com.org.core”**

图2-6为BPEL文件解析包“com.org.core”类图，它实现整个工具的核心搜索功能。

* **“Node”类**是BPEL的结点类，包含后继节点（afterNodes属性），前驱节点（beforeNodes属性），“定义-使用对”集合（vaHashMap）等属性。
* **“Converter”类**为BPEL文件的解析类，包含多个解析方法，针对不同的结点类型调用不同的方法解析。其中“processAssign”方法解析“Assign”结点，“processInvoke”方法解析“Invoke”结点，“processReceive”方法解析“Receive”结点，其它方法不再赘述。
* **“FindVariable”类**对BPEL中变量的定义使用关系进行搜索，为工具的核心类。“processCondition”方法为解析执行轨迹中的谓词信息，而“processSearch”方法实现对关键谓词后向切片搜索。

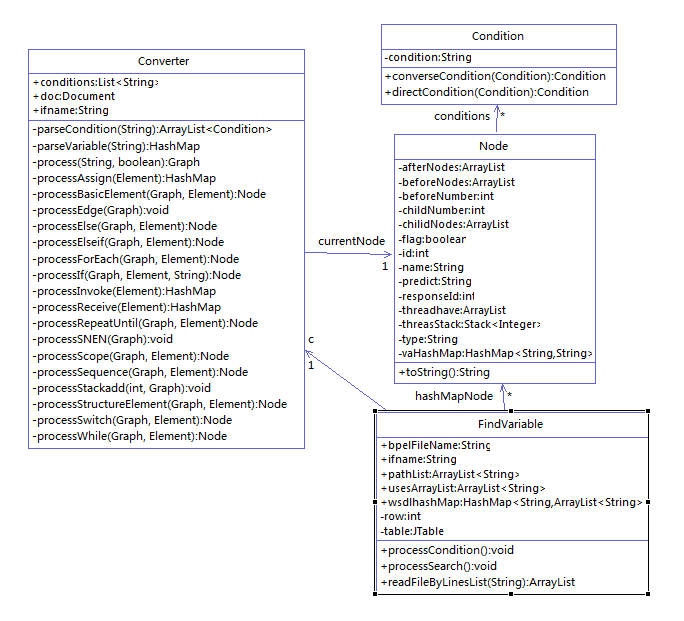


图 2-6 核心搜索包类图

* 1. 工具演示

以“SmartShelf”程序实例对故障定位工具BPEL\_SLLocator进行演示。

第一步：如图2-7所示文件选择界面，在界面中分别有三种提示，“Open BPEL”为选择待检测的BPEL程序，包含该BPEL程序的文件夹下必须包含与其相对应的WSDL文件；“Choose Case”为选择能够发现错误的测试用例文件，其为txt文件格式；“Expect Button”为执行该测试用例期望的输出结果文件。点击各自的按钮能够转换至如图2-8所示的选择框，分别选择所需的文件进入系统。

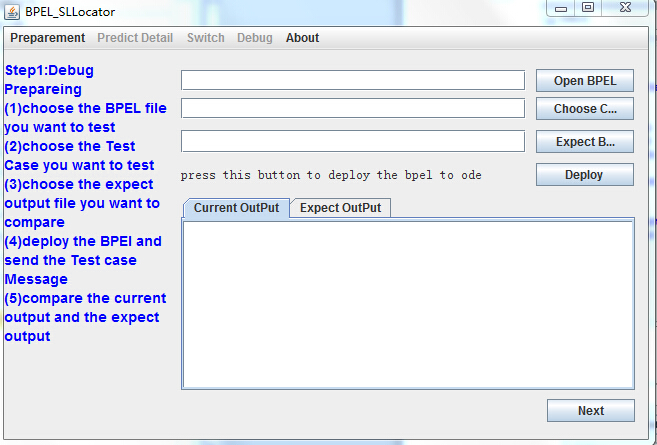


图 2-7 BPEL文件选择界面图

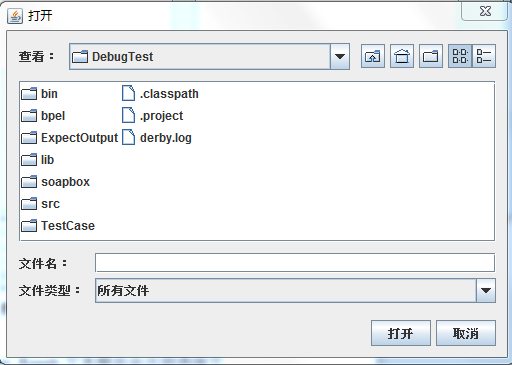


图 2-8 打开文件界面图

选择了BPEL文件和Test Case文件之后，通过“Deploy”按钮对BPEL文件进行部署，对所选中的测试用例进行消息的发送，当发送完毕后，将消息的结果回显到左边的“Current OutPut”，然后和右边的“Expect OutPut”中的预期输出结果进行对比。如果与预期结果一致则不能进行“Next”，如果对比结果不一致，能进行“Next”操作。进行“Expect Button”按钮进行页面切换后，如图2-10所示。

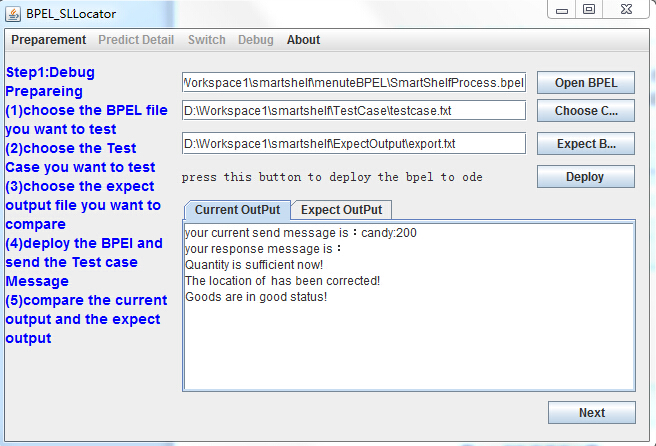


图 2-9 实际结果显示图

![HMC87`8_))P]([$D{QA[`1B](data:image/jpeg;base64,)

图 2-10 预期结果显示图

第二步，选择BPEL文件，执行失败的测试用例之后，会记录下执行该测试用例的执行轨迹，并在文本框中显示这个执行轨迹。通过对执行轨迹中结点类型的判断，提取出执行轨迹中的谓词，并将谓词的名称和谓词的内容显示在文本框中，通过“#”进行分隔。在本例执行轨迹中有四个谓词分别为“predict0”、“predict1”、“predict2”、“predict3”。每个谓词后边都对应一个文本框代表该谓词语句的内容。通过“Switch”按钮对谓词逐一切换。“SwitchAll”为对执行轨迹中的所有谓词进行自动切换，直至找到关键谓词。如图2-11所示为自动显示的结果。

