基于谓词切换的 BPEL 程序故障定位技术与支持工具研究

郑彩云

北京科技大学

密　　级：公开

论文题目：**基于谓词切换的 BPEL 程序故障定位技术与支持工具研究**

学　　号：G20138526

作　　者：郑彩云

专 业 名 称：软件工程

2015 年 12 月 23 日

基于谓词切换的 BPEL 程序故障定位技术与支持工具研究

研究生姓名：郑彩云

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：Zheng Caiyun

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： 基于谓词切换的 BPEL 程序故障定位技术与支持工具研究

作者： 郑彩云

指 导 教 师： 孙昌爱　**单位：** 北京科技大学

指导小组成员：　**单位：**

　　单位：

论文提交日期：2015 年 12 月 24 日

学位授予单位：北 京 科 技 大 学

致　谢

在本文完成之际，我衷心地感谢我的导师孙昌爱老师并向他致以最崇高的敬意！在硕士两年半的学习中，孙老师为人师表，治学严谨的态度让我受益匪浅。无论是本文的选题、研究过程中的问题讨论、学术论文的撰写、本文的修改与最终完成，还是从做人、做事、做学问等，我都得到了孙老师的无私帮助，自身也受到了其潜移默化的影响。

同时也感谢这两年半来与我互勉互励的诸位同学，在各位同学的共同努力之下，我们始终拥有一个良好的生活环境和一个积极向上的学习氛围，能在这样一个团队中度过，是我极大的荣幸。

最后，我要感谢参与我论文评审和答辩的各位老师，他们给了我一个审视几年来学习成果的机会，让我能够明确今后的发展方向，他们对我的帮助是一笔无价的财富。我将在今后的工作、学习中加倍努力。再次感谢他们，祝他们一生幸福、安康!

摘要

故障定位是程序调试的一项重要环节，在确定程序存在故障的前提下通过某种技术来找到程序中故障的存在位置。目前的故障定位技术主要有基于代码检测的故障定位技术、基于程序分析的故障定位技术、基于频谱的故障定位技术和基于谓词的故障定位技术。不同的故障定位技术对不同类型的程序的有效性不同。与传统的面向对象或面向过程的程序相比，BPEL 程序具有很多新的特点，且 BPEL 程序的故障定位技术的研究尚处于起步阶段。

本文提出了一种基于谓词切换的 BPEL 程序故障定位技术，通过谓词切换逐步缩小程序的故障定位的范围，并采用与关键谓词相关的程序切片来分析故障的根源。开发了基于谓词切换的 BPEL 程序的故障定位工具 BPEL_PSLocator，用来辅助开发人员进行 BPEL 程序的故障定位。本文取得的主要成果总结如下：

- **提出了一种基于谓词切换的 BPEL 程序故障定位技术**：该方法基于 BPEL 程序的语句块结构和能够检测出故障的测试用例，通过多次切换谓词找到关键谓词，分析与关键谓词相关的程序切片定位程序故障。
- **开发了基于谓词切换的 BPEL 程序的故障定位支持工具**：设计与实现基于谓词切换的 BPEL 程序的故障定位工具，提高了故障定位的效率。
- **采用经验研究的方式评估了提出的故障定位的有效性**：采用了三个程序实例验证了基于谓词切换的故障定位技术在 BPEL 程序故障定位过程中的可行性，评估了基于谓词切换的故障定位技术的定位效率和定位精度。实验结果表明：与其他的故障定位技术相比，基于谓词切换的故障定位技术在 BPEL 程序故障定位中具有更高的定位效率和定位精度。

本文研究基于谓词切换的技术应用于 BPEL 程序的故障定位，提出的面向 BPEL 程序的故障定位技术与支持工具，该工具有更高的定位效率和定位精度。

关键词：故障定位，程序调试，服务组装，BPEL

A Predicate Switching based Approach to Locating Faults of BPEL Programs and Supporting tool

Abstract

Fault localization is an important step of program debugging. Its basic idea is to employ some techniques to find the possible location of faults in the program. So far, the representative fault localization techniques include program analysis-based fault localization techniques, statistics-based fault localization techniques, and predicate-based fault localization techniques. The effectiveness of fault localization techniques may vary with different types of programs. BPEL programs are significantly different from traditional programs in that they are represented as XML files and with many new features. Up to now, the study on fault localization for BPEL programs is still in its infancy.

This thesis proposed a predicate switching based fault localization technique for BPEL programs which narrows the search scope of fault location by critical predicate switching and precisely determines the location of faults by slicing analysis of relevant statements. The main contributions of this thesis are summarized as follows:

- A predicate switching-based BPEL program fault localization framework: The framework is based on BPEL program block structure and failed test case that detected the fault. It first searches the possible location of the fault through multiple switching processes in order to find the critical predicate, and then narrow the suspicious nodes by means of slicing analysis of statements that are related to the critical predicate.
- A predicate switching-based fault localization tool that was developed based on the proposed fault localization framework, and to automate the localization process as much as possible.
- An empirical study where three BPEL programs were used to validate the applicability and effectiveness of the proposed fault location technique in terms of fault localization correctness and precision. We also compared the effectiveness of our technique with two other fault localization techniques

(Tarantula and Code-Coverage). Experimental results show that the proposed fault localization technique can deliver better fault localization effectiveness and precision.

This thesis focuses on key issues of applying the predicate switching fault localization technique to BPEL programs. Comparing with the existing fault localization techniques, the proposed fault localization technique and supporting tool is more effective in locating faults for BPEL programs.

Key Words：Fault localization, Program debugging， Service compositions，BPEL

目录

1 引言

随着计算机技术的发展和软件应用领域的扩展，由软件故障而引发的软件维护等一系列问题逐步引起人们的重视。软件中的故障通常由程序故障引起。程序故障是指程序在运行过程中出现的一种不正常的内部状态[43]。当存在至少一个测试用例能够检测出程序中出现故障时，则需要进行程序调试。

程序调试是找到程序中的故障并进行修正的过程。调试活动主要由两部分组成：一是确定程序中可疑故障的确切性质和位置；二是对程序（设计、编码）进行修改，排除这个故障[1]。由于程序的故障定位的难度要大于故障修正的难度，而且程序调试受到多种因素的影响，比如程序规模、程序语言、开发人员的能力等[2]，因此一种高效的故障定位技术对提高开发效率至关重要。

传统的故障定位技术有很多种。根据定位故障过程中“是否需要运行软件”的准则，可以将故障定位技术分为以下两类，包括基于静态分析的故障定位技术和基于测试的故障定位技术。基于静态分析的故障定位技术不用运行软件，而是依据程序语言的语法和语义，静态地分析软件结构和程序实体之间的依赖关系来进行故障定位。基于测试的故障定位技术首先需要设计测试用例，然后运行软件程序，最后根据软件程序的动态执行信息和输出结果进行故障定位[10]。

根据定位故障利用的信息及定位方式的区别，基于测试的故障定位技术可以分为基于代码检测的故障定位技术、基于程序切片的故障定位技术、基于频谱的故障定位技术、基于谓词的故障定位技术等。基于代码检测的故障定位技术通过嵌入式模块分析方法获取软件发生故障时的模块运行序，通过检测这些模块中的代码来定位程序的故障[44]。基于程序切片的技术通过寻找和错误输出相关的程序语句来定位程序中的故障[14]。基于频谱的技术通常是根据两类运行(成功运行和失败运行)中程序语句或分支的状态特征的统计信息，直接定位到故障语句，其中成功运行是指程序的实际输出与预期输出不一致，失败运行是指程序的实际输出与预期输出一致。基于谓词的故障定位技术从谓词入手，期望通过寻找谓词与错误输出的关系来缩小故障搜索的范围。课题组在前期的 BPEL 程序故障定位研究中利用了基于频谱的技术，实验表明其定位效率仅达到 50%。因此本文希望从基于谓词的故障定位技术角度来找到一种面向 BPEL 程序的故障定位技术。

BPEL(Business Process Execution Language)是一种基于 XML 的服务组装语言，广泛用于开发面向 Web 服务的分布式程序。BPEL 程序与传统程序有着明显的不同，主要表现在：

- BPEL 定义了一套完整的执行语句，如顺序、选择和循环等控制结构，所有的控制结构皆以 XML 的格式存在。
- BPEL 程序是以活动(Activity)为单位，是一种解释性的编程语言。
- BPEL 程序可以以伙伴连接的形式调用其它外部服务。

上述特点使得 BPEL 程序的故障定位面临新的挑战，既无法直接通过设置断点的方式进行故障定位，也难通过程序切片及频谱技术实现快速的故障定位。鉴于基于谓词切换的故障定位技术已经在 C 语言的故障定位中取得了很好的效果，本文将基于谓词切换的故障定位技术应用于 BPEL 程序的故障定位中，并通过实例来验证其对 BPEL 程序故障定位的效率，同时开发了相应的支持工具 BPEL_PSLocator，较好的解决了 BPEL 程序的故障定位问题。

2 背景介绍

本章介绍 BPEL 相关概念与技术、国内外研究进展、研究的背景和意义以及论文组织结构。

2.1 Web 服务相关概念简介

2.1.1 WSDL

WSDL 是基于 XML 的用于描述 Web 服务以及如何访问 Web 服务的语言，对于使用标准化的消息格式/通信协议的 Web 服务，它需要以某种结构化的方式对 Web 服务的调用/通信加以描述，这是 Web 服务即时装配的基本保证。

WSDL 文档将 Web 服务定义为服务访问点或端口的集合。在 WSDL 中，由于服务访问点和消息的抽象定义已从具体的服务部署或数据格式绑定中分离出来，因此可以对抽象定义进行再次使用。例如消息和端口，消息是指对交换数据的抽象描述，端口类型是指操作的抽象集合，用于特定端口类型的具体协议和数据格式规范构成了可以再次使用的绑定。

WSDL 定义了一套基于 XML 的语法，将 Web 服务描述为能够进行消息交换的服务访问点的集合。它包含一系列描述某个 Web 服务的定义，包括 Types、Message、PortType、Operations、Binding、Port、Service 等。

- **Types** 指定 Message 中需要的类型，其中 Message 可以理解为函数中的参数，如果一个函数有多个参数应该把这些参数定义到一个 Message 中而不能定义为多个 Message。
- **Message** 指通信消息的数据结构的抽象类型化定义。使用 Types 所定义的类型来定义整个消息的数据结构。
- **ProtType** 定义一个服务接口，定义在服务中应该包含哪些操作，每一个 PortType 对应一个 Web 服务。
- **Operations** 用来定义操作，也就是对应每个接口中的函数名称，每一个 Operations 都可以有 input， output， fault 等输入输出和错误选项。
- **Binding** 用来绑定相应的传输协议，它定义的是一种通讯的方式，每一个 PortType 对应一个 Binding，然后在 Binding 中进一步细化设置每一个操作，配置每一个 input, output, fault 的传出方式，编

码方式等，Web 服务则将 Binding 与对应的地址相关联。

- **Port** 为协议/数据格式，绑定与具体 Web 访问地址组合的单个服务访问点。
- **Service** 描述的是一个具体的被部署的 Web 服务所提供的所有访问入口的部署细节，一个 Service 往往会包含多个服务访问入口，而每个访问入口都会使用一个 Port 元素来描述。

2.1.2 BPEL 程序

BPEL 程序本身是一个服务，它的功能是将已经定义过的具体的服务组装起来。BPEL 流程的基本结构可以分为八个部分：变量(variable)声明、伙伴链接(partnerLink)声明、处理器(handler)声明、活动、关联集合、事件处理程序、事务与补偿机制、异常管理。BPEL 程序样例如图 2-1 所示。

（1）**变量声明**：定义了程序使用的数据变量，包括两种类型：XML Schema 简单类型、XML Schema 元素及 WSDL 消息类型。变量是有作用域的，每个变量只有在它定义的作用域内才是可见的，如图 2-2 所示。XPATH 为数据处理提供支持。BPEL 程序支持的四种表达式包括布尔表达式、持续时间表达式、截止时间表达式、普通表达式(string， number， boolean 格式)。

```
<process name="ncname" targetNamespace="uri"
          queryLanguage="anyURI"?
          expressionLanguage="anyURI"?
          suppressJoinFailure="yes|no"?
          enableInstanceCompensation="yes|no"?
          abstractProcess="yes|no"?>
    <partnerLinks> ... </partnerLinks>
    <partners>... </partners>
    <variables>... </variables>
    <correlationSets> ... </correlationSets>
    <faultHandlers>... </faultHandlers>
      <compensationHandlers> ...
      </compensationHandlers>
    <eventHandlers> ... </eventHandlers>
    activity.
</process>
```

图 2-1 BPEL 程序样例

```
<variables>
      <variable name=”po” messageType=”lns:POMessage” />
      <variable name=”Invoice” messageType=”lns:InvMessage” />
      <variable name=”POFault” messageType=”lns:orderFaultType” />
<variables>
```

图 2-2 BPEL 程序变量定义

（2）**伙伴链接声明**：描述了 BPEL 流程同其内部被调用的 Web 服务的关系。通过 PartnerLinkType 定义与 Web 服务的通信接口，与 WSDL 文档中的 PortType 相对应。一个流程既可以作为服务的请求者也可以扮演服务的提供者，BPEL 程序中的伙伴定义如图 2-3 所示，其中 myRole 指出了业务流程本身的角色，属性 partnerRole 指出了伙伴的角色。

```
<partnerLinks>
      <partnerLink name=”customer” serviceLinkType= ”lns:purchasePLT” myRole
        = ”purchaseService” />
      <partnerLink name=”inventoryChecker” serviceLinkType=”lns:inventoryPLT”
      myRole=”inventoryRequestor” partnerRole=”inventoryService” />
      <partnerLink name=”creditChecker” serviceLinkType=”lns:creditPLT”
      myRole=”creditRequestor” partnerRole=”creditService” />
</partnerLinks>
```

图 2-3 BPEL 程序样例

（3）**活动**：BPEL 程序中的活动包括基本活动和结构化活动，基本活动在程序中被称为原子语句，结构化活动在程序中被称为非原子语句。基本活动是与外界进行交互的最简单的形式，活动内不会嵌套其他活动，是无序的。基本活动包括 Receive、Invoke、Reply、Assign、Throw、Exit、Wait、Empty。其中各个活动表示的意义如下所示：

- **Receive**：从流程的外部伙伴那获取数据，并将其保存到流程变量。通常一个 Receive 是一个流程的初始点，它会阻塞执行直到匹配的消息的到达。该元素有 5 个属性，分别是 partnerLink、portType、operation、variable、createInstance。
- **Reply**：活动发送消息给伙伴来应答通过 receive 活动所接收到的消息。
- **Invoke**：活动允许业务流程同步或异步调用由合作伙伴提供的服务，服务实现可以是单向或请求-响应操作。
- **Assign**：活动的作用是用新的数据来更新变量的值。Assign 活动可以包括任意数量的基本复制操作，如图 2-4 所示。

```
<bpel:assign name="Assign">
   <bpel:copy>
      <bpel:from> China</bpel:from>
      <bpel:to variable="country"/>
   </bpel:copy>
</bpel:assign>
```

图 2-4 BPEL 程序中 Assign 节点格式

- **Throw**：用来抛出程序的故障和异常。
- **Exit**：用来退出流程的运行。
- **Wait**：活动会暂停流程执行，等待一段给定的时间或等到某一时刻才继续运行。
- **Empty**：活动不执行任何操作。

结构化活动规定了一组活动发生的顺序，描述业务流程怎样把基本活动组成结构而被创建的，这些机构包括业务流程实例间的控制形式、数据流程、故障和外部事件的处理及消息的交换。结构化活动可以被任意嵌套。主要的结构活动包括 sequence，while，switch，flow，pick。

- **Sequence**：定义一段按顺序先后执行的活动。
- **While**：在一个条件满足的情况下反复处理一个活动。
- **Switch**：用于从一组分支情况中选择一个特定的活动分支加以执行。
- **Flow**：用来定义一个或者多个并行执行的活动。
- **Pick**：执行与发生的事件相关联的活动。

（4）**关联集合**：用来指定服务实例中相关联的操作组。一组相关标记可以定义为相关联的组中所有消息共享的一组特征。这一组特征就是关联集合。在 BPEL 程序中有全局关联集和局部关联集。

（5）**事件处理程序**：是指整个流程以及每个作用域都可以与一组相应的事件发生时并发调用事件处理的程序相关联。这种事件包括有两种类型，分别是与 WSDL 中请求或相应的操作对应的消息及用户设置的时间过后发出的警报。

（6）**事务与补偿机制**：补偿不理是为了将流程的状态回滚，回到进入作用域之前一样，将该作用域内已经执行部分采用其它行为进行撤销。补偿处理的调用方法是 compensate。

（7）**异常管理**：异常管理用于在活动执行过程中发生异常，业务流程必须对错误进行处理时，通常通过 faultHandler 来捕获程序异常。

2.1.3 BPEL 故障类型

变异测试是一种基于故障的测试技术[45]。在变异测试中对被测试程序的源代码进行改变，植入错误形成一个含有错误的代码版本，植入的错误称为一个变异体[46]，形成的新的代码版本称为一个变异版本，也就是本文所说的故障版本。

已经提出的BPEL语言的故障类型[47]可以分为四大类，包括标识符替换、活动替换、表达式替换、异常与事件替换[48]。

标识符替换类型中主要有 ISV 类型，ISV 类型用一个相同类型的变量标识符替换另一个。

表达式替换类型包括以下这些子类型，分别是 EAA，EEU，ERR，ELL，ECC，ECN，EMD，EMF。这些故障类型代表的含义如下：

- 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：这个故障类型常用于 wait 或者 pick 活动中，用于将时间表达式的值赋值为 0，或者其他时间。
- EMF：这个故障类型是用来设置截止日期为 0 或者其他时间。

活动替换类型包括两类：与并发相关的替换、与并发无关的替换，其中与并发相关的子类型包括：ACI、AFP、ASF、AIS，其具体含义如下：

- ACI：将 createInstance 属性从“yes”变为“no”。
- AFP：将顺序的 forEach 活动改为并行的，即将该标签的“parallel”属性“no”修改为“yes”。
- ASF：用 flow 活动替换 sequence 活动，如图 2-5，2-6 所示。
- AIS：将一个域的 isolated 属性改为“no”。

与并发无关的子类型包括：AEL、AIE、AWR、AJC、ASI、APM、APA。其具体含义如下：

- 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,假设这两个序列没有关联。
- 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。

```
< sequence >
    <invoke name="checkFlight" . . . >
        <sources>
            <source linkName="checkFlight-To-BookFlight"    Attribute />
        </sources>
    </invoke>
    <invoke name="checkHotel" . . . />
        <targets>
        <target linkName="checkFlight-To-BookFlight" />
        </targets>
    </invoke>
</ sequence>
```

图 2-5 AFP 类型示例原始代码

```
< flow >
    <invoke name="checkFlight" . . . >
        <sources>
            <source linkName="checkFlight-To-BookFlight"    Attribute />
        </sources>
    </invoke>
    <invoke name="checkHotel" . . . />
        <targets>
        <target linkName="checkFlight-To-BookFlight" />
        </targets>
    </invoke>
</flow>
```

图 2-6 AFP 类型示例有故障的代码

异常与事件操作符替换类型包括的子类型有：XMF、XMC、XMT、XTF、XER、XEE。

- XMF：这个故障类型用来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：这个故障类型用来将异常处理中的“faultname”的属性用通类

型的属性替换。

- XER：该故障类型用来移除 rethrow 活动。
- XEE：从事件处理程序中移除 onEvent 元素。

除了这些 BPEL 的故障类型，Antonia 等人又总结了一些新的故障类型[49]，这些故障类型有：CDE、CCO、CDE、CFA、EIN。

- CDE：将布尔表达式转换成返回 true 函数。
- CCO：将布尔表达式转换成返回 false 函数。
- CFA：删除 sequence 节点。
- CDE：将一个节点用一个空节点替代。
- EIN：为变量添加 not()函数。

BPEL 程序中可能存在的这些故障类型，都是在程序设计中可能出现的会影响程序运行的问题。因此我们将这些最容易产生故障的故障类型植入到 BPEL 程序中，用这些故障版本来验证我们基于谓词切换的故障定位技术的有效性。

2.1.4 SOAP 协议

SOAP 以 XML 形式提供了一个简单、轻量的用于在分散或分布环境中交换结构化和类型化信息的机制。用于访问网络服务的协议，它通过提供一个有标准组件的包模型和在模块中编码数据的机制，定义了一个简单的表示应用程序语义的机制。

SOAP 采用了已经广泛使用的两个协议：HTTP 和 XML。HTTP 用于实现 SOAP 的 RPC 风格的传 输，而 XML 是它的编码模式。SOAP 请求是一个 HTTP 的 POST 请求。SOAP 请求的“content-type”属性必须用“text/xml”。而且它必须包含一个请求 URI。SOAP 消息是从发送端到接收端的单向传输，它常常结合起来执行，类似于请求/应答的模式。

SOAP 定义了一套编码规则，该规则定义如何将数据表示为消息，以及怎样通过 HTTP 协议来传输 SOAP 消息，它由以下四部分组成：

- SOAP 信封（Envelope）：定义了一个框架，该框架描述了消息中的内容是什么，包括消息的内容、发送者、接收者、处理者以及如何处理这些消息。
- SOAP 编码规则：它定义了一种系列化机制，用于交换应用程序所定义的数据类型的实例。
- SOAP RPC 表示：它定义了用于表示远程过程调用和应答的协定。
- SOAP 绑定：它定义了一种使用底层传输协议来完成在节点间交换