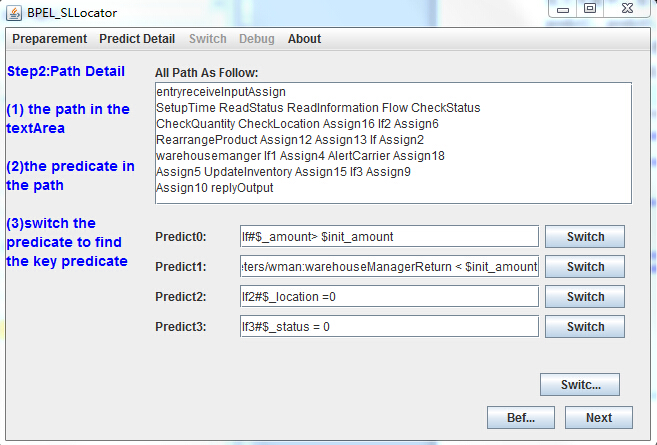


图 2-11 执行轨迹显示图

第三步，点击每一个“Switch”按钮，将会对对应的谓词进行自动切换，显示出谓词切换界面如图2-12所示。“Original Predict”对应的文本框内容为当前进行切换的谓词，“New Predict”对应的文本框内容为切换之后的谓词内容。然后对修改后的BPEL文件重新部署和重新发送消息。发送消息用到的测试用例是第一步用户输入的测试用例。最后对重新发布后的返回消息 (Current ResPonse)和预期结果(Expect ResPonse)进行对比。

如图2-13所示，如果结果相同，则确定该谓词为关键谓词，然后对该关键谓词进行下一步“Next”的故障定位；如果结果不同，则谓词不是关键谓词，返回上一步“Before”，重新对下一个谓词进行切换。

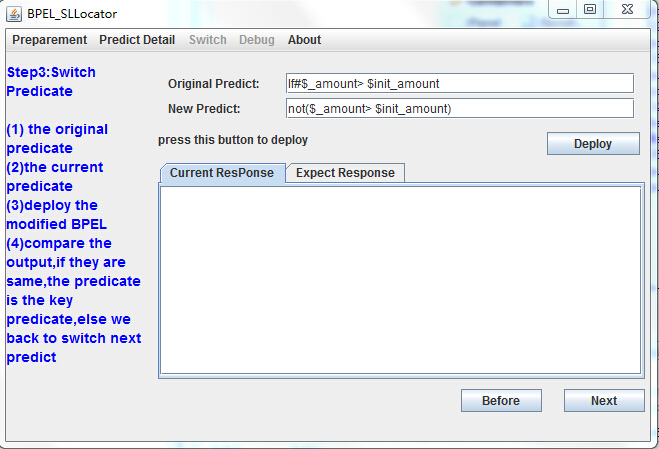


图 2-12 切换谓词的显示图

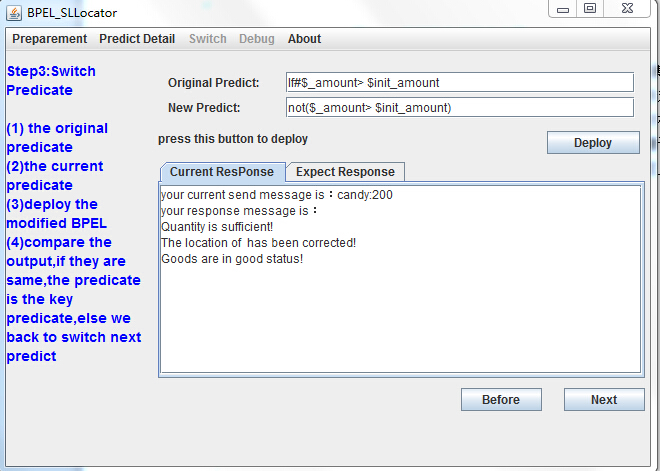


图 2-13 重新发布后的显示图

第四步，找到关键谓词后，应用后向切片算法，找到相关联的结点集合。如图2-14所示。“Debug”标签里边显示的为根据关键谓词找到的相关切片，包含结点中定义使用到的变量。如图2-15所示，“Variable”显示变量的定义使用关系是执行轨迹中所有结点的定义使用对。

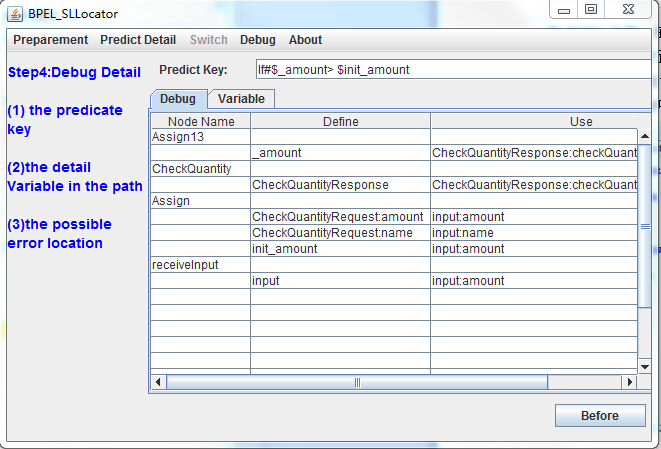


图 2-14 故障定位显示图

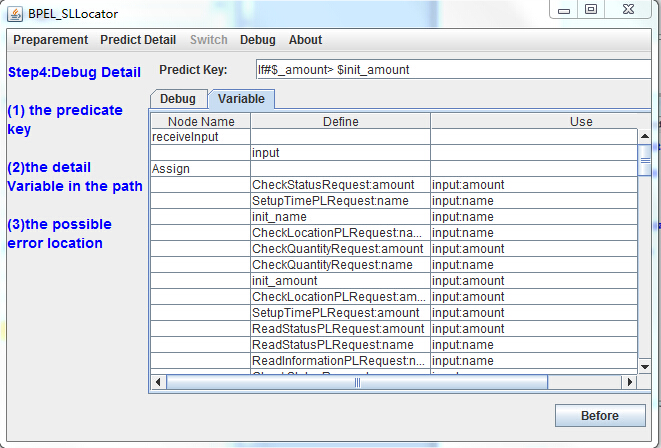


图 2-15 执行轨迹中结点变量显示图

* 1. 本章小结

本章详细介绍了基于切片的BPEL程序的故障定位支持工具的设计与实现。包括支持工具的需求分析、工具的体系结构、工具的实现，最后采用一个实例对该工具演示。支持工具提高了开发人员对BPEL程序的调试效率。

1. 基于切片的故障定位技术的实验评估

本章基于“SmartShelf”、“TravelAgency”及“QuoteProcess”三个实例来验证基于切片的故障定位技术的有效性。验证过程包括故障版本的生成，测试用例的筛选，故障版本的执行及数据的分析，最后对分析的结果从故障定位的准确率和精度两个方面进行对比。

* 1. 实验对象与度量指标

本文用于研究BPEL故障定位的BPEL程序共有三个，分别是“TravelAgency”、“SmartShelf”和“QuoteProcess”。

* + 1. TravelAgency实例

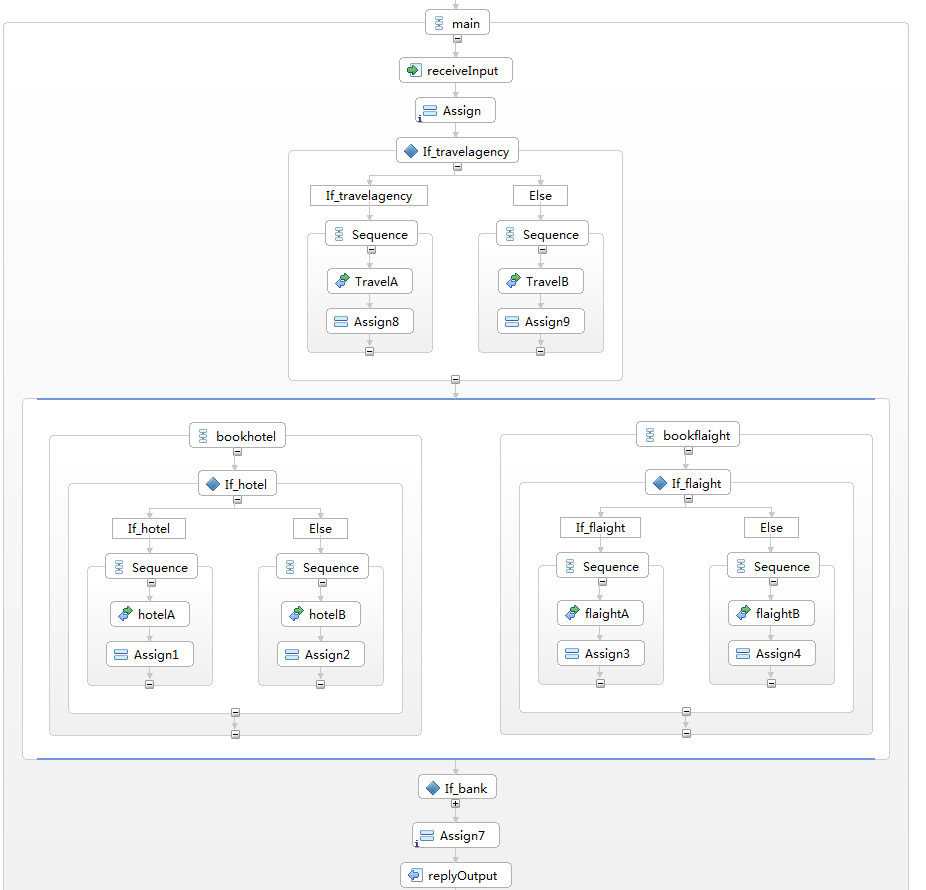


图5-1 TravelAgency实例的BPEL流程图

TravelAgency实例是旅行社预订实例，包括旅行方案选择、酒店预订、机票预订、银行结算服务。用户输入预订旅行的个人信息和人数，调用业务流程，进行输出反馈。其业务流程如图3-1所示。首先选择旅行方案travelAgency服务，然后并行选择机票预订flight服务和酒店预订hotel服务，最后选择银行结算bank服务。

* + 1. SmartShelf实例

SmartShelf实例的详细介绍见1.5节，其BPEL流程图如图1-9所示。用户需要输入商品的货物名称、数量，调用该业务流程，输出包含商品的库存数量、货架位置、是否过期等状态信息。

* + 1. QuoteProcess实例

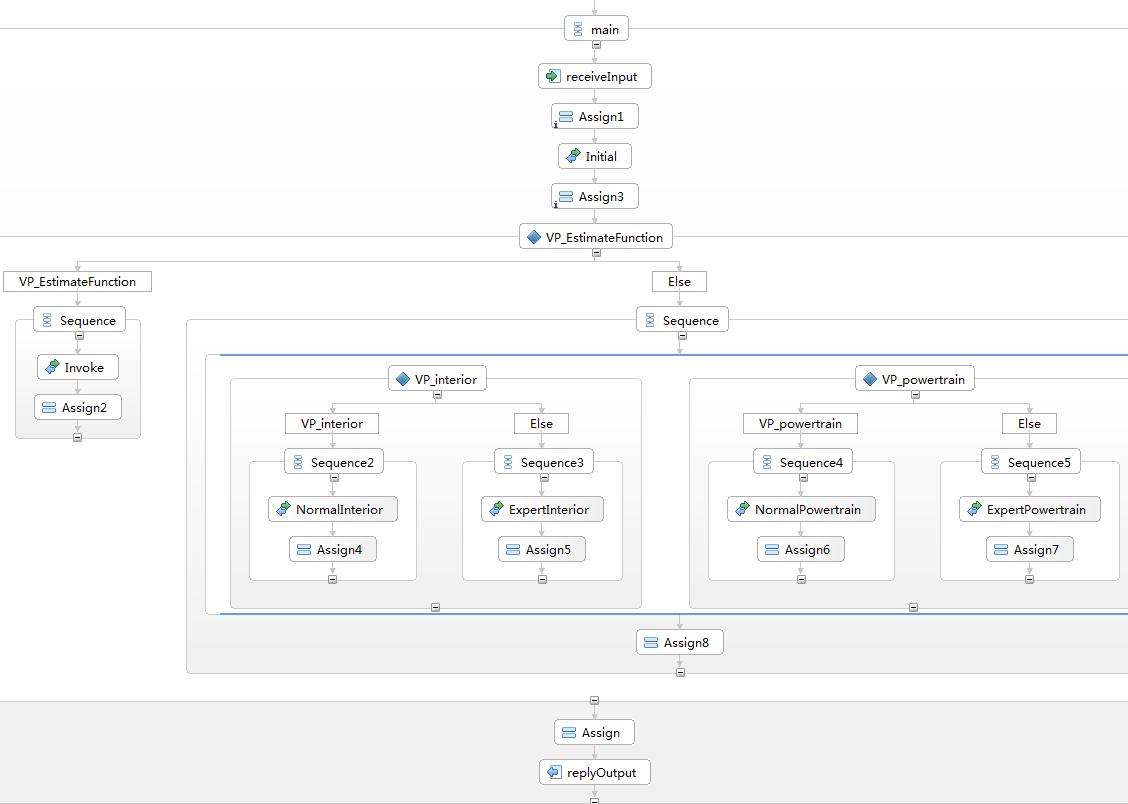


图3-2 QuoteProcess实例的BPEL流程图

QuoteProcess实例用来模拟用户活动选择，根据用户输入值来进行活动选择。业务流程在获得参数之后，根据参数的大小来选择不同的评估方案。并行选择评估标准，每组评估标准中分为专家评估和普通评估。该实例的流程如图3-2所示。

在QuoteProcess的BPEL程序中，“Receiveinput”、“Assign1”、“Initial”块主要对程序中的变量进行初始化，“Assign1”活动用来将用户的输入赋值给BPEL服务的参数。程序的结果将从“Receiveoutput”活动中输出。

* + 1. 度量指标

为了评估本文提出的基于切片的故障定位技术的有效性，使用如下两个指标进行实验度量。

**故障定位准确率：**给定一个变异版本集合总数为*P*，其中能够被正确定位出故障的变异版本个数为*P*t1，记故障定位准确率为*P*t1/*P*。

**故障定位精度：**针对某一变异版本，其程序总的语句行数为*S*，应用故障定位技术定位出可能存在故障的语句行数为*R*，记故障定位精度为*R*/*S*。

* 1. 实验设计

在本次实验中，三个实验示例都运行在Windows环境中，对于三个实验示例的具体描述如表3-1所示，其中“Program”表示程序实例名称，“Line of Code”表示程序代码行，“Mutants”表示故障版本数量，“Nodes”表示程序的原子节点的数目：

表 3-1 实验示例描述

|  |  |  |  |
| --- | --- | --- | --- |
| **Program** | **Line of Code** | **Mutants** | **Nodes** |
| SmartShelf | 579 | 57 | 36 |
| TravelAgency | 427 | 56 | 24 |
| QuoteProcess | 400 | 53 | 21 |

三个实验的实例共包括166个故障版本，下述实验的设计和统计均以该实验描述为基础。本节设计了实验步骤，包括故障版本的生成，测试用例的选择和实验结果的统计。

* **故障版本的生成**：针对每一个实例，设计了相应的故障版本，对于每个故障版本仅存在一个错误。对于故障版本的类型主要参考了文献[41]中对于BPEL故障类型的总结。所有的故障版本是由mubpel工具来完成[42]。工具生成的有些故障版本，经过大量测试用例测试后，输出结果与预计输出结果总是相同，需要剔除这类故障版本。生成的有些故障版本，输入测试用例后不能执行，由于对比的三种故障定位技术都是依赖程序执行结果的，因而对该类型故障版本进行故障定位没有可比性，需要剔除这类故障版本。
* **测试用例的选择**：大量执行随机产生的测试用例，直到找到能发现错误的测试用例，即为执行该测试用例的输出与预期的输出不一致。
* **实验结果收集**：使用设计的基于切片的BPEL程序故障定位工具对故障版本进行故障定位，并记录推荐的可能存在故障的语句集合。
  1. 故障版本生成