SOAP 信封的约定。

在本文中需要从一个客户端对整个的 BPEL 流程服务发送 SOAP 消息，其结果也是以 SOAP 消息的格式返回。图 2-7 为 SOAP 消息的格式。

```
<soapenv:Envelopexmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
xmlns:q0="ustb.bpel.org"xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <q0:SmartShelfProcessRequest>
            <q0:name>candy</q0:name>
            <q0:amount>100</q0:amount>
        </q0:SmartShelfProcessRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

图 2-7　SOAP 消息格式

2.2 故障定位技术

1） Tarantula 技术

Jones 提出了一种基于统计的方法—Tarantula 方法来辅助程序开发人员进行故障定位[6]，并实现了该方法的支持工具。Tarantula 方法的核心思想是：对代码中的每条语句，记录测试用例执行时其被覆盖的状况，及当前测试用例是否失效。综合每条语句在失效运行和成功运行中被覆盖的状况（失效运行表示当前测试用例无法检测出故障，成功运行表示当前测试用例能够检测出程序中存在故障），计算出每一条语句的可疑度并根据可疑度的高低排名来帮助用户进行定位故障。这种方法的计算公式是：

$$\mathrm{sus(s)} = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed}+\frac{failed(s)}{totalfailed}} \qquad (2\text{-}1)$$

在公式（2-1）中，passed(s)表示执行某个程序片段 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，得到最可能存在故障的地方进行检查。

该方法的局限性在于，Tarantula 方法需要一个较大规模的测试集，并且至少需要一次通过测试和一次未通过测试。否则当 tatalpassed 和 totalfailed 中某一个为 0 的时候，该方法是无效的，因为公式的分母不能存在 0 的情形。

2） Code-Coverage 技术

Code-Coverage 技术[7][8]的原理是如果这段程序被很多能得到成功的测试用例测试过，那么再被能得到期望输出的测试用例测试可能就没有刚开始的测试用例对于故障定位有那么大的贡献，也就是说如果一段程序已经被成功的执行了 n 次，那么第 n+1 次成功执行的贡献可能比第一次成功执行后第二次再成功执行的贡献要小，可以用下面的公式（2-2）和（2-3）来计算怀疑度：

$$\sum_{i=1}^{n} c_{i,j} * r_i - f(\sum_{i=1}^{n} c_{i,j}) * (1 - r_i) \qquad (2\text{-}2)$$

其中，

$C_{i,j}$=1 时，表示第 i 个测试用例测试了第 j 个程序；

$C_{i,j}$=0 时，表示第 i 个测试用例没有测试到第 j 个程序；

r_i=1 时，表示程序执行第 i 个测试用例与期望输出不符；

r_i=0 时，表示程序执行第 i 个测试用例与期望输出相符。

$$f(k) = \begin{cases} k; & k = 0, 1, 2 \\ 2 + (k - 2) \times 0.1; & 3 \le k \le 10 \\ 2.8 + (k - 10) \times \alpha; & k \ge 11 \end{cases} \qquad (2\text{-}3)$$

其中α是一个很小的数，公式（2-3）是用一个函数来调整对于每一个增加的成功的测试用例对于进行故障定位的贡献。

3） Set-union 与 Set-Intersection 技术

Set-union 技术[9]就是从一个失败的测试用例测试过的语句集中移除掉所有被成功的测试用例测试过的语句，得到一个新的集合，就是原始的怀疑语句。这种方法可以用以下的公式（2-4）来描述

$$E_{initial} = E_f - \bigcup_{p \in P} E_p \qquad (2\text{-}4)$$

Set-Intersection 方法[9]是通过被每个成功的测试用例测试过的语句集中移除掉被一个失败的测试用例测试过的语句得到一个初始的语句集，这个故障定位的技术可以用下面的公式（2-5）进行描述

$$E_{initial} = \bigcap_{p \in P} E_p - E_f \qquad (2\text{-}5)$$

这两种方法统称为集合运算法，该方法认为出现在失败执行轨迹中的程序实体可能存在故障，采用集合的并和交运算可以缩小怀疑的范围[10]。该方法的局限在于，若选取成功测试用例的集合若过大，则会使得交集为空，则

无法参与后续的计算，因此在选取成功测试用例集合的时候应该慎重，尽量去避免交集为空的情形。

4） SOBER 技术

Liu 等人提出基于谓词的 SOBER 故障定位方法[31]。在该方法中，一个谓词 P 的值可以为 true 也可以为 false，假设 P 为 true 的次数为 P_t，P 为 false 的次数为 P_f，则谓词 P 的取值模式为 $w(P)= P_t / (P_t + P_f)$。假设所有成功执行时 P 的取值模式为 $w(P/\theta_t)$，失败执行时 P 的取值模式为 $w(P/\theta_f)$，若 $w(P/\theta t) \neq w(P/\theta_f)$，则 P 是与故障相关的，而且 $w(P/\theta_t)$与 $w(P/\theta_f)$差异越大，P 是故障的可能性就越大。谓词 P 的怀疑度计算公式如 2-6 所示：

$$\mathrm{SOBER(P)} = log_a\left(\frac{\sigma_p}{k*e^{-\frac{Z^2}{2}}}\right) \qquad (2\text{-}6)$$

其中，底数 a=|R|≠0.1，建议取值 e 或者 10；方差$\sigma_p^2 = \frac{1}{n-1}\sum_{i=l}^{n}(x_i - u_p)^2$；期望$u_p = \frac{1}{n}\sum_{i=1}^{n} x_i$；$x_i$是谓词 P 第 i 次成功执行时取真值的概率；系数$k = \sqrt{\frac{|T_f|}{2\pi}}$，$Z = \frac{Y-u_p}{\sigma_p/\sqrt{m}}$，$Y = \frac{1}{m}\sum_{i=1}^{m} y_i$，$y_i$是谓词 P 第 i 次失败执行时取真值的概率。

5）Ochiai 技术

Abreu 等人引入分子生物学领域的相似系数“Ochiai”来定义语句的怀疑度计算公式如 2-7 所示：

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalfailed * (failed(s) + passed(s))}} \qquad (2\text{-}7)$$

公式中 totalfailed 表示所用的测试用例集中所有失败运行的测试用例，failed(s)表示经过程序语句 s 且运行失败的测试用例的个数，passed(s)表示经过程序语句 s 且运行成功的测试用例的个数。该方法只考虑语句在成功执行的轨迹和失败执行的轨迹中的频率，而没有考虑语句和执行结果之间的关系。

5) Crosstab 技术

Wong 等人提出了一个良好的故障定位技术 Crosstab[51]。该方法通过统计成功执行与失败执行状态下，程序语句s分别被覆盖的次数，并构建该语句的交叉表，交叉表如表 2-1 所示。进而对s计算统计量$\chi^2(s)$，如公式 2-8 所示，其中 $ECF(s) = \frac{NC(s)*NF}{N}$，$EUF(s) = \frac{NU(s)*NF}{N}$，$EUS(s) = \frac{NU(s)*NS}{N}$，$ECS(s) = \frac{NC(s)*NS}{N}$。$\chi^2(s)$表示语句$s$与测试结果的关联程度，在给定显著性水平α下进行假设检验，可以判断语句s是否独立于测试结果。为了求s的统计量$\chi^2(s)$，引入了如下参数，如表 2-2 所示。由于关联程度会随着样本数量增多而上升，为了方便度量程序语句 s 的怀疑度，需要给语句数量与测试用例数量及各个

程序语句的相关系数$\mathrm{M(s)}=\frac{\chi^2(s)/N}{\sqrt{(row-1)(col-1)}}$，其中 row 代表交叉表的总行数，列代表交叉表的总列数，(row-1)实则是测试用例的个数，(col-1)实则是程序语句的规模。语句 s 的怀疑度计算公式如 2-9 所示，其中$\varphi(\mathrm{s})=\frac{PF(s)}{PS(s)}=\frac{\mathrm{NCF(s)/NF}}{NCS(s)/NS}$。

$$\chi^2(s)=\frac{(NCF(s)-ECF(s))^2}{ECF(s)}+\frac{(NCS(s)-ECS(s))^2}{ECS(s)}+\frac{(NUF(s)-EUF(s))^2}{EUF(s)}+\frac{(NUS(s)-EUS(s))^2}{EUS(s)} \tag{2-8}$$

$$\xi(\mathrm{s})=\begin{cases} M(s),\ if\varphi(s)>1 \\ 0, \qquad if\varphi(s)<1 \\ -M(s),\ if\varphi(s)=1 \end{cases} \tag{2-9}$$

表 2-1　crosstab 方法交叉表

	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	r
t1	1	0	0	0	1	1	0	1	1	1	1
t2	1	1	0	1	0	0	0	0	1	0	0
t3	1	0	1	0	0	1	1	1	1	1	1
t4	1	1	1	1	0	0	1	1	1	0	0
t5	1	1	1	1	0	1	1	1	1	0	0
t6	1	0	0	0	0	0	1	1	1	1	0
t7	1	1	0	0	0	1	1	0	1	1	0
t8	1	1	1	1	0	0	1	1	1	0	0
t9	0	1	0	0	0	1	0	0	1	1	0
t10	1	0	0	0	1	0	1	1	1	1	1
t11	0	1	1	0	0	0	0	1	1	0	0

表 2-2　crosstab 方法各参数表示意义

N	测试用例的总数
NF	运行失败的测试用例的总数
NS	运行成功的测试用例的总数
$\mathrm{NC}(s)$	经过语句 s 的测试用例的总数
$\mathrm{NCF}(s)$	经过语句 s 运行失败的测试用例的总数
$\mathrm{NCS}(s)$	经过语句 s 运行成功的测试用例的总数
$\mathrm{NU}(s)$	未经过语句 s 的测试用例的总数
$\mathrm{NUF}(s)$	未经过语句 s 运行失败的测试用例的总数
$\mathrm{NUS}(s)$	未经过语句 s 运行成功的测试用例的总数

6)　Liblit 技术

Ben Liblit 等人提出了基于谓词覆盖的故障定位技术，该技术假设程序中存在某个谓词的取值与其后某个故障的出现有着必然联系[52]。例如，一段程

序的谓词 P 的真分支上存在故障 F，假分支上没有错误，那么该程序只有在谓词取真时，故障 F 才会被发现。该技术在程序分支、函数返回及标量位置进行插桩，执行测试用例，捕获谓词的覆盖信息，谓词 P 的怀疑度计算公式为公式 2-10 所示。

$$\text{importance(s)} = \frac{2}{\frac{1}{\frac{F(P_t)}{S(P_t)+F(P_t)} - \frac{F(P_c)}{S(P_c)+F(P_c)}} + \frac{1}{\frac{\log(F(P_c))}{\log(|T_f|)}}} \tag{2-10}$$

其中 P_t 表示 P 取值为真，$S(P_t)$和$F(P_t)$分别表示 P 取值为真时程序成功执行和失败执行的次数；$S(P_c)$和$F(P_c)$分别表示 P 在成功和失败执行中被覆盖的次数。

7) SAFL 技术

Hao 等人提出了相似度故障定位技术 SAFL，该技术基于概率理论计算程序语句与失败测试用例和所有测试用例的关联程度[53]。假设语句覆盖信息和测试结果用执行矩阵$\mathrm{E} = (e_{ij})$来表示。其中e_{ij}表示测试用例 t_i 时，语句s_j的执行信息，被覆盖时为 1，否则为 0。矩阵 $F=(f_{ij})$,其中 f_{ij} 表示语句s_j对测试用例t_i的贡献程度，公式 2-11 所示。

$$\mathrm{F} = (f_{ij}) = \begin{cases} \frac{1}{\sum_{k=1}^{m} e_{ik}}, & if\ e_{ij} = 1 \\ 0, & if\ e_{ij}! = 1 \end{cases} \tag{2-11}$$

$$\mathrm{SALF}(s_j) = \frac{\sum_{k=1}^{m} \max(f_{ik} | e_{ij}>0 \wedge e_{i\,(m+1)}=0)}{\sum_{k=1}^{m} \max(f_{ik} | e_{ij}>0)} \tag{2-12}$$

语句s_j在失败执行和所有执行中的贡献度的比值表示故障怀疑度，比值越大，语句s_j出错的可能想就越大，其比值用 SALF 表示，如 2-12 所示。

2.3 国内外研究现状

本节从三个方面来介绍了故障定位的国内外研究现状，主要包括基于程序切片的故障定位方法、基于统计的故障定位方法及基于谓词的故障定位方法。

2.3.1 基于程序切片的故障定位方法

如何有效的调试程序一直是国内外专家学者探索的问题。Gould 曾经说过，很多的程序员在调试信息的时候往往是从程序的开头开始去读这些代码，而忽略了从程序的错误输出着手考虑[10]。Dijkstra 及其他学者指出调试的时间应该在严格关注程序正确性的基础上得以缩短，提出了通过寻找感染变量

的方式来锁定程序的故障[12]。这种调试思想在当时被称作“后向工作选择”。在此基础上，Weiser 于 1979 年在他的博士论文中首先提出了程序切片的概念，主要通过寻找程序内部的相关特性，来达到分解程序的目的[13]。此后，又有很多的专家学者对程序切片给予了不同的定义，并且提出了有关切片的不同算法。Ottenstein[14]提出了有关程序切片的算法，引入了程序依赖图(Program Dependence Graph)的概念。在 1990 年，Horwitz[15]提出了系统依赖图(System Dependence Graph)的概念，解决了 Weiser 所提出的切片算法中存在的无法解决过程调用的问题。

Korel 和 Laski 等人提出了动态切片的概念，并将动态切片分为后向动态切片和前向动态切片。在对后向切片的研究中[16]，Agrawal 和 DeMillo 提出了一种定位模型，希望通过变量分组的方式进行切片的处理来缩小故障定位的范围[17]。如果错误语句出现的位置比较靠近程序的结尾，那么其后向切片的规模也较大[9]。针对此问题 Gupta 等人提出了错误诱导切片的排查方法，旨在减小与错误代码相关的切片的范围[20]。

Zhang，Korel 等人将切片经过整理，划分为数据切片[23]、完全切片[24]和相关切片[25] 。Zhang 等人[26]对数据切片、完全切片和相关切片三种动态切片方法进行了试验评估，分析比较了各种切片的大小和正确定位故障的能力。曹等人提出了一种将动态切片、关联分析及排序策略相互结合的一种故障定位方法[29]。

2.3.2 基于统计的故障定位方法

Jones 提出了一种基于统计的方法—Tarantula 方法来辅助程序开发人员进行故障定位，并实现了该方法的支持工具 Tarantula[6]。Hong 等提出了一种基于简化运行程序的统计错误定位方法[27]。该方法首先通过聚类减少冗余执行路径，然后计算简化后的失败和成功执行路径之间差异的统计特征，对其进行排序来定位故障。Chen 等提出了 Jacard 故障定位方法，使用了聚类分析中的系数[28]。Abreu 提出了一种称为“Ochiai”的故障定位方法，引入分子生物学领域的相似系数“Ochiai”来定义语句的怀疑度[30]。

惠等人提出了基于程序特征谱覆盖统计的整数溢出故障定位方法，旨在解决整数溢出的错误问题。通过获取分支覆盖特征谱信息和定义使用对特征谱覆盖信息的方法，计算分支可疑度和定义使用对的怀疑度，计算错误传播率，从而求出语句怀疑度之间的关系来定位故障[33]。贺等人提出了一种称为 Muffler 的技术，Muffler 使用程序变异分析来修正错误代码定位结果，以提高定位的准确率，该实验设计的核心思想是核心想法是利用程序变异分析来

降低偶然性成功测试用例的影响，但是错误语句所依赖的算法对故障定位的影响很大[32]。

2.3.3 基于谓词的故障定位方法

谓词是在程序中起判断作用的程序表达式，强行改变谓词的布尔值来改变其路径的选择，这种操作叫做谓词切换。Liu 等人提出基于谓词的 SOBER 故障定位方法。在该方法中，一个谓词 P 的值既可以为 true 也可以为 false，假设 P 为 true 的次数为 P_t，P 为 false 的次数为 P_f，则谓词 P 的取值模式为 $w(P)= P_t/(P_t+P_f)$。设所有成功执行时 P 的取值模式为 $w(P/\theta_t)$，失败执行时 P 的取值模式为 $w(P/\theta_f)$，若 $w(P/\theta_t)\neq w(P/\theta_f)$，则 P 是与故障相关的，而且 $w(P/\theta_t)$ 与 $w(P/\theta_f)$差异越大，P 是故障的可能性就越大[31]。Zhang 等人提出了关键谓词的概念，在谓词切换过程中，对于相同的测试用例，若切换后的程序的输出结果与预期结果一致，则该谓词为关键谓词[25]。

2.4 研究背景与意义

张等人于 2006 年提出了基于谓词切换的故障定位技术。该技术通过依赖关系寻找程序中的谓词，通过对谓词的切换来获取关键谓词。然后寻找关键谓词的后向切片来进行故障的定位。该方法的优点在于：

(1) 基于谓词切换的故障定位方法按照通过谓词切换寻找关键谓词，减少了程序的故障定位范围，降低了故障定位的难度。

(2) 基于谓词切换的故障定位技术不依赖于任何的怀疑度计算公式，排除了相关公式中参数的影响。例如 Code-Coverage 方法易受到 α 值的影响。

(3) 将基于谓词切换的方法应用于 BPEL 程序的故障定位中，可以有效利用 BPEL 程序的结构特点，快速获取程序中的谓词，并进行有效的谓词切换，从而提高 BPEL 程序的故障定位效率。

鉴于该方法的以上优点，本文尝试将这种故障定位技术应用在 BPEL 程序的故障定位中，期望能够有效的提高 BPEL 程序的故障定位效率。

2.5 研究内容与成果

课题组在前期的研究工作中已经取得了一些成果[36][37]，主要是将基于频谱的故障定位技术用于 BPEL 程序的故障定位中，这种方法的主要思想是在把程序划分为多个块的基础上，通过大量的测试用例来计算每一个语句块的怀疑度，并按怀疑度从高到低进行排序，怀疑度越高的语句所包含的故障的

能力就越强[35]。实验表明基于频谱的故障定位技术的定位效率仅达到 50%。

本文通过将基于谓词切换的故障定位技术应用于三个实验示例，并将该技术作用于三个实验示例的故障定位效率、故障定位精度与 Tarantula 及 Code-Coverage 技术的定位效率与精度进行了对比。实验表明：基于谓词切换的故障定位技术在 BPEL 程序中的故障定位过程中表现了较大的优势，其故障定位的效率达到 90%，远远高于 Tarantula 技术的 50%，而定位的精度也保持在 10%以内，其他技术则在 20%到 50%之间。

2.6 论文组织结构

本文的组织结构安排如下：

第1章， 引言。

第2章， 背景介绍，主要包括相关概念及技术，国内外研究现状，研究内容及成果和研究意义。

第3章， 讨论基于谓词切换的关键技术，包括对技术的原理描述，切换规则的定义，一般方法步骤和举例说明。