在本文的相关概念的介绍中，介绍了BPEL程序的故障类型。根据目前研究中提出的针对BPEL流程的故障类型，并结合BPEL实例的实际情况，在SmartShelf例子中植入了7种类型的错误，在TravelAgency例子中植入了9种类型的错误，在QuoteProcess例子中植入了9种类型的错误。如表3-2所示，其中“Type”代表的为故障的类型，“Description”代表对故障版本的举例描述。

表3-2 SmartShelf例子的故障版本

|  |  |  |
| --- | --- | --- |
| **Num** | **Type** | **Description** |
| 1 | ISV | Replace the variable “quantity”in “Assign10” to“location” |
| 2 | ERR | Replace“<”in“If ($\_amount <$init\_amount)” to“>” |
| 3 | ECN | Replace“$\_location = 0” to“$\_location = 1” |
| 4 | ASF | Replace a sequence by a flow activity |
| 5 | AIE | Remove “else” element in “warehouse” activity |
| 6 | ASI | Reverse“SetupTime”and“ReadStatus” |
| 7 | EIN | Add “not()”in“If ($\_amount <$init\_amount)” |

表3-3 TravelAgency例子的故障版本

|  |  |  |
| --- | --- | --- |
| **Num** | **Type** | **Description** |
| 1 | ISV | Replace “<bpel:from variable="quantity"/>”in “Assign10” to“<bpel:from variable="status"/>” |
| 2 | ERR | Replace“<”in“$input.payload/tns:amount <50” to“<=” |
| 3 | ELL | Replace“and”in “If\_travelagency” to“or” |
| 4 | ECN | Replace“50”in“$input.payload/tns:amount <50” to“51” |
| 5 | EIU | Replace“50”in“$input.payload/tns:amount <50” to“-50” |
| 6 | EIN | Replace“$input.payload/tns:amount <50” to “not($input.payload/tns:amount <50)” |
| 7 | EAP | Replace“$input.payload/tns:amount <50” to “$input.payload/tns:amount <50\*50” |
| 8 | EAN | Replace“$input.payload/tns:amount <50” to “$input.payload/tns:amount <-50\*50” |
| 9 | AEL | Remove “else”in “If\_hotel” |

表3-4 QuoteProcess例子的故障版本

|  |  |  |
| --- | --- | --- |
| **Num** | **Type** | **Description** |
| 1 | ISV | Replace “<bpel:from variable="interior"/>”in “Assign10” to “<bpel:from variable="train"/>” |
| 2 | ERR | Replace“<”in“($input.payload/tns:Amount < 10.0)” to“>” |
| 3 | ECN | Replace“10”in“($input.payload/tns:Amount < 10.0)” to“11” |
| 4 | AEL | Remove “else” element in “VP\_interior” activity |
| 5 | EIU | Replace“10”in“($input.payload/tns:Amount < 10.0)” to“-10” |
| 6 | EIN | Replace“($input.payload/tns:Amount < 10.0)” to“not(($input.payload/tns:Amount < 10.0))” |
| 7 | ASI | Reverse“Assign”and“Assign9” |
| 8 | CDE | Replace“($input.payload/tns:Amount < 10.0)” to“true()” |
| 9 | CCO | Replace“($input.payload/tns:Amount < 10.0)” to“false()” |

* 1. 实验结果

本节主要从故障定位的准确率和精度两个维度来对比基于切片的故障定位技术、基于谓词切换的故障定位技术、基于Tarantula的故障定位技术。

* + 1. 故障定位方法

本节主要是对Tarantula技术的计算方法、基于谓词的故障定位技术和基于切片的计算方法进行介绍，为下一节故障定位准确率和故障定位精度的计算提供理论依据。

**（1）Tarantula计算公式如下所示：**

 （3-1）

在公式（3-1）中，passed(s)表示执行程序片段s一次或多次成功的测试用例的数量，failed(s)表示执行这个程序片段s一次或多次失败的测试用例数量。totalpassed和totalfailed分别表示整个测试用例集中成功和失败的测试用例总数量。根据上述公式的计算方法来计算测试模块s的怀疑度的值，并根据怀疑度的值进行排序，得到最可能存在错误的语句集合。

**（2）基于谓词切换方法的计算方法**

执行能发现错误的测试用例，记录其执行轨迹，找出包含的谓词。每次修改一个谓词的取值并重新运行程序，观察运行结果是否与期望输出一致，多次切换，直到找到程序的关键谓词。由于谓词切换的方法只能定位到出现在谓词或谓词之前的错误，因而将在执行轨迹中关键谓词之前的语句都记为可能出现故障的位置。

**（3）基于切片的方法的计算方法**

* 在基于谓词切换的基础上能找到关键谓词，对关键谓词后向数据切片进行定位，切片集合即为故障可能出现的位置。
* 如果在谓词切换后不能找到关键谓词，直接对错误输出语句后向数据切片，所得的切片集合为故障可能出现的位置。
  + 1. 故障定位准确率

下述的实验是针对本文提出的故障定位技术准确率的验证，需要和基于谓词的故障定位技术和基于Tarantula的故障定位技术进行对比。

“SmartShelf”、“TravelAgency”、“QuoteProcess”的定位准确率见表3-5，其中的“Technique”列表示故障定位时使用到的技术，“Slicing”代表应用基于切片的故障定位技术；“Switching”代表应用基于谓词切换的故障定位技术；“Tarantula”代表应用基于Tarantula的故障定位技术。“Mutants”代表对应实例生成的变异版本的数量，“Located Mutants”代表能够定位出的变异版本个数，而“Correct Located Mutants”代表正确定位出的变异版本个数， “Correct Rate”表示故障定位技术的定位准确率。

表 3-5 三个实例的定位准确率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Program** | **Technique** | **Mutants** | **Located Mutants** | **Correct**  **Located**  **Mutants** | **Correct Rate** |
| SmartShelf | Slicing | 57 | 50 | 50 | 87.72% |
| Switching | 57 | 41 | 41 | 71.93% |
| Tarantula | 57 | 30 | 15 | 26.32% |
| TravelAgency | Slicing | 56 | 52 | 52 | 92.86% |
| Switching | 56 | 40 | 40 | 71.43% |
| Tarantula | 56 | 33 | 16 | 28.57% |
| QuoteProcess | Slicing | 53 | 46 | 46 | 86.79% |
| Switching | 53 | 35 | 35 | 66.04% |
| Tarantula | 53 | 31 | 20 | 37.74% |

表3-5列出了基于谓词切换的故障定位技术、基于切片的故障定位技术、基于Tarantula故障定位技术应用于三个实例的定位准确率。

以“SmartShelf”为例，该实例包含57个故障版本，选择基于切片技术进行故障定位可以定位出50个故障版本，其中能够正确定位出故障的变异版本个数为50个，故其定位准确率为87.72%；选择基于谓词切换的故障定位技术进行故障定位可以定位出41个故障版本，定位准确率为71.93%，选择Tarantula技术进行故障定位时可以定位出30个故障版本，在这30个定位出的故障版本中，正确定位故障的版本数为15个，故其定位准确率为26.32%。

如图3-3所示，分别对比三个BPEL实例采用不同的故障定位技术的定位准确率。“横坐标”表示采用何种故障定位技术，“纵坐标”表示故障定位的准确率。得出基于切片的故障定位技术(Slicing)应用在实验的三个实例后，其定位准确率明显高于基于谓词切换的故障定位准确率(Switching)和基于Tarantula的故障定位技术的故障定位准确率。

图 3-3 定位准确率对比

* + 1. 故障定位精度

通过实验，SmartShelf、TravelAgency、QuoteProcess的定位精度依次如表3-6，表3-7，表3-8所示。其中“Num”对应故障版本序号，“Slicing”表示应用基于切片技术进行故障定位的精度，“Switching”表示应用基于谓词切换技术进行故障定位的精度，“Tarantula”表示应用基于Tarantula技术进行故障定位的精度。其中如果精度值为“-”表示不能定位出故障的位置。“LFSL”表示应用切片技术是否正确定位到存在故障的模块（“Y”表示正确，“N”表示错误）。“LFSW”表示应用谓词切换技术是否正确定位到存在故障的模块（“Y”表示正确，“N”表示错误）。“LFTR”表示应用“Tarantula”技术能否正确定位到存在故障的模块（“Y”表示正确，“N”表示错误）。

基于切片的故障定位技术的定位精度的计算方法为定位到的语句集合除以整个待测程序的语句数；由于基于谓词的故障定位技术由于只能定位到故障可能出现在关键谓词之前，所以定位的精度计算方法为关键谓词前边所有的语句除以待测程序总的语句数；基于“Tarantula”技术的定位精度计算方法为怀疑度最高的语句集合除以待测程序总的语句数。

表3-6 Smartshelf 定位精度实验结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Num** | **Slicing** | **Switching** | **Tarantula** | **LFSL** | **LFSW** | **LFTR** |
| 1 | 2.76% | 24.87% | 6.22% | Y | Y | N |
| 2 | 4.15% | 24.87% | 6.22% | Y | Y | N |
| 3 | 6.74% | 35.58% | 16.75% | Y | Y | Y |
| 4 | 6.74% | - | - | Y | - | - |
| 5 | 8.46% | - | - | Y | - | - |
| 6 | 8.46% | - | - | Y | - | - |
| 7 | 8.46% | - | - | Y | - | - |
| 8 | 7.43% | - | - | Y | - | - |
| 9 | 7.43% | - | - | Y | - | - |
| 10 | 7.43% | - | - | Y | - | - |
| 11 | 6.39% | - | - | Y | - | - |
| 12 | 6.39% | - | - | Y | - | - |
| 13 | 5.18% | 41.28% | 24.01% | Y | Y | Y |
| 14 | 5.18% | 41.28% | 8.29% | Y | Y | Y |
| 15 | 5.18% | 41.28% | - | Y | Y | - |
| 16 | 5.18% | 33.68% | 16.75% | Y | Y | Y |
| 17 | 5.18% | 33.68% | 6.22% | Y | Y | Y |
| 18 | 4.15% | 41.62% | 10.36% | Y | Y | Y |
| 19 | 4.15% | 41.62% | 11.74% | Y | Y | Y |
| 20 | 4.15% | 38.86% | 6.22% | Y | Y | Y |
| 21 | 4.15% | 38.86% | 8.29% | Y | Y | Y |
| 22 | 4.15% | 38.86% | - | Y | Y | Y |
| 23 | 4.49% | 33.33% | 8.29% | Y | Y | Y |
| 24 | 4.49% | 33.33% | - | Y | Y | - |

表3-6 Smartshelf 定位精度实验结果（续）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Num** | **Slicing** | **Switching** | **Tarantula** | **LFSL** | **LFSW** | **LFTR** |
| 25 | 4.49% | 33.33% | 7.25% | N | N | N |
| 26 | 4.49% | 33.33% | - | Y | Y | - |
| 27 | 4.66% | 39.38% | 1.04% | Y | Y | N |
| 28 | 4.66% | 39.38% | - | Y | Y | - |
| 29 | 4.66% | 39.38% | 6.22% | Y | Y | N |
| 30 | 4.66% | 39.38% | - | Y | Y | - |
| 31 | 4.49% | 33.33% | - | Y | Y | - |
| 32 | 4.49% | 33.33% | 1.04% | Y | Y | N |
| 33 | 4.49% | 33.33% | 1.04% | Y | Y | N |
| 34 | 4.49% | 33.33% | 1.04% | Y | Y | N |
| 35 | 4.66% | 39.38% | - | Y | Y | - |
| 36 | 4.66% | 39.38% | 1.21% | Y | Y | N |
| 37 | 4.66% | 39.38% | 1.21% | Y | Y | N |
| 38 | 4.66% | 39.38% | 1.21% | Y | Y | N |
| 39 | - | - | - | - | - | - |
| 40 | - | - | - | - | - | - |
| 41 | - | - | - | - | - | - |
| 42 | - | - | - | - | - | - |
| 43 | - | - | - | - | - | - |
| 44 | - | - | - | - | - | - |
| 45 | - | - | - | - | - | - |
| 46 | 5.18% | 33.68% | - | Y | Y | - |
| 47 | 4.15% | 35.75% | - | Y | Y | - |
| 48 | 4.49% | 33.33% | - | Y | Y | - |
| 49 | 4.66% | 39.38% | - | Y | Y | - |
| 50 | 0.35% | 41.28% | 6.22% | Y | Y | Y |
| 51 | 0.52% | 41.62% | 41.62% | Y | Y | Y |
| 52 | 0.69% | 33.33% | 10.54% | Y | Y | Y |
| 53 | 0.86% | 39.38% | 39.72% | Y | Y | Y |
| 54 | 0.35% | 33.68% | 7.25% | Y | Y | N |
| 55 | 0.52% | 38.86% | 1.04% | Y | Y | N |
| 56 | 0.69% | 33.33% | 6.22% | Y | Y | N |
| 57 | 0.86% | 39.38% | 1.04% | Y | Y | N |

表3-6列出了基于谓词的故障定位技术、基于切片的故障定位技术和Tarantula技术应用在SmartShelf实例的故障定位精度。基于切片的故障定位精度，除了不能进行正确定位的故障版本外，能够被正确定位的故障版本的定位精度在10%之下。基于切片的故障定位技术的定位精度明显高于基于谓词切换的故障定位技术和基于Tarantula技术的故障定位技术的故障定位精度。

基于切片的故障定位技术的优势如下：

（1）由表3-6可知基于切片的故障定位技术比基于谓词切换的故障定位技术能够多定位出12个故障版本类型为ISV（用一个相同类型的变量标识符替换另一个），并且精度比基于谓词切换的故障定位技术有显著提高。产生这种情况的原因为：

* 当没有发现关键谓词时，通过基于谓词切换的故障定位技术不能定位此类型的错误。基于切片的故障定位技术通过对输出结果进行后向数据切片来进行故障定位，能够定位出该类故障。
* 基于切片的故障定位技术能够通过对关键谓词数据切片分析，能够更加精确的定位到关键谓词之前的错误。

（2）基于切片的故障定位技术比基于Tarantula故障定位技术定位出的故障版本多。产生这种情况的原因为：

* Tarantula故障定位技术计算公式中的分母不能为0，即为测试用例集合中成功的测试用例数量和失败的测试用例数量不能为0。实验表明，一类故障会导致预期输出结果和实际输出结果总是一致或不一致，因而Tarantula定位不出该类故障。