第4章， 讨论基于谓词切换的原理进行支持工具设计与实现问题，包括需求分析，总体设计，实现中的关键问题以及系统演示。

第5章， 对基于谓词切换技术的可行性与有效性进行实例研究，包括实验问题阐述、实验目标程序的说明、实验一般步骤及实验结果分析。

第6章， 研究工作的总结以及未来的展望。

3 基于谓词切换的 BPEL 程序故障定位技术

本章主要介绍基于谓词切换的基本原理以及一般过程，并以一个例子展示该方法的使用。

3.1 故障定位的原理

故障定位的前提是至少有一个测试用例能够证明该程序存在故障。在前期的工作中，对 BPEL 程序植入各种类型的故障，产生多个故障版本。同时利用多次随机产生测试用例的方法获取多个测试用例，并将这些用例输入应用于每一个故障版本，得到相应的输出结果，并将该结果与预期输出进行对比，若结果不同，说明该测试用例能够检测出程序中存在故障，并记录该测试用例集合为 Ts，作为后续谓词切换过程中的输入用例集。

在执行 Ts 中的每一个测试用例的过程中能够通过引擎获取所有的执行路径集合，记为 $Pa=\{a_1, a_2, a_3, \dots, a_n\}$，并能够通过获取的执行路径来获取所有的谓词集合，记为 $Pr=\{r_1, r_2, r_3, \dots, r_n\}$。为了提高故障定位的效率，本文对谓词做了排序的处理（在 3.2 节详细介绍）。

在谓词切换的过程中，每次对 r_i 的布尔值进行修改，强行改变其执行分支，并将修改后产生的新的 BPEL 文件进行重新部署和执行，然后收集其输出结果，将该输出结果与预期结果进行对比，若不一致则继续切换下一个谓词，否则则确定当前切换的谓词为关键谓词，记为 Cp。

在确定关键谓词后需要从程序中去搜索和关键谓词相关的变量或者常量。在搜索的过程中遵循这样一种规则：若 Cp 与 Cq 相关，Cq 与 Cr 相关，则 Cp 与 Cr 相关（Cq，Cr 代表程序中的常量或变量）。这些相关变量所在的语句块将是故障怀疑度最高的块。根据 BPEL 程序的特点，在获取到怀疑度较高的块时，若该块为非原子节点时则可以深入到块的内部搜索和变量相关的原子节点。这样可以进一步缩小故障定位的范围。

根据谓词切换的原理，本文绘制了基于谓词切换的 BPEL 程序故障定位的原理图，其内容如图 3-1 所示，该原理图主要包括三部分，分别是 BPEL 程序解析、谓词切换和切片分析。BPEL 程序解析的目的是为用户呈现当前 BPEL 程序的流程结构，谓词切换的目的是为了找到关键谓词，切片分析的目的是为了找到故障所在位置的集合。图 3-2 详细描述了谓词切换的流程，其中 p_i 表示第 i 个故障版本，$Bp_{i,j}$ 表示第 i 个故障版本的第 j 个谓词进行切换后的 BPEL 版本。

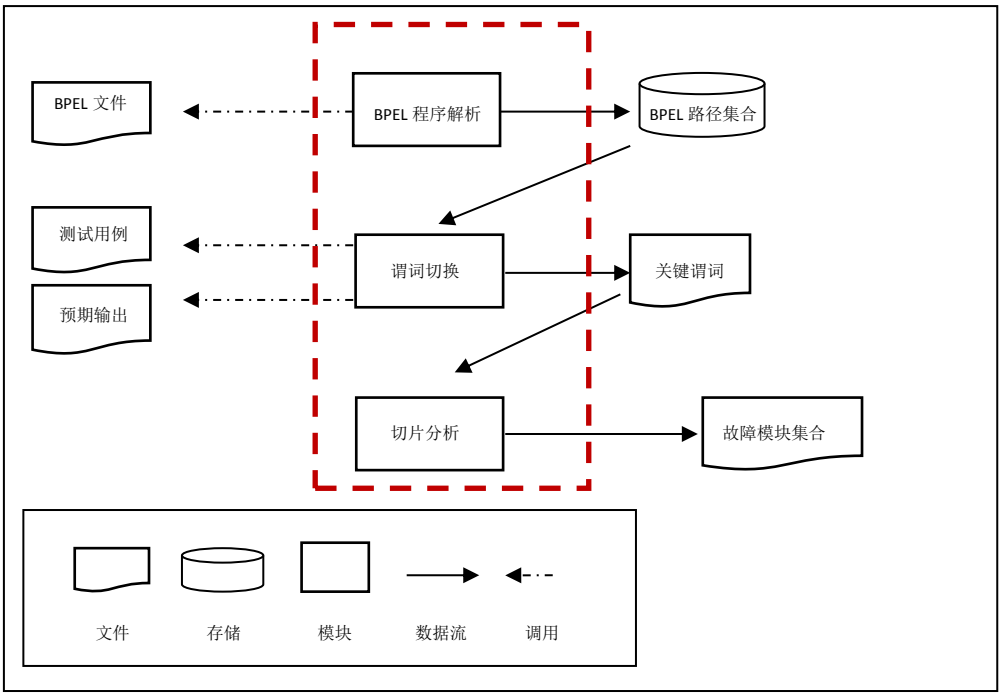

图 3-1 基于谓词切换的 BPEL 程序故障定位原理图

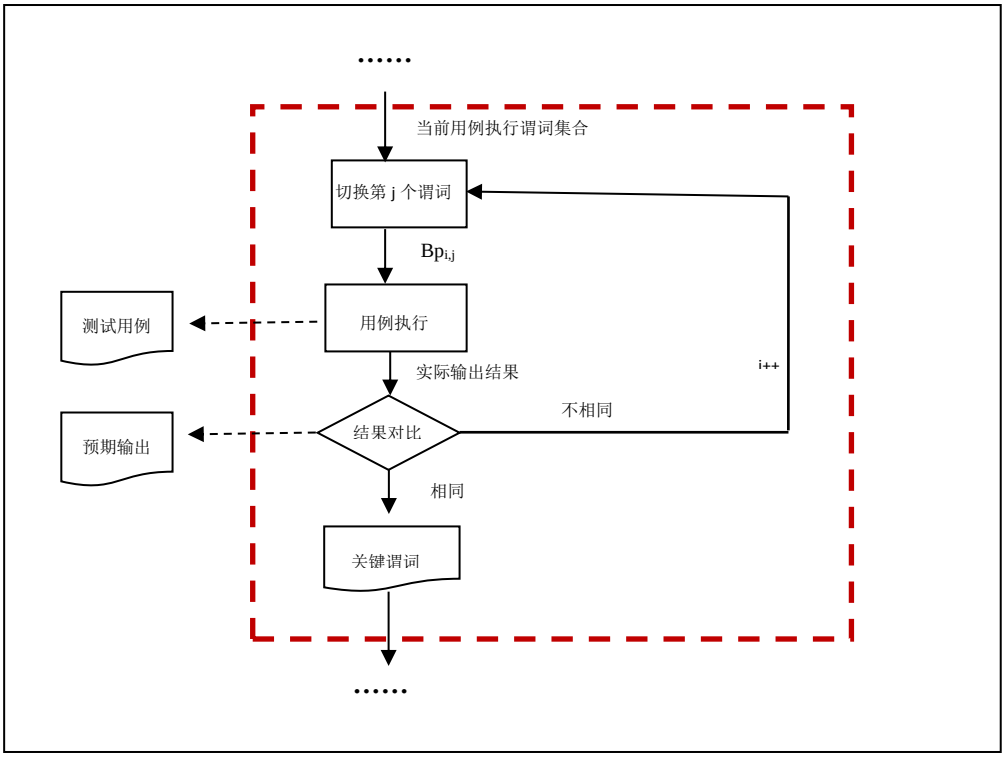

图 3-2 谓词切换基本原理

3.2 BPEL 程序解析

在一个程序中，所有为了产生输出而进行计算的程序语句都可以被分为两类：一类是数据切片(DP)[25]，另一类为选择切片(SP)。

数据切片一般用于变量的声明与计算，不参与到程序分支的判断。而选

择切片是对程序分支进行选择的判断，选择切片依赖于数据切片的计算，对程序的分支起判定作用，通常存在于 if, switch, while 等语句中。程序的输出结果极大的依赖于程序中的选择切片，而选择切片极大的依赖于数据切片的计算，因此，通过从选择切片着手不断缩小故障切片的搜索范围是非常必要的。在 BPEL 程序中，数据切片和选择切片的划分如图 3-3 所示：

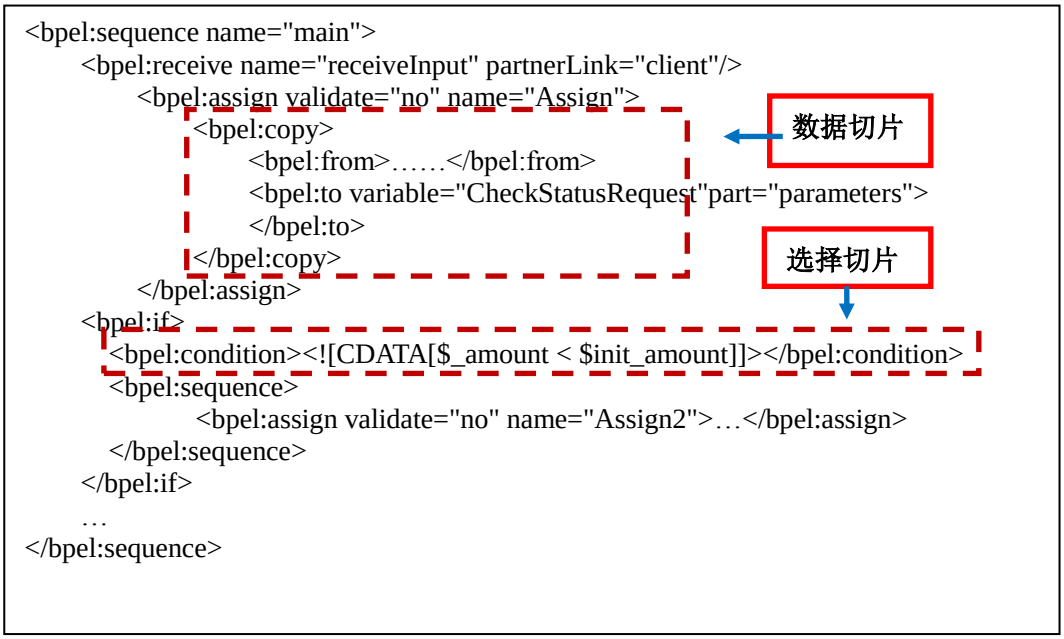

图 3-3 BPEL 程序中 SP & DP 的示例

在本文中所有的选择切片都可以称为谓词。在谓词切换之前需要对谓词集合中的元素进行排序，目前主要有两种策略：一种策略遵循最后执行的谓词最先切换的(简称 LEFS 策略)排序，另外一种策略遵循最后执行的谓词最后切换（简称 LELS）策略。

策略一：LEFS 排序策略是根据谓词距离错误输出的距离来进行排序的，这种排序方法基于一种观察，即：错误的代码通常离错误输出总是比较近。因此可以根据距离错误输出的远近对所有谓词进行倒序，这样得到的序列就是经过 LEFS 方法处理之后的顺序了，假设 $P = \{p_1, p_2, p_3, \ldots, p_n\}$，其中 p_n 距离错误输出最近，那么利用 LEFS 排序后的谓词的顺序为 $P'=\{ p_n , p_{n-1} , p_{n-2} , \ldots , p_1 \}$。

策略二：LELS 策略，其排列规则与 LEFS 的排列规则相反，首先是从距离错误输出距离最远的地方进行谓词切换，依次进行，直到离错误最近的谓词。假设 $P = \{p_1, p_2, p_3, \ldots, p_n\}$，其中 p_n 离错误输出的距离最近，那么利用 LELS 排序后的谓词顺序为 $P'=\{ p_1 , p_2 , p_3 , \ldots , p_n \}$。本文分别运用两种谓词排序策略做了实验，并且分析了不同排序策略的定位效率和精度。

3.3 BPEL 程序执行过程

BPEL 程序的执行依赖于 Apache ODE 引擎，ODE 引擎的系统架构的关键模块包括 ODE BPEL 编译器、ODE BPEL 运行时、ODE 数据访问对象（DAOS）。在 BPEL 程序的编译过程中，BPEL 引擎会将 BPEL 源文件转化成一个编译好的、适合于引擎执行的流程表示形式，即 BPEL 对象。该对象包括 BPEL 文档中的命名引用（例如变量名）、所需要的 WSDL 文件及类型信息等。

当用户向 BPEL 服务发送消息时，消息会以 SOAP 格式被 BPEL 服务接收，接收的消息会立即赋值给“ReceiveInput”块中定义的参数，“ReceiveInput”块是 BPEL 流程开始的入口。在收到 SOAP 格式的消息后，选择哪条执行路径是通过 ODE 引擎的 Jacob 框架来实现的，Jacob 提供了一个持久虚拟机来执行 BPEL 结构，在执行过程中需要某个活动时，当前线程会通过数据访问对象（DAOs）在数据库中选择所需要的活动，并将最终的结果返回。

综上所述，用户在调用 BPEL 服务的过程经历了三个阶段，首先是 BPEL 程序在引擎上的部署，也是 BPEL 程序的编译过程。然后，对已经部署的服务发送消息，通过引擎集成的 AXIS2 服务封装为 SOAP 格式，传送给当前流程的实例，也就是当前的 BPEL 服务。最后，Jacob 为当前实例提供执行环境并通过 AXIS2 服务将结果返回给用户。

3.4 结果对比

结果对比模块的主要功能是将当前故障版本的实际输出与预期输出进行对比，并检测该故障版本的输出与预期输出是否一致，如果一致，则说明当前切换的谓词是关键谓词，否则的话继续进行下一个谓词的切换。本文中所有的故障版本的输出结果是以文件的形式进行保存。

3.5 谓词切换过程

如何对谓词进行切换是本文的一个重点内容。首先，在 3.1 节获取 Ts 的过程中已经得到其执行路径和谓词集合；然后，利用 LEFS 方法对谓词集合进行排序，根据排序后的谓词顺序逐个修改每个谓词的布尔值，并将新生成的 BPEL 故障版本重新部署和执行，通过对比实际输出与预期输出来确定关键谓词。图 3-4 介绍了寻找关键谓词的过程。

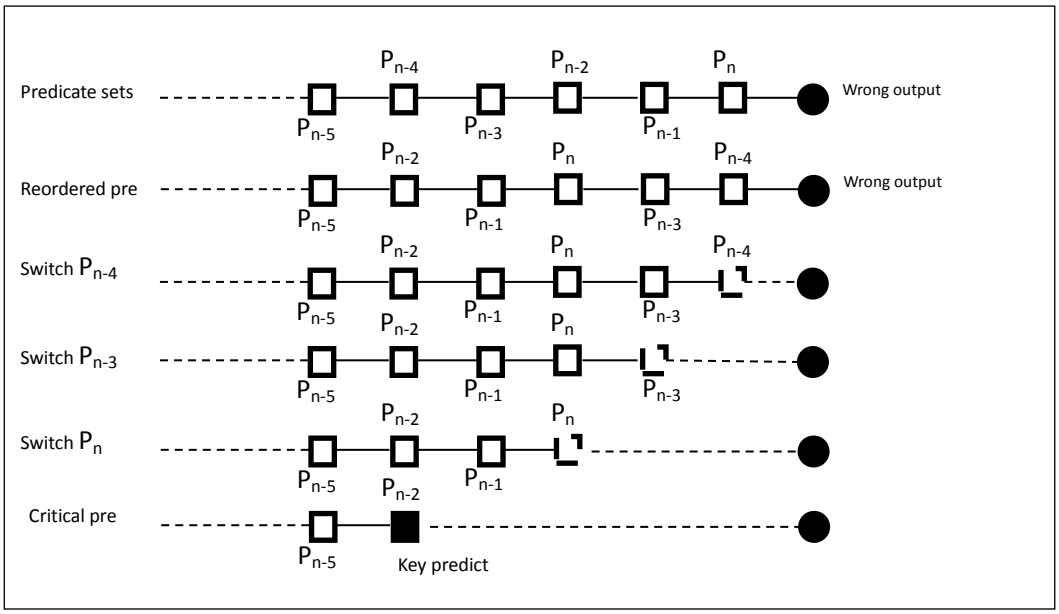

图 3-4 谓词切换过程

例如：在图 3-4 中，假设存在谓词的集合 p={p_1 , p_2 , p_3 , … , p_n }。第一，根据 LEFS 排序方法对其进行排序 p'=={ p_n , p_{n-1} , p_{n-2} , … , p_1 }。第二，当 p_n 为 false 的时候，将其修改为 true，然后重新部署修改后的 BPEL 文件，并发送相同的测试用例并观察修改后的输出结果与预期输出是否一致，如果不一致则进行下一个谓词的切换，如果一致则找到关键谓词，结束该切换过程。第三，在确定关键谓词以后，需要在程序中寻找和关键谓词相关的后向切片，后向切片所在的位置是故障怀疑度最高的位置，以此方法来获取故障所在位置的集合。具体的步骤可以详细划分为如下：

步骤 1：对事先拟好的测试用例进行输入，并记录每一个测试用例 t_i 的实际输出结果及通过的执行路径所对应的块 Bps={ s_1 , s_2 , s_3 , … , s_n }。

步骤 2：将每一个测试用例 t_i 的实际输出结果与预期输出进行比较，即可得出每一个测试用例的“pass”或者“failed”状态。

步骤 3：对于所有的“failed”状态的测试用例的集合 Cs= {c_1 , c_2 , c_3 , … , c_n }中的每一个测试用例 c_i，获取执行路径中的谓词集合 Pres={ p_1 , p_2 , p_3 , … , p_n }∈Bps,分别更改每一个谓词 p_i 的布尔值，强制改变判断分支，并记录下该结果的“pass”与“failed”状态，在执行完 p_i 后会出现两种情况：

（1） 执行完 p_i 后，结果的状态为“pass”，就找到了关键谓词。

（2） 执行完 p_i 后，结果的状态为“failed”，则继续寻找下一个谓词。

步骤 4：根据关键谓词来寻找后向数据切片，并根据后向数据切片来确定故障语句块的集合。

其算法过程如图 3-5 步骤 4 所示：

Fault location algorithm for BPEL program using the predicate switching:
INPUT:
P:{$block_i$| 1<=*i*<*n*, and the type of $block_i$ }
TS:{t_i|*1*<*i*<=*m*, t_i is a test case of test suite for *P*}.
EXP:{v_i'|*1*<*i*<=*m*, v_i is the base node of the block}.
PPS: the possible output of each test case.
OUTPUT:
Loc:The possible error location.
PROCEDURE
1. set *loc* to *Φ*