基于切片的故障定位技术的不足如下：

由表3-6可知基于切片的故障定位技术有些故障版本没有定位出，这些故障类型为删除节点、删除分支、活动顺序交换，造成这种情况的原因为：

* 基于切片的故障定位技术是基于执行轨迹进行故障定位分析的，删除节点，删除分支类型的错误不能记录执行轨迹，因而不能定位此类错误。

表3-7 TravelAgency定位精度实验结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Num** | **Slicing** | **Switching** | **Tarantula** | **LFSL** | **LFSW** | **LFTR** |
| 1 | 1.41% | - | - | Y | - | - |
| 2 | 1.41% | - | - | Y | - | - |
| 3 | 1.41% | - | - | Y | - | - |
| 4 | 2.81% | - | - | Y | - | - |
| 5 | 2.81% | - | - | Y | - | - |
| 6 | 2.81% | - | - | Y | - | - |
| 7 | 4.22% | - | - | Y | - | - |
| 8 | 4.22% | - | - | Y | - | - |
| 9 | 4.22% | - | - | Y | - | - |
| 10 | 5.62% | - | - | Y | - | - |
| 11 | 5.62% | - | - | Y | - | - |
| 12 | 5.62% | - | - | Y | - | - |
| 13 | 0.70% | 48.24% | 4.22% | Y | Y | Y |
| 14 | 0.70% | 48.24% | 8.67% | Y | Y | Y |
| 15 | 0.70% | 48.24% | 2.11% | Y | Y | Y |
| 16 | 0.70% | 48.24% | 2.11% | Y | Y | Y |
| 17 | 0.70% | 48.24% | 2.11% | Y | Y | N |
| 18 | 0.70% | 48.24% | 2.11% | Y | Y | Y |

表3-7 TravelAgency定位精度实验结果（续）

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Num** | **Slicing** | | **Switching** | **Tarantula** | | **LFSL** | | **LFSW** | **LFTR** |
| 19 | 0.70% | | 48.24% | 1.87% | | Y | | Y | N |
| 20 | 0.70% | | 48.24% | 6.79% | | Y | | Y | Y |
| 21 | 0.70% | | 48.24% | 2.11% | | Y | | Y | Y |
| 22 | 0.70% | | 48.24% | 2.58% | | Y | | Y | N |
| 23 | 0.94% | | 50.82% | 8.90% | | Y | | Y | Y |
| 24 | 0.94% | | 50.82% | 2.11% | | Y | | Y | Y |
| 25 | 0.94% | | 50.82% | - | | Y | | Y | - |
| 26 | 0.94% | | 50.82% | 2.11% | | Y | | Y | Y |
| 27 | 0.94% | | 50.82% | 6.79% | | Y | | Y | N |
| 28 | 1.17% | | 53.40% | 8.90% | | Y | | Y | Y |
| 29 | 1.17% | | 53.40% | 2.11% | | Y | | Y | N |
| 30 | 1.17% | | 53.40% | - | | Y | | Y | - |
| 31 | 1.17% | | 50.82% | 4.22% | | Y | | Y | Y |
| 32 | 1.17% | | 50.82% | 1.87% | | Y | | Y | N |
| 33 | 1.41% | | 55.97% | 8.43% | | Y | | Y | Y |
| 34 | 1.41% | | 55.97% | 2.11% | | Y | | Y | N |
| 35 | 1.41% | | 55.97% | - | | Y | | Y | - |
| 36 | 1.41% | | 55.97% | 8.67% | | Y | | Y | Y |
| 37 | 1.41% | | 55.97% | 2.11% | | Y | | Y | N |
| 38 | 0.70% | | 48.24% | 4.68% | Y | | Y | | N |
| 39 | 0.70% | | 48.24% | 2.11% | Y | | Y | | N |
| 40 | 0.70% | | 48.24% | 2.11% | Y | | Y | | Y |
| 41 | 0.70% | | 48.24% | 2.58% | Y | | Y | | N |
| 42 | 0.70% | | 48.24% | 2.11% | Y | | N | | N |
| 43 | 0.94% | | 50.82% | 2.11% | Y | | N | | Y |
| 44 | 0.94% | 50.82% | | 2.11% | | Y | | Y | N |
| 45 | - | - | | - | | - | | - | - |
| 46 | - | - | | - | | - | | - | - |
| 47 | - | - | | - | | - | | - | - |
| 48 | - | - | | - | | - | | - | - |
| 49 | 0.70% | 48.24% | | 2.11% | | Y | | Y | N |
| 50 | 0.70% | 48.24% | | 2.58% | | Y | | Y | N |
| 51 | 0.70% | 48.24% | | - | | Y | | Y | - |
| 52 | 0.94% | 50.82% | | - | | Y | | Y | - |
| 53 | 1.17% | 53.40% | | - | | Y | | Y | - |
| 54 | 1.41% | 55.97% | | - | | Y | | Y | - |
| 55 | 0.70% | 48.24% | | 4.68% | | Y | | Y | N |
| 56 | 0.94% | 50.82% | | 4.92% | | Y | | Y | N |

表3-7列出了三种技术应用于TravelAgency实例的故障定位精度。基于切片的故障定位技术的故障定位精度明显高于基于谓词切换的故障定位技术的定位精度和基于Tarantula的故障定位技术的定位精度。基于切片的故障定位技术比基于谓词切换的故障定位技术多定位出12个故障版本（Num（1-12）），故障类型为“ISV”（用一个相同类型的变量标识符替换另一个）。原因同表3-6分析。

基于切片的故障定位技术比基于Tarantula故障定位技术能够多定位出故障版本，产生的原因同表3-6分析。

表3-8 QuoteProcess定位精度实验结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Num** | **Slicing** | **Switching** | **Tarantula** | **LFSL** | **LFSW** | **LFTR** |
| 1 | 4.75% | - | - | Y | - | - |
| 2 | 4.75% | - | - | Y | - | - |
| 3 | 3.25% | - | - | Y | - | - |
| 4 | 3.25% | - | - | Y | - | - |
| 5 | 1.75% | - | - | Y | - | - |
| 6 | 1.75% | - | - | Y | - | - |
| 7 | 4.75% | - | - | Y | - | - |
| 8 | 4.75% | - | - | Y | - | - |
| 9 | 3.25% | - | - | Y | - | - |
| 10 | 3.25% | - | - | Y | - | - |
| 11 | 1.75% | - | - | Y | - | - |
| 12 | 1.75% | - | - | Y | - | - |
| 13 | 0.75% | 53.75% | 5.50% | Y | Y | Y |
| 14 | 0.75% | 53.75% | 5.50% | Y | Y | Y |
| 15 | 0.75% | 53.75% | 5.50% | Y | Y | Y |
| 16 | 1.00% | 57.00% | 4.75% | Y | Y | Y |
| 17 | 1.00% | 57.00% | 2.50% | Y | Y | Y |
| 18 | 1.00% | 57.00% | 12.50% | Y | Y | Y |
| 19 | 1.00% | 57.00% | 2.50% | Y | Y | Y |
| 20 | 1.00% | 57.00% | 2.25% | Y | Y | N |
| 21 | 1.25% | 54.25% | 5.00% | Y | Y | Y |
| 22 | 1.25% | 54.25% | 2.50% | Y | Y | Y |
| 23 | 1.25% | 54.25% | 12.50% | Y | Y | Y |
| 24 | 1.25% | 57.00% | 5.00% | Y | Y | Y |
| 25 | 1.25% | 57.00% | 2.50% | Y | Y | N |
| 26 | 0.75% | 53.75% | 5.50% | Y | Y | Y |
| 27 | 0.75% | 53.75% | 2.50% | Y | Y | Y |
| 28 | 0.75% | 53.75% | 5.50% | Y | Y | Y |
| 29 | 0.75% | 53.75% | 2.50% | Y | Y | Y |
| 30 | 1.00% | 54.25% | 2.50% | Y | Y | Y |
| 31 | 1.00% | 54.25% | 2.50% | Y | Y | N |
| 32 | 1.00% | 57.00% | 2.50% | Y | Y | N |
| 33 | 1.00% | 57.00% | 2.50% | Y | Y | N |
| 34 | 1.25% | 57.00% | 2.50% | Y | Y | Y |
| 35 | 1.25% | 54.25% | 2.25% | Y | Y | Y |
| 36 | 1.25% | 57.00% | 2.50% | Y | Y | N |
| 37 | 1.25% | 54.25% | 2.25% | Y | Y | N |
| 38 | - | - | - | - | - | - |
| 39 | - | - | - | - | - | - |

表3-8 QuoteProcess定位精度实验结果（续）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Num** | **Slicing** | **Switching** | **Tarantula** | **LFSL** | **LFSW** | **LFTR** |
| 40 | - | - | - | - | - | - |
| 41 | - | - | - | - | - | - |
| 42 | 0.75% | 53.75% | 2.50% | Y | Y | Y |
| 43 | 1.00% | 57.00% | 2.50% | Y | Y | N |
| 44 | 1.25% | 54.25% | 2.50% | Y | Y | N |
| 45 | 0.75% | 53.75% | - | Y | Y | - |
| 46 | 1.00% | 54.25% | - | Y | Y | - |
| 47 | 1.25% | 57.00% | - | Y | Y | - |
| 48 | - | - | - | - | - | - |
| 49 | - | - | - | - | - | - |
| 50 | - | - | - | - | - | - |
| 51 | 0.75% | 54.25% | 64.50% | Y | Y | Y |
| 52 | 1.00% | 57.00% | 2.25% | Y | Y | N |
| 53 | 1.25% | 54.25% | 2.50% | Y | Y | N |

表3-8列出了三种技术应用于QuoteProcess实例的故障定位精度。基于切片的故障定位技术的故障定位精度在0.75%至4.75%之间浮动。基于切片的故障定位技术的定位精度高于基于谓词的技术定位精度和基于Tarantula技术定位精度。

由表3-8可知，基于切片的故障定位技术比基于谓词切换的故障定位技术多定位出12个故障版本（Num（1-12）），故障类型为“ISV”（用一个相同类型的变量标识符替换另一个）。原因同表3-6分析。

基于切片的故障定位技术比基于Tarantula故障定位技术定位出的故障版本多，产生的原因同表3-6分析。

表3-6，表3-7，表3-8分析表明不同实例之间的精度存在较大差异，产生这种情况的原因是：

* 不同实例的谓词包含变量的个数不同，个数比较多的，得到的切片集合比较大，定位精度比较低，而个数比较少的，得到的切片集合比较小，定位精度比较高。
* 不同实例之间的数据依赖复杂关系不同，有的实例变量的数据依赖链包含的结点比较多，因而定位精度比较低，而有的变量的数据依赖链包含节点比较少，因而定位精度比较高。
* 不同的实例，不同结点的语句长度不一样，因而定位精度存在差异。

通过对三个实例进行实验分析发现，基于切片的故障定位技术相比于基于基于谓词的故障定位技术、Tarantula故障定位技术在故障定位精度方面有着显著提高。但是这三种技术都有着缺陷，即不能发现对BPEL程序结构进行改变的故障，如删除结点、删除分支、交换结点等故障。

* 1. ODE引擎
     1. 修改ODE

本文中故障定位方法需要通过对ODE引擎修改来获取指定的执行轨迹，主要是通过对ODE引擎包中的ode-bpel-runtime-1.3.6包进行修改。

通过export类实现活动名称的打印输出功能。通过修改引擎中的INVOKE、ASSIGN、FLOW、FOREACH、PICK、SWITCH等类，在这些类的构造方法中添加语句如下：

String path="E:\\path.txt";

print.stringexport(self.o.name+"#", path);

其中path为打印的执行轨迹，由于打印的执行轨迹中没有换行，所以在实验中，每次执行一个测试用例后都需要在该执行轨迹中添加“\r\n”，实现执行轨迹中的换行。

* + 1. BPEL文件部署

每次执行变异BPEL版本前，需要对BPEL重新部署，以“TravelAgency.bpel”实例部署为例，主要步骤如下：

1. 确保tomcat 6\webapps\ode\WEB-INF\processes文件夹下已经部署要重新部署的BPEL程序（TravelAgency）。
2. 重新部署BPEL文件的过程主要是文件替换的过程。

* 删除原先的TravelAgency.bpel文件。
* 将要重新部署的“.BPEL”文件复制到“tomcat 6\webapps\ode\WEB-INF\processes\TravelAgency\TravelAgency.bpel”。
* 将要重新部署的“.BPEL”文件对应“.deployed”删除，完成重新部署。“tomcat 6\webapps\ode\WEB-INF\processes\ TravelAgency.deployed”为对应的“.deployed”文件的路径。
  1. 本章小结

本章讨论了实验方案的设计及实验结果的分析。把基于切片的故障定位技术和基于谓词切换的故障定位技术应用于SmartShelf、TravelAgency、QuoteProcess这三个BPEL例子上。实验方案的设计包括测试用例的选择、故障版本的生成、试验数据的记录。在实验结果的分析部分，从故障定位准确率，定位的精度两个维度分析了基于切片的故障定位技术、基于谓词切换的故障定位技术、Tarantula技术的优势和不足。

1. 结论

针对BPEL程序的特点及其特有的故障类型，本文将基于切片的故障定位技术应用于BPEL程序的故障定位，提出了一个基于切片的BPEL程序的故障定位框架。将能够检测出故障的测试用例、BPEL程序语句集合及预期输出作为输入，输出最有可能存在故障的语句集合。根据该框架开发出了相应的故障定位支持工具，并用实例演示了这个工具的使用。最后为了验证和评估基于切片的故障定位技术的有效性，将基于切片的故障技术与基于谓词切换的故障定位技术和基于Tarantula的故障定位技术的定位准确率和精度进行了对比。

本文取得的主要研究成果包括：

1. **提出了一种基于切片的BPEL程序故障定位方法**

该方法通过执行能发现故障的测试用例，记录执行轨迹，然后基于执行轨迹进行程序切片，极大的缩小了定位故障的范围和提高了定位的准确率。

1. **开发了基于切片的BPEL程序的故障定位支持工具**

设计并实现了基于切片的BPEL程序故障定位支持工具BPEL\_SSLocator，实现了流程的自动化。

1. **采用经验研究的方式评估提出的基于切片的故障定位技术的有效性**

采用了三个程序实例验证提出的故障定位技术及支持工具的有效性。将基于切片的故障定位技术与基于谓词的故障定位技术和基于Tarantula的故障定位技术从故障定位准确率和故障定位精度两个维度进行了比较。实验结果表明本文提出的基于切片的故障定位技术更加高效。

需要进一步研究的工作包括：

* 本文应用了三个实例来验证基于切片的故障定位技术的有效性，在后期的研究中希望采用更多的BPEL实例来验证该技术的有效性。
* 改进故障定位的支持工具，提供更好的用户体验。
* 本文的故障定位研究仅仅停留在BPEL程序的故障定位上，希望未来的研究可以深入到BPEL程序所调用的外部服务，深入探索BPEL服务的故障定位问题。