2. input the test case and record the block={b_1,b_2...b_i... b_n} (n∈Z)
3. Fliter the predicates from the *block*, pre={b_i,..., b_{i+n}}∈block
4. FOR EACH *predicate* ∈*pre*,
   IF(*Arraylist[pre]*)
      Result=Switch predicate(*predicate*);
      IF(*Result* is critical predicate)
         {
            Criticalpre=predicate;
         }
      END IF
   END IF
5. Find backward slice about the *Criticalpre*, rel={v_1,v_2,v_3,...,v_n};
6. FOR EACH v_i in *rel*
   IF (v_i *!= exp(* v_i*')*)
   {
      Return *Loc(*v_i*);*
   }
  END FOREACH
END

图 3-5 基于谓词切换应用于 BPEL 程序故障定位算法

3.6 切片搜索与分析

根据关键谓词可以搜索出和谓词相关的变量、常量以及运算关系。目前的思路是从关键谓词的位置向程序的起始位置寻找相关变量，直到寻找到“Receiveinput”节点处，因为该节点是 BPEL 程序执行的起始节点。

程序中的依赖关系主要有数据依赖和选择依赖，在后向切片的搜索过程中需要利用这两种依赖关系，其搜索的步骤如图 3-6 所示：

步骤 1：找到与关键谓词相关的变量集合 V={v_1, v_2 , v_3 , … , v_n } (参数 V 中的元素不仅包括变量也包括常量)；

步骤 2：寻找所有的与 V 相关的后向切片 Bv={b_1, b_2 , b_3 , … , b_n} (参数 Bv 所包含的变量都直接或者间接的与 V 中的元素相关)；

步骤 3：检测 Bv 中的元素所在的块，若该语句块为非原子节点则深入到块内部去搜索和谓词相关的原子结点，这些原子结点的集合将是故障怀疑度最高的块；

图 3-6 搜索与谓词相关切片步骤

3.7 方法示例

SmartShelf 的 BPEL 程序总共被分成了 53 个块，主要涉及到的外部服务有 11 个，分别是“SetupTime”，“ReadStatus”，“ReadInformation”，“CheckStatus”，“ CheckLocation ”，“ CheckQuantity ”，“ SendstatusTowarehouse ”，“RearrangeProduct”，“WarehouseManager”，“AlertStaff”，“AlertCarrier”。在 SmartShelf 的 BPEL 服务中需要三个参数，分别是“name”，“amount”，“status”，其中前两个是需要用户根据自己的需要来输入的，而第三个参数是通过“ReadStatus”服务从数据库中获取的。每一个测试用例从“Receiveinput”进入，最终会从其中的某一个分支结束，程序的结果将从“ReplyOutput”模块中输出。

SmartShelf 服务在接收到输入的参数值之后，会根据流程的部署去执行不同分支，调用不同的服务。服务会自动根据输入的产品的名称（name）和数量（amount）来查询其状态（status），并且根据其状态值来判断该产品是否过期，若过期需要被替换掉，替换完成后返回“Expired produce has been replaced”，否则返回“Goods are in good status”；根据产品的名称将其查询到的在货架上的位置和产品本身所应该在的位置进行对比，如果不同则表明产品位置错误，需要重新整理该商品在货架上的位置并返回“the location is correct now”，若位置正确直接返回“location is right”；根据产品的数量来回货架上的该商品数量对比，如果不足，则会与仓库中的货物数量对比，若仓库货物充足，则将仓库的货物搬运一部分到货架，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”，如果货架上的商品数量可以满足需要，则返回“Qulitity is sufficient”。SmartShelf 中的 BPEL 流程图如图 3-7 所示。

在实例中，以{name: “candy” , amount : “100”}做测试用例，该测试用例的执行路径为 Path = {1-6, 7-10, 16-17, 18-21, 27-28, 29-42, 48-53}(其中的数字代表块编号)。然后从该路径中找到谓词的集合 Pre={10, 21, 32, 37},也就是说在该测试用例的执行路径上存在四个谓词。接着对四个谓词进行排序，排序有两种方法可供选择，分别是 LEFS 策略和 LELS 策略（详见 3.2 节解释）。假设切换后谓词的顺序是 Pre’={10, 21, 32, 37}，如表 3-1 所示。

第一次切换谓词“$_status=0”，将其修改为“$_status！=0”，发现其预期输出与实际输出不一致，则进行下一个谓词的切换。在第二次切换中将“$_location=0”切换为“$_location!=0”，并且其输出结果与预期结果一致，因此“$_location=0”就是关键谓词，这个谓词与程序的错误有着极大的关联。在寻找到本次的关键谓词以后，需要分析出该谓词相关的变量，与当前关键谓词相关的变量是“_location”，而整个程序中与“_location”相关的变量有“CheckLocationPLResponse”，“CheckLocation”，“Receiveinput”，这些相关的变量存在的节点都可能是故障可能存在的节点。

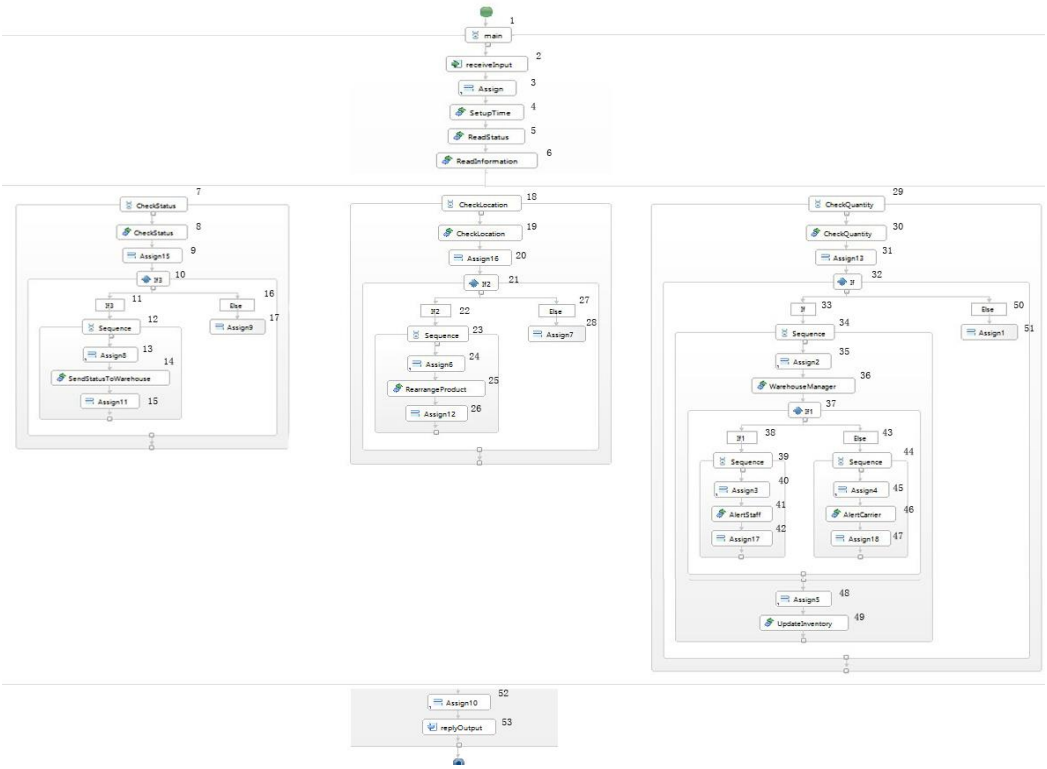

图 3-7　SmartShelf 实例的 BPEL 流程图

表 3-1 谓词排序后的状态集

If3(10)	If2(21)	If(32)	If1(37)
$_status =0	$_location =0	$_amount<$init_amount	$warehouseManagerReturn<$init_amount

3.8 小结

本章主要介绍了故障定位的原理，并设计了故障定位的框架，该框架主要包括 BPEL 程序的解析，BPEL 程序的执行，结果的对比，谓词的切换及切片的搜索与分析，最后通过 SmartShelf 实验示例来解释了如何将故障定位技术应用于 BPEL 程序的故障定位中。

4 基于谓词切换的 BPEL 程序故障定位工具设计与实现

本章讨论基于谓词切换的 BPEL 程序故障定位工具的设计与实现，包括需求分析、工具结构设计以及工具实现中的关键问题，最后用一个实例验证并演示该工具。

4.1 需求分析

基于谓词切换的 BPEL 程序故障定位工具(BPEL_PSLocator)面向的主要用户是 Web 服务开发人员，其目的是辅助完成 BPEL 程序的故障定位，提高开发的效率。

为了尽可能符合用户需求，期望 BPEL_PSLocator 能够达到以下要求：

（1） 工具系统的操作尽量符合用户的输入输出习惯，尽量减少用户输入的次数，取而代之以鼠标选择的方式，比如用下拉框代替文本框进行选择。

（2） 工具界面直观易懂，全面考虑各种输入产生的各种异常输出，并对此做出各种异常处理。

（3） 体系结构符合逻辑，功能完整、正确，具有可扩展性，易于修改和维护。

图 4-1 是 BPEL_PSLocator 工具的用例图。根据用户的需求 BPEL_PSLocator 的主要功能包括：BPEL 程序的解析、用例执行、结果对比、谓词切换和切片分析。

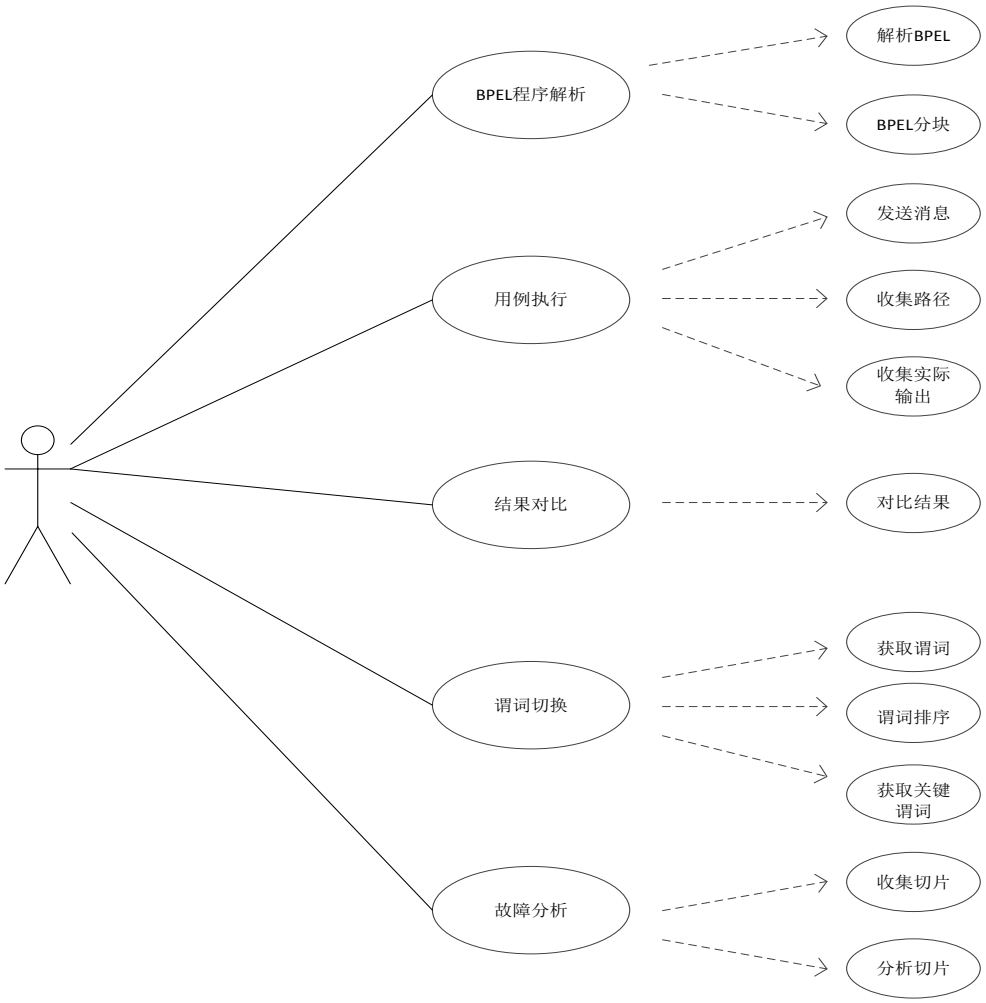

图 4-1　系统用例图

1）**BPEL 程序解析**：该用例用于 BPEL 程序的分析，包括解析 BPEL 和 BPEL 分块两个用例。

- **解析 BPEL**：该用例用于将用户导入的 BPEL 进行解析，将其结构展示给用户。
- **BPEL 分块**：根据 BPEL 程序的模块划分标准对解析后的 BPEL 程序进行模块的划分及编号的分配。

2）**用例执行**：该用例主要用于 BPEL 程序的执行，包括消息的发送、路径的收集、以及实际结果的捕获。

- **发送消息**：主要向正在运行的 BPEL 程序发送 SOAP 消息。
- **收集实际输出**：主要是捕获当前发送的 SOAP 消息的返回结果。
- **收集路径**：主要用来获取当前执行测试集的路径。

3）**结果对比**：该用例主要是将当前测试集的结果和预期结果进行对比，目的是发现程序中是否有错误。

4）**谓词切换**：该用例主要是针对已经发现了错误的程序，根据用例执行中收集路径用例所得结果得到所有谓词，并逐个切换谓词，寻找出关键谓词，

再根据关键谓词进一步分析错误所在的位置。该用例包括获取谓词、谓词排序、获取关键谓词三个部分。

- **获取谓词**：获取当前测试集的路径，并从中找到经过的谓词节点。
- **谓词排序**：对获取的谓词进行排序。
- **获取关键谓词**：对排序好的谓词依次进行切换，直到找到关键谓词。

5）**故障分析**：根据关键谓词，对 BPEL 程序中的数据流进行分析，最终定位出错误，该用例包括收集切片和分析切片。

- **收集切片**：获取和关键谓词相关的变量的程序切片。
- **切片分析**：对切片进行分析，找出最可能发生故障的程序切片。

4.2 系统结构设计

该工具的设计主要来源于基于谓词切换的故障定位的思想，更细致的功能划分如下：BPEL 程序解析功能、用例执行功能（环境部署功能、发送消息功能、ODE 消息获取功能）、结果对比功能、谓词切换功能、故障分析功能及界面展示功能。

第一，在工具中首先需要对 BPEL 程序进行解析，以此来划分出不同的模块，因此在 BPEL 程序解析模块中主要提供两种服务，第一种是 BPEL 程序的解析功能，另外一种是根据 name 属性值获取任意谓词节点并修改其条件值的功能。

第二，在用例执行中包括环境部署功能、发送消息功能、ODE 消息获取功能，接收返回消息，获取执行路径等功能。ODE 引擎可根据新的 BPEL 文件自动进行流程的部署。用户对当前 BPEL 服务发送消息时，需要通过 AXIS2 包对输入的用例进行 SOAP 格式的封装。执行完毕后，BPEL 流程服务将最终的结果以 SOAP 格式返回给用户。

第三，在谓词切换环节，每次需要将当前 BPEL 程序中的某个谓词进行切换，即通过改变其布尔值来强行改变当前测试用例的执行路径。每次切换后会产生新的 BPEL 文件，需要重新部署，重新执行并获取服务返回的结果，通过与预期输出来对比判断当前切换的谓词是否为关键谓词。

第四，结果对比模块和谓词切换是相互承接的，每次切换执行完毕需要将当前 BPEL 版本的实际输出结果与预期输出结果进行对比，以帮助判断当前的谓词是否为关键谓词，因此在工具的设计中需要添加结果对比这一个类来提供对比的功能。

第五：当获取到关键谓词后需要进行切片的分析，在 BPEL 程序中寻找和当前谓词相关的变量，并通过跟踪变量值的变化来获取可疑的故障模块。

第六：在使用工具的过程中用户不可能永远在控制台进行输入输出，而且在控制台输入输出没法体现整个工具的完整性，因此给该工具加入了界面展示层，希望用户能够通过该层看到切换的数据。

根据 4.1 的需求分析，设计了 BPEL_PSLocator 的系统结构图，如图 4-2 所示：

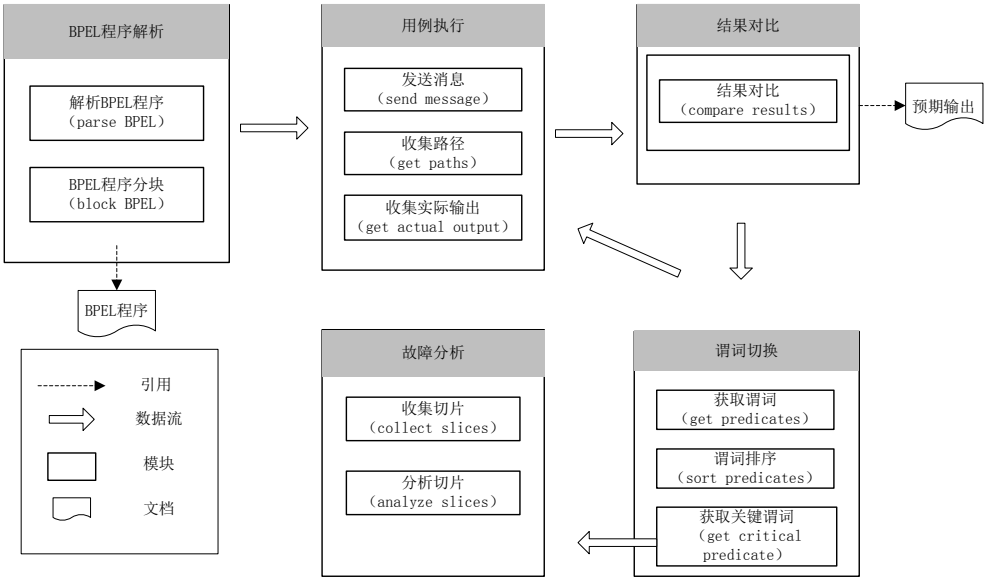

图 4-2　BPEL_PSLocator 系统结构图

系统主要包括五个大类，其中最重要的步骤是如何进行消息的交换，消息的交换包括从消息的发送到消息的解析，直到最终呈现给用户的过程。下面主要介绍一下在系统中各个类是如何相互协作实现消息的交换的，如图 4-3 所示。

图 4-3 给出了 BPEL_PSLocator 中消息的发送到消息的获取、解析，以及预期输出与实际输出对比的过程。首先在 messageFrame 上有 addEventListener()的方法来监听用户的操作 Messagesend 类中的 messagesend()方法来向 BPEL 服务发送消息，同时通过 getMessage()方法来获取 BPEL 服务返回的消息，并调用 OmelementParse 类中的 parseOmelements()来进行返回消息的处理，返回的消息是一个 Omelement 类型，包括两个方法来获取自身的属性，分别是 getLocalName()和 getText()，当对解析完毕的 Omelement 消息返回给用户时，还需要将该消息与预期输出进行对比，用户可以看到本次执行的 BPEL 程序的实际输出是否与预期输出一致。

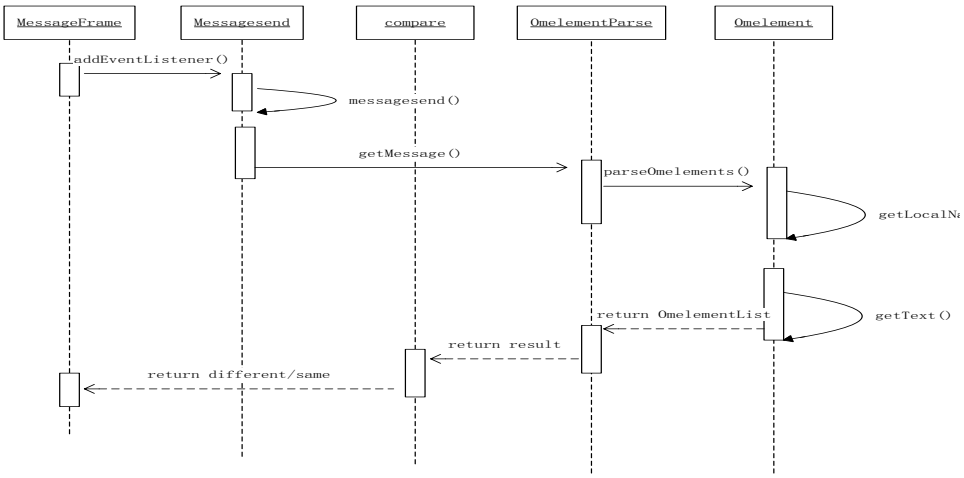

图 4-3 BPEL_PSLocator 消息交换处理图

4.3 工具实现关键问题

本节介绍在实现 BPEL_PSLocator 过程中的一些关键问题。包括两个方面。

（1）数据库的初始化问题

对于需要向数据库写入文件的 BPEL 程序，需要进行数据库的初始化。为了保证数据库里的数据不影响程序的输出结果，每一次切换过程中首先应该对数据库进行初始化，保证程序需要从数据库读取的初始值保持一致，从而排除因为数据库中数据差异而带来的结果差异。

（2）Apache ODE 引擎的支持

BPEL 流程编译的最后一个步骤是将流程作为 Web 服务，同时将该服务所有操作（Operation）都绑定一个消息接收器，这里的消息接收器就可以认为是引擎与 AXIS2 之间的桥梁，通过它来将 AXIS2 接收到的消息转发到引擎内部。在引擎系统中所有的消息可以分为两种，一种是由外部客户端发送过来的，而另一种是引擎内部自己作为流程执行所需而传递的消息。对于外部消息可以采用 AXIS2 系统中的 MessageContext 类作为消息表示。

在流程执行过程中，所有的消息应该采用内部封装的格式，利用事件触发的形式进行流程的执行。Apache ODE 引擎执行 BPEL 程序的原理图如图 4-4 所示。

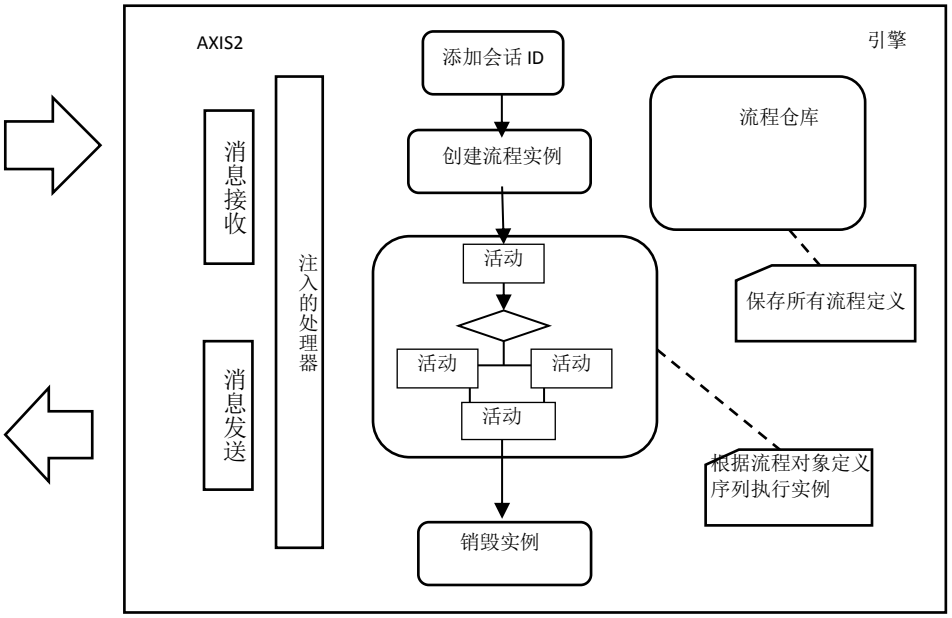

图 4-4　Apache ODE 引擎执行 BPEL 程序的原理图

所有流程的都是以活动为单位，引擎在运行的过程中会经过该测试用例的所有路径，路径是以活动的形式表现。因此，对引擎中的代码进行插桩，可以捕捉到运行过程中的状态。对引擎的修改主要针对运行时模块，包括服务调用、赋值、路径捕获、结果捕获。根据 BPEL_PSLocator 工具需求，对引擎做了如下修改：

- “ACTIVITY 类”是引擎运行时模块中处理 BPEL 节点的类。通过对该类进行插桩来获取当前测试用例执行过程中的路径节点，并根据相应的数据结构对节点进行解析，从而分析出相应的节点“name 属性值。
- “ASSIGN 类”是引擎运行时模块中负责赋值的类，跟踪所有的值，并对返回值进行解析，获取到每一个变量每一次变化的值。
- “INVOKE 类”是引擎运行时模块中负责外部服务调用的类，对“INVOKE 类”中的“request”和“response”对象进行解析，可以获取每一次服务调用的返回值。

4.4 系统演示

BPEL_PSLocator 工具采用 Java 语言进行开发。下面详细介绍一下它的实现。

1）采用 Java swing 作为图形化界面开发技术作为与用户进行交互的界面。界面设计了用户向导，逐步知道用户的操作。

2）BPEL 程序的解析主要采用了深度优先的策略，逐步解析出所有的节点及

组合路径，并以 JTree 的方式将所有的模块展示给用户。

3）在服务的部署和执行阶段主要依赖于 Apache ODE 引擎自动部署 BPEL 服务的特点，只需在服务器启动的状态下去修改相应的服务文件夹下的部署文件即可。

4）相关数据的获取，是通过对引擎源码的修改来进行捕获的，目的是实现对测试用例的路径和 BPEL 程序中变量值的变化进行抓取，并参与后续的分析过程，详细的修改过程见 4.3 节。

下面我们以 SmartShelf 这个例子作为输入来演示 BPEL_PSLocator 工具。

第一步：在图 4-5 界面中首先要输入 BPEL 程序的地址，BPEL 文件是本次调试过程中的故障版本文件，该文件中有且仅存在一个故障，是本次调试需要发现的目标。第二步，选择测试用例的文件，该文件中只含有一个能够检测出错误的测试用例，是与该故障版本所对应的用例。该测试用例实际上是从众多的测试用例经过检验能够发现故障的一个测试用例，前序的检验步骤不在该工具设计范围之内，接着通过“deploy”按钮和“startserver”按钮进行部署并启动服务器，“parse”按钮提示本次部署的 BPEL 文件的结构和路径组合。图 4-6 显示了当前的 BPEL 程序的模块结构。本次试验所用的服务器是 Apache Tomcat6.0，在文本框中会有提示文字，告诉用户部署完毕，启动服务器后在 Windows 环境下回弹出 Tomcat 的显示窗口。

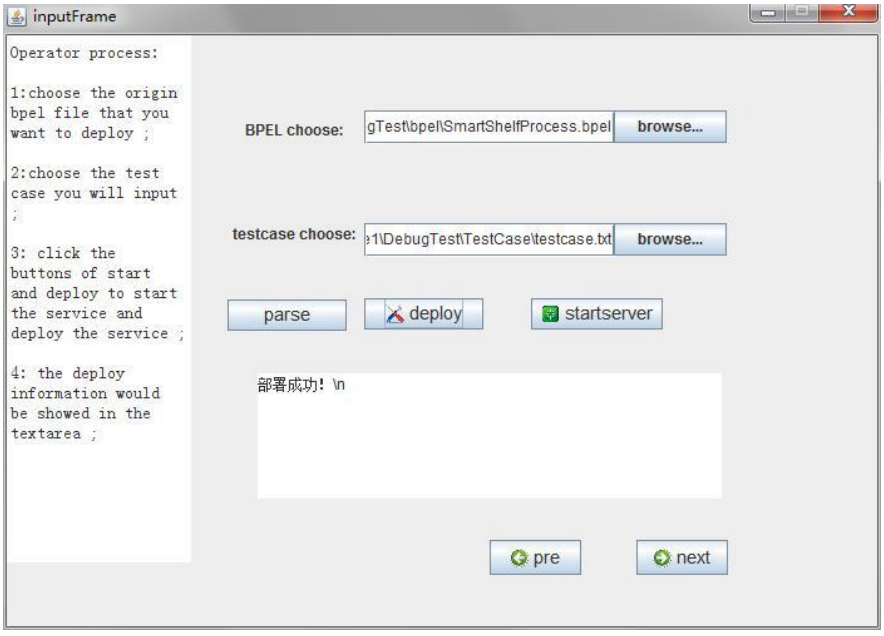

图 4-5　BPEL_PSLocator 输入界面

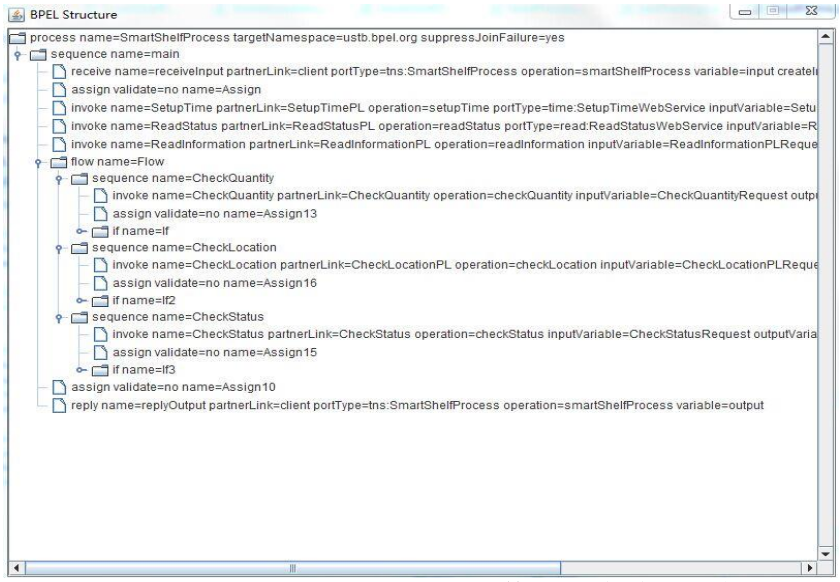

图 4-6　BPEL_PSLocator 展示当前 BPEL 程序的结构

第二步：当服务器启动完成后，用户可以点击“next”按钮进入图 4-7 所示界面并点击“send msg”按钮，将上一步输入的测试用例封装为 SOAP 的形式发送给 BPEL 服务，在界面中的文本框中会有提示显示服务器端返回的消息，该用例的原始输出结果会被保存在文件中，该步骤完成后点击“next”进入到图 4-7 的操作。

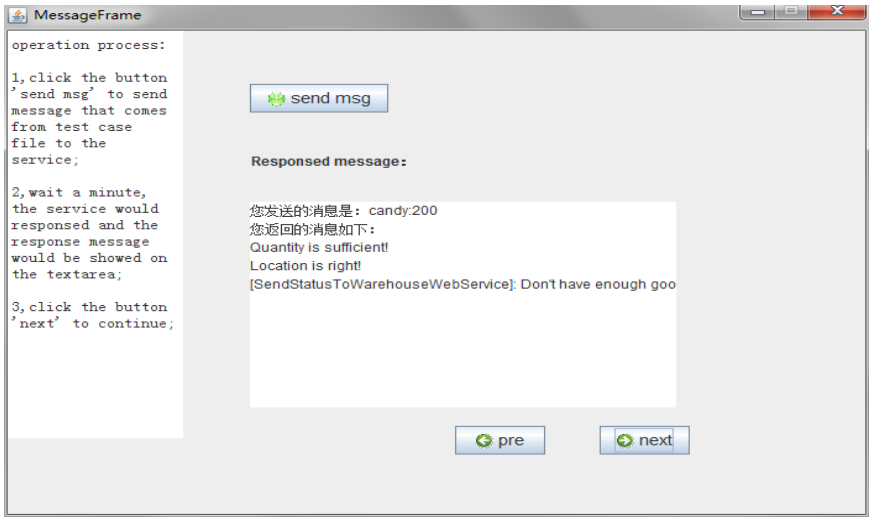

图 4-7　BPEL_PSLocator 向 BPEL 服务发送消息

第三步：在图 4-8 所示的操作界面中输入“expect output”的路径，该路径代表预期输出结果的文件地址。然后选择“check”按钮，来对比实际输出与预期输出有何差异。目前有“same”和“different”两个选项。然后点击“next”进入图 4-9 所示界面。

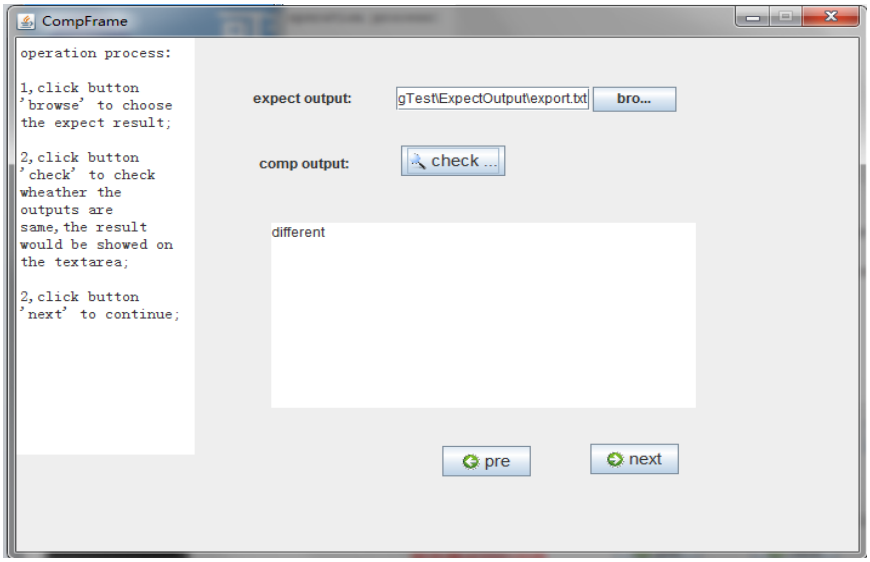

图 4-8　BPEL_PSLocator 结果比对界面

第四步：在图 4-9 中通过点击“details of this test case”按钮可以看到此过程中测试用例通过的所有的 BPEL 路径节点，从该路径中解析所有的谓词集合，同时利用 LEFS 策略对谓词进行排序。

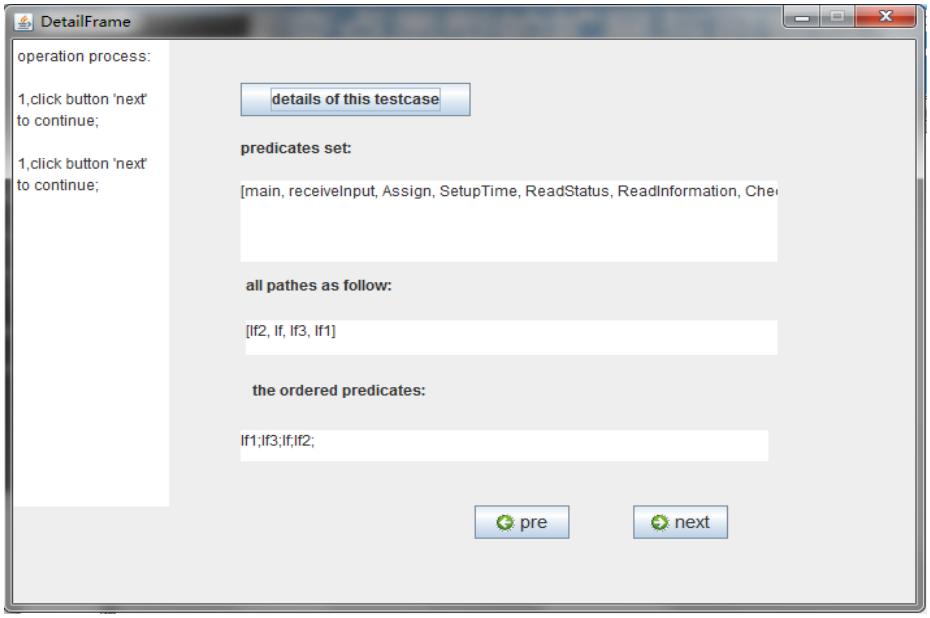

图 4-9　BPEL_PSLocator 路径及谓词获取界面

第五步：在图 4-10 中列出了所有的谓词及其内容，在每一个谓词对应的位置分别有“switch”和“checkinfo”按钮，点击“switch”按钮后可以查看到该谓词的所有信息（如图 4-11 所示），点击“checkinfo”按钮可以查看本次谓词切换后的输出结果（如图 4-12 所示）。

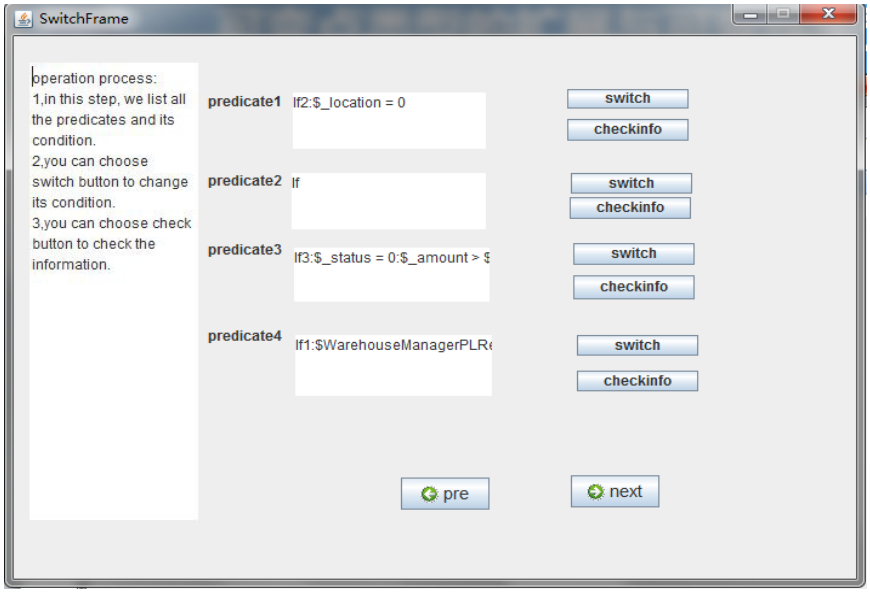

图 4-10　谓词切换界面

第六步：在图 4-11 中用户可以修改谓词的内容，修改谓词的布尔值，然后点击“deploy”按钮重新部署，并点击“send msg”发送同样的消息给服务器，服务器在执行完毕之后将结果存入文件中，文本框中会显示本次测试用例执行的返回结果。点击“back”按钮，回到第五步页面（如图 4-10 所示）。

第七步：回到图 4-10 的界面，用户可以点击“checkinfo”按钮来查看本次切换的详细信息（如图 4-12，4-13 所示），同时点击“compare”按钮可以查看该次切换后的输出与预期输出有没有差异。若对比结果为“same”，则在以下的选项中选择“yes”，否则选择“no”，出现“same”时代表寻找到关键谓词。

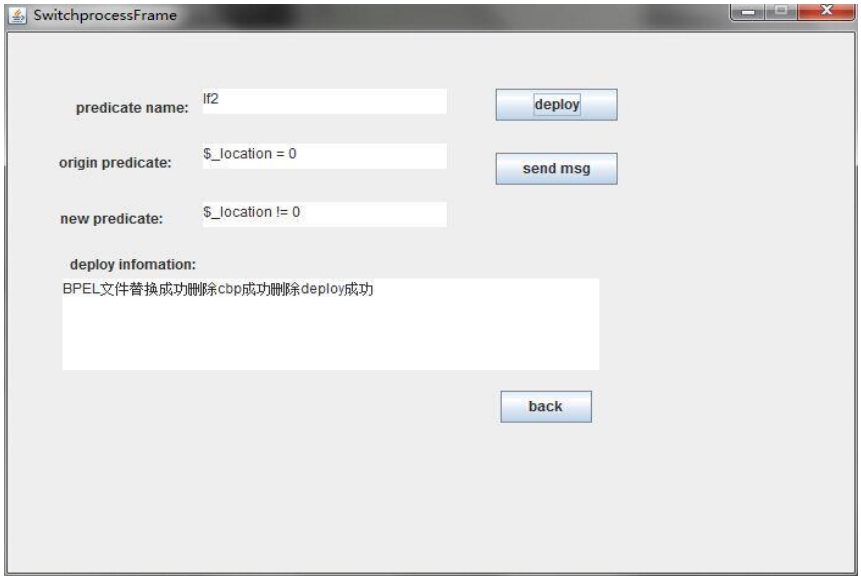

图 4-11　谓词切换过程界面

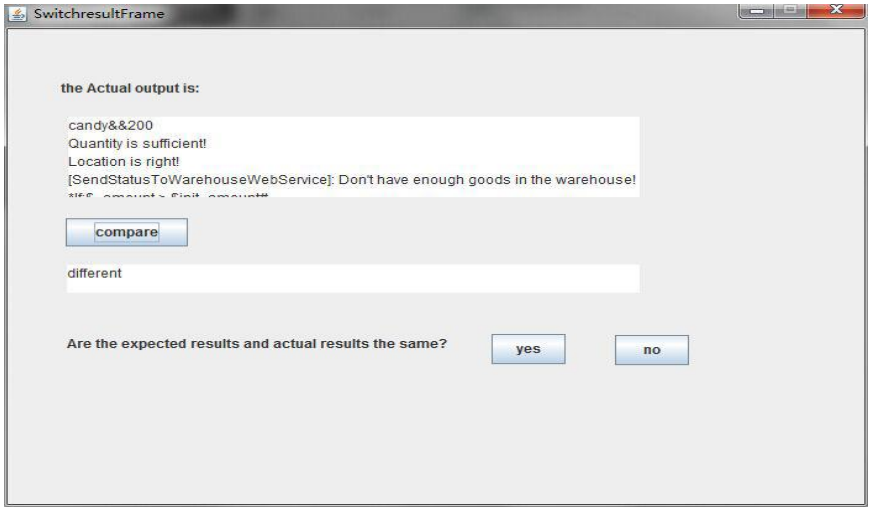

图 4-12　切换后输出及结果对比（不同）