参考文献

1. M. Papazoglou, P. Traverso, S. Dustdar. Service-Oriented Computaing: A Research Roadmap[J]. International Journal on Cooperative Information Systems, 2008, 17(2): 223-255.
2. 孙昌爱. SOA软件开发中的若干开放问题探讨[EB/OL].北京:中国科技论文在线 .http://www.paper.edu.cn/releasepaper/content/201107-461, 2011.
3. C. Sun, Y. Zhai, Y. Shang, Z.Y. Zhang. Toward Effectively Locating Integration-Level Faults in BPEL Programs[C]//Proceeding of twelfth International Conference on Quality Software (QSIC 2012), 2012: 17-20.
4. 白晓颖, 赵冲冲, 戴桂兰. Web 服务测试研究[J]. 计算机科学, 2006, 33(2): 252-256.
5. OASIS, Web Services Business Process Execution Language Version 2.0[EB/OL], http://docs. oasisopen.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.
6. C. Sun, Y. Zhai, Y. Shang, Z.Y. Zhang. BPELDebuger: An effective BPEL-specific fault localization framework[J]. Information and Software Technology, Elsevier, 2013, 55(12): 2140-2153.
7. Bertolino A, Gao J, Marchetti E, et al. TAXI - a tool for XML-based testing[C]//Proceedings of the International Conference on Software Engineering (ICSE 2007), 2007: 53-54.
8. J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments[C]//Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), 2005: 402-411.
9. J. Boubeta-Puig, A. García-Domínguez, and I. Medina-Bulo. Analogies and Differences between Mutation Operators for WS-BPEL 2.0 and Other Languages[C]//Proceedings of the 6th International Workshop on Mutation Analysis (ICSTW 2011), 2011: 398-407.
10. G. Canfora, and M. Di Penta. Service Oriented Architecture Testing: A Survey[J]. LNCS 5413, Springer, 2009: 78-105.
11. 虞凯, 林梦香. 自动化软件错误定位技术研究进展[J]. 计算机学报, 2011, 34(8): 1411-1424.
12. 翟忆蒙. 面向BPEL程序的故障定位方法与工具研究[D]. 北京: 北京科技大学计算机与通信工程学院, 2014.
13. 郑彩云. 基于谓词切换的BPEL程序故障定位技术与支持工具研究[D]. 北京: 北京科技大学计算机与通信工程学院, 2015.
14. 叶钰, 应时, 李伟斋, 张韬. 面向服务体系结构及其系统构建研究[J], 计算机应用研究, 2005, 22(2): 32-35.
15. Huhns M N, Singh M P. Service-Oriented Computing: Key Concepts and Principles[J]. IEEE Internet Computing, 2005, 9(1): 75-81.
16. W3C. Simple Object Access Protocol Version 1.1 [EB/OL]. http://www.w3.org/TR/soap/.
17. W3C. Web Services Description Language Version 2.0 [EB/OL]. http://www.w3.org/TR/wsdl20/.
18. OASIS. UDDI Version 3.0.2 [EB/OL]. <http://uddi.org/pubs/uddi_v3.htm.>
19. Curbera F, Duftler M, Khalaf R, et al. Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI[J]. IEEE Internet Computing, 2002, 6(2): 86-93.
20. 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): 2012-2014.
21. 曾云峰, 周航, 黄志球. BPEL 的测试用例生成研究[J]. 计算机工程与设计, 2008, 29(20): 5243-5246.
22. J. García-Fanjul, J. Tuya, et al. Generating test cases specifications for BPEL compositions of web services using SPIN[C]//Proceedings of the International Workshop on Web Services–Modeling and Testing (WS-MaTe 2006), 2006: 83.
23. Y. S. Ma, J. Offutt, et al. MuJava: an automated class mutation system[J]. Software Testing, Verification and Reliability, 2005, 15(2): 97-133.
24. J. J. Domínguez-Jiménez, A. Estero-Botaro, et al. GAmera: an automatic mutant generation system for WS-BPEL compositions[C]//Proceedings of the 7th IEEE European Conference on Web Services (ECOWS 2009), 2009: 97-106.
25. A. Estero-Botaro, F. Palomo-Lozano, I. Medina-Bulo. Quantitative evaluation of mutation operators for WS-BPEL compositions[C]//Proceedings of the 3rd International Conference on Software Testing (ICSTW 2010), 2010: 142-150.
26. J. A. Jones．Fault Localization Using Visualization of Test Information[C]//Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), 2004: 54-56.
27. L. N. Hong, R. Chen. Statistical fault localization with reduced program runs[M]. Artificial Intelligence Applications and Innovations IFIP Advances in Information and Communication Technology, 2010, 399: 319-327.
28. M. Y. Chen，E. Kiciman, E. Fratkin, et al. Pinpoint: problem determination in large, dynamic internet services[C]//Proceedings of the 32nd International Conference on Dependable Systems and Networks (DSN 2002), 2002: 595-604.
29. R. Abreu, P. Zoeteweij, J. C. Gemund. On the accuracy of spectrum based fault localization[C]//Proceedings of the Testing: Academic and Industrial Conference, Practice and Research Techniques (TAIC PART 2007), 2007: 89-98.
30. 惠战伟, 黄松, 嵇孟雨. 基于程序特征谱整数溢出故障定位技术研究[J]. 计算机学报, 2012, 35(10): 2204-2214.
31. 贺韬, 王欣明, 周晓聪. 一种基于程序变异的软件故障定位技术[J].计算机学报, 2013, 36(11): 2236-2243.
32. C. Liu, X. F. Yan, L. Fei. SOBER: statistical model-based bug localization [J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 286-295．
33. X. Y. Zhang. Locating Faults Through Automated Predicate Switching[C]//Proceedings of the 28th International Conference on Software Engineering (ICSE 2006). ACM, 2006: 272-281.
34. M. Weiser. Program Slicing[J]. IEEE Transactions on Software Engineering, 1984, 10(4): 352-357.
35. K. J. Ottenstein. The program dependence graph in a software development environment[J]. Acm Sigplan Notices, 1984, 19(5): 177-184.
36. S. Horwitz, T. Reps, D. Binkley. Interprocedural slicing using dependence graphs[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, 12(1): 26-60.
37. B. Korel and J. Laski. Dynamic Program Slicing[J]. Acm Sigplan Notices, 1990, 25(6): 246-256.
38. H. Agrawal, R. DeMillo, E. Spafford. Debugging with dynamic slicing and backtracking[J]. Software Practice and Experience, 1993, 23(6): 589-616.
39. N. Gupta, H. He. Locating Faulty Code Using Failure-Inducing Chops[C]//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), 2005: 263-272.
40. 曹鹤玲, 姜淑娟, 鞠小林. 基于动态切片和关联分析的故障定位方法[J]. 计算机学报, 2013, 37(11): 104-109.
41. E. Antonia, P. L. Francisco. Mutation Operators for WS-BPEL 2.0 [C]//Proceedings of the 21th International Conference on Software & Systems Engineering and their Applications, 2008: 143-151.
42. A. Estero-Botaro, F. Palomo-Lozano. et al. Quality metrics for mutation testing with applications to WS-BPEL compositions[J]. Software Testing Verification and Reliability, 2014, 25(5-7): 536-571.
43. 金仙力, 杨庚. 一种基于 ODE 的服务组合自动化部署方案[J]. 计算机与数字工程, 2012, 40(10): 73-75