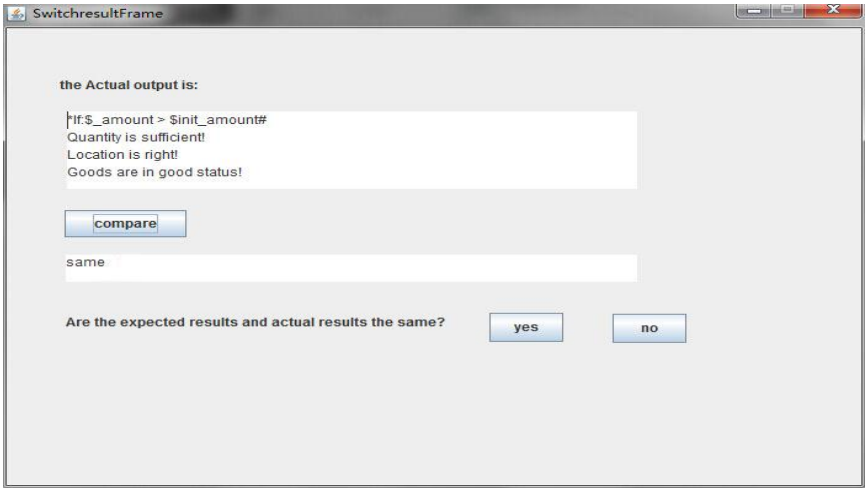

图 4-13　切换后输出及结果对比（相同）

第八步：在图 4-14 中用户可以看到本次循环所找到的关键谓词的详细信息，包括谓词的节点名称，谓词的内容，然后选择“next”，可以看到分析结果。

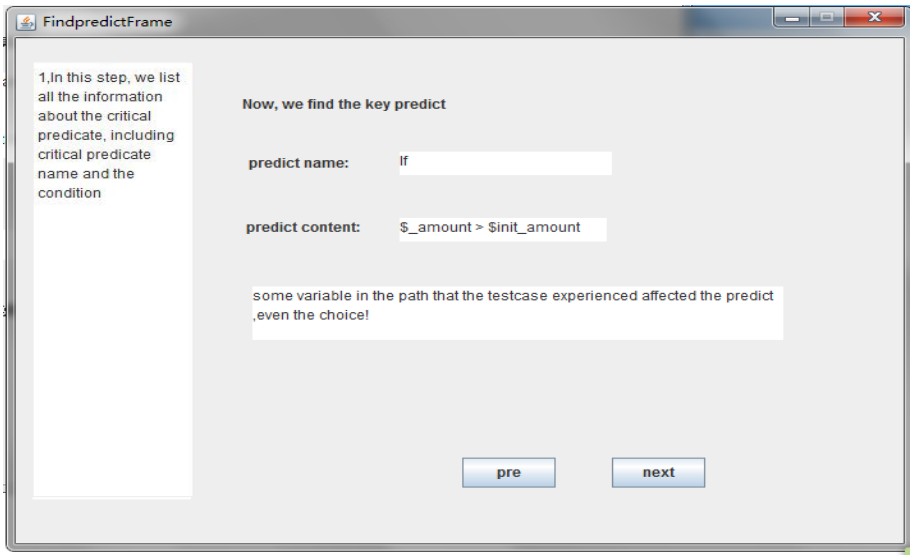

图 4-14　找到关键谓词

第九步：在图 4-15 中用户可以直接在界面中看到所有的结果信息，包括对谓词中变量的分析处理，后向切片的获取以及定位节点信息。

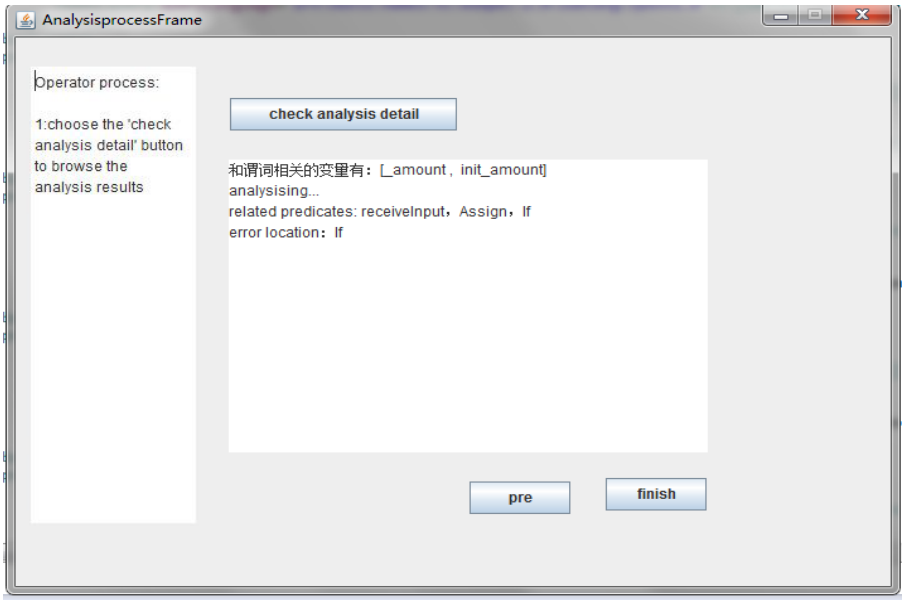

图 4-15　显示分析结果

4.5 小结

本章介绍了基于谓词切换的 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，系统结构及工具演示。支持工具进一步提升了 BPEL 程序的故障定位的自动化程度。

5 实例验证

本章采用 SmartShelf、TravelAgency 及 QuoteProcess 三个实例来验证基于谓词切换的故障定位技术，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收集，最后对统计的结果进行分析，主要从故障定位的效率、故障定位的精度及影响因子三个维度来进行了综合的分析。

5.1 实验实例

5.1.1 SmartShelf

在 SmartShelf 实例的详细介绍见 3.7 节，其 BPEL 流程图如图 3-7 所示。该流程首先通过“Assign”赋值节点、“setupTime”和“readStatus”两个服务来完成相关变量的初始化。然后采用一个并行的结构，来调用“CheckStatus”、“CheckLocation”、“CheckQulitity”三种服务。这三种服务都需要参数输入，所有参数都通过 BPEL 服务的“ReceiveInput”块分别传送给“CheckStatusRequest”、“CheckLocationRequest”、“CheckQulitityRequest”，这样调用的这三种外部服务就可以获取用户的输入参数进行正常的运行，外部服务的返回值分别是“CheckStatusResponse”、“CheckLocationResponse”、“CheckQulitityResponse”，这时流程会以赋值的形式将这些返回值赋给 BPEL 程序中的某些变量，参与流程中的其他计算。

BPEL 程序中的变量皆以块的形式存在，获取变量的赋值，寻找程序中的依赖关系皆要通过节点的解析来获取。

5.1.2 TravelAgency

TravelAgency 例子是一个用来模拟旅游中的各种活动选择，包括方案选择、酒店预订，机票预订等选择。TravelAgency 在获得参数之后会根据参数的大小来选择不同的旅行方案，然后在“flow”块中并行选择酒店预订和航班预订选择方案。该服务的流程如图 5-1 所示：

在 TravelAgency 的 BPEL 程序中，每一个测试用例从“Receiveinput”块进入，最终会从其中的某一个分支结束，程序的结果将从“ReplyOutput”块中输出。和 SmartShelf 实例一样，该程序也是是通过块来进行故障的定位。整个程序共划分为 18 个块，定位的结果是程序中故障可以度较高的块集合。

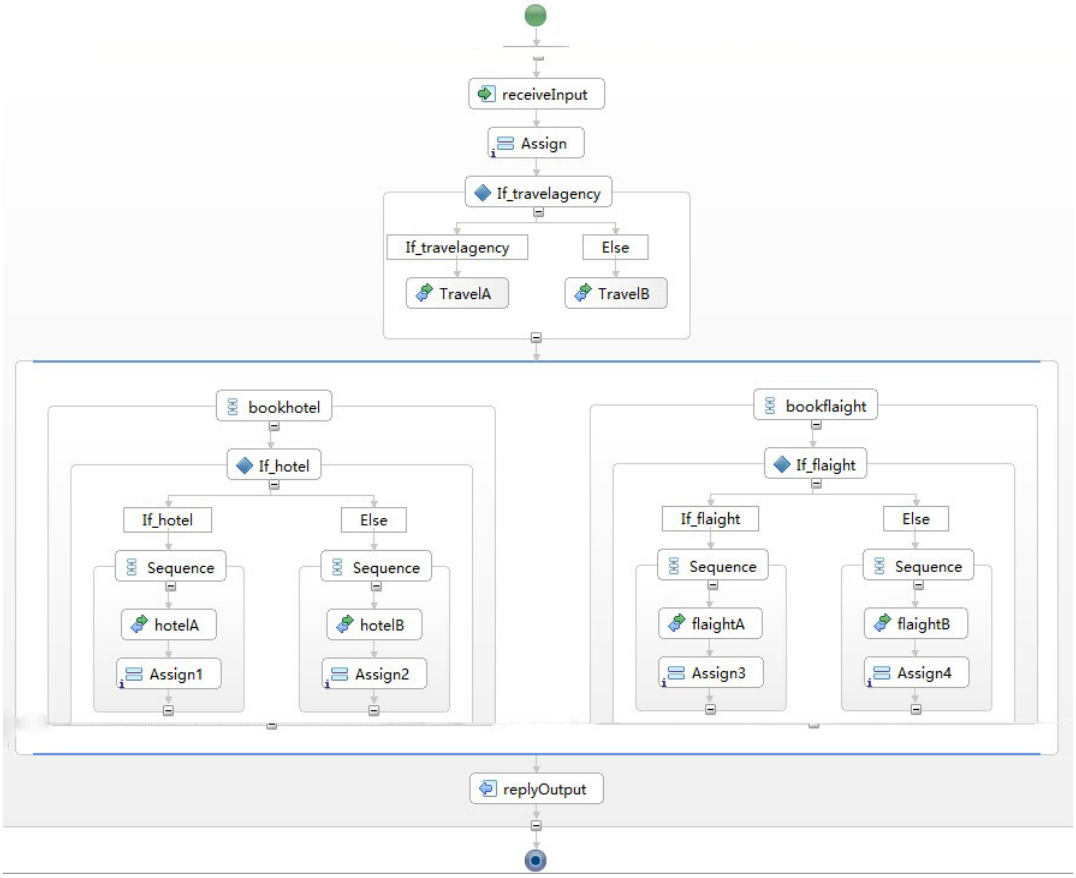

图 5-1　TravelAgency 实例的 BPEL 流程图

5.1.3 QuoteProcess

QuoteProcess 例子是一个用来模拟用户活动选择，主要根据用户输入的值来进行活动的选择。QuoteProcess 在获得参数之后会根据参数的大小来选择不同的评估方案，然后“flow”块中并行选择普通评估和专家评估方案。该实例的流程如图 5-2 所示：

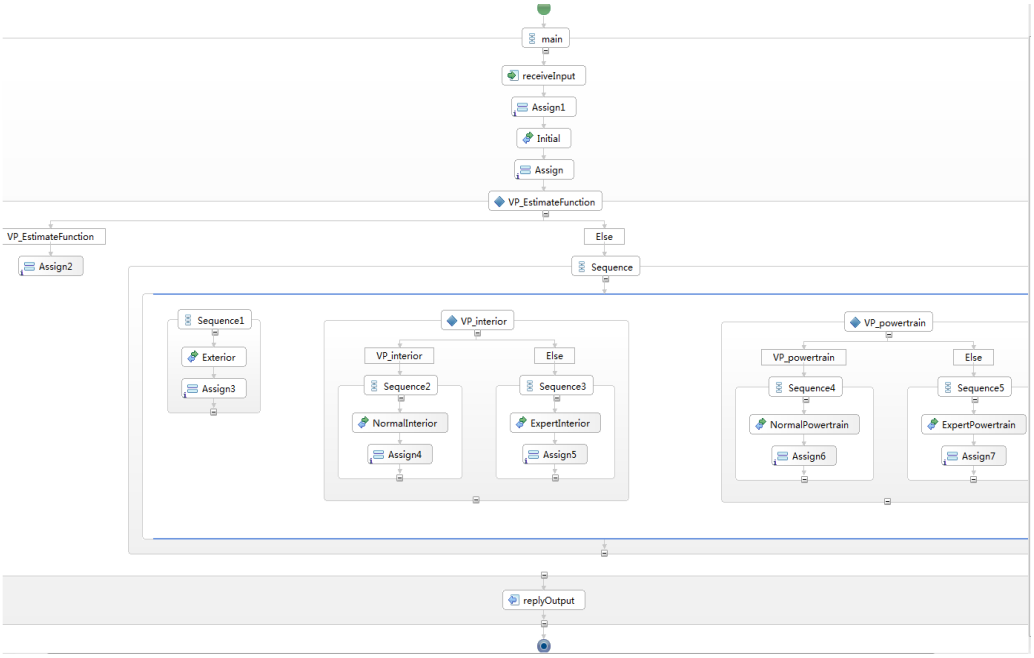

图 5-2　QuoteProcess 实例的 BPEL 流程图

在 QuoteProcess 的 BPEL 程序中，“Receiveinput”、“Assign1”、“initial”块主要用来对程序中的变量进行初始化，“Assign”块用来将用户的输入转为为 BPEL 服务的参数。程序的结果将从“Receiveoutput”块中输出。该程序共划分为 21 个块，定位的结果主要是确定故障所在的块集合。

5.2 实验设计

在本次实验中，三个实验示例都运行在 Windows 环境中，除了 SmartShelf 调用了 11 个外部服务外，TravelAgency 和 QuoteProcess 分别调用了 6 个外部服务。对于三个实验示例的具体描述如表 5-1，其中 Program 表示程序名称，Line of Code 表示程序代码行，Mutants 表示故障版本数量，Block 表示程序分块数量：

表 5-1　实验示例描述

Program	Line of Code	Mutants	Block
SmartShelf	733	103	38
TravelAgency	429	92	18
QuoteProcess	590	83	21

三个实验的实例，包括 278 个故障版本，所有的实验流程的设计和统计均以该实验结果为基础。本节设计了实验步骤，包括故障版本的选择，测试用例的筛选和实验示例的选择。

（1）故障版本的生成：针对每一个实验示例，我们设计了相应的故障版

本，每一个故障版本有且仅存在一个错误。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，主要包括 26 种故障类型，根据实验示例，主要总结了 13 种错误类型，其中包括：EEU、 ERR、 ELL、 ECN、 ACI、 ISV、 ASF、 AIE、 EIN、 CFA、 CDE、 CCO、 CDC。所有的故障版本是由 mubpel 工具来完成[49]。

（2）错误测试用例的筛选：在寻找到可以检测出故障的错误测试用例之前，大量随机的测试用例将被输入，直到发现某个测试用例的输出与预期输出不相符合时即可确定该测试用例可以检测出故障版本中存在的错误，并将该测试用例保存下来，作为后续故障定位的测试用例。

（3）执行引擎：所有的故障版本最终都运行于 Apache ODE 引擎上，为了获取故障版本运行时期的数据，引擎中的源代码将被修改，主要的目的是在引擎运行中能够捕捉到变量值的传递信息，并且捕捉到 BPEL 程序运行过程中的路径节点集合。

（4）数据统计：在获取所有的故障定位效率及定位效率之后，统计所有的数据，并以图形的形式来展现其定位效率的高低及定位精度的大小。

5.3 实验结果及分析

本节主要从故障定位的效率，故障定位的精度及影响因子三个维度来分析多种技术的有效性，从中探测出基于谓词切换的故障定位技术的优势。

5.3.1 故障定位效率

SmartShelf、TravelAgency、QuoteProcess 的定位效率依次见表 5-2，表 5-3，表 5-4，其中“P(b)”表示基于谓词切换的技术选择 LEFS 策略时的程序实例定位效率，“P(f)”表示基于谓词切换的技术选择 LELS 策略时程序实例的定位效率。“Tarantula”表示利用 Tarantula 技术程序实例的定位效率，“Code-Coverage(α = 0.1)”表示利用 Code-Coverage 技术，且α=0.1 时程序实例的定位效率，“Code-Coverage(α=0.01)”表示利用 Code-Coverage 技术，且α=0.01 时程序实例的定位效率，“Code-Coverage(α=0.001)”表示利用 Code-Coverage 技术，且α=0.001 时程序实例的定位效率，“Mutants”表示故障版本数量，“Located Mutants”表示能够被正确定位出来的故障版本的数量，“Correct Rate”表示定位的效率。

表 5-2 SmartShelf 定位效率

Technique	Mutants	Located Mutants	Correct Rate
P(b)	103	93	90.29%
P(f)	103	93	90.29%
Tarantula	103	67	65.05%
Code-Coverage(α=0.1)	103	60	58.25%
Code-Coverage (α=0.01)	103	54	52.43%
Code-Coverage (α=0.001)	103	60	58.25%

表 5-3 TravelAgency 定位效率

Technique	Mutants	Located Mutants	Correct Rate
P(b)	92	83	90.22%
P(f)	92	83	90.22%
Tarantula	92	61	66.30%
Code-Coverage(α=0.1)	92	52	56.52%
Code-Coverage(α=0.01)	92	55	59.78%
Code-Coverage(α=0.001)	92	60	65.22%

表 5-4 QuoteProcess 定位效率

Technique	Mutants	Located Mutants	Correct Rate
P(b)	83	82	98.80%
P(f)	83	82	98.80%
Tarantula	83	52	62.65%
Code-Coverage(α=0.1)	83	49	59.04%
Code-Coverage(α=0.01)	83	47	56.63%
Code-Coverage(α=0.001)	83	45	54.22%

表 5-2 列出了多种技术应用于 SmartShelf 实例的定位效率，在 103 个故障版本中，基于谓词切换技术且选择 LEFS 策略时可以定位出 93 个故障版本，选择 LELS 策略时也可以定位出 93 个故障版本。Tarantula 技术可以定位出 67 个故障版本。Code-Coverage 技术选用α=0.1 时能够定位出 60 个故障版本。选用α=0.01 时，可以定位出 54 个故障版本；选用α=0.001 时，可以定位出 60 个故障版本。通过 SmartShelf 实例可以看出，基于谓词切换技术的故障定位效率要优于其它技术，且 LEFS 策略和 LELS 策略的定位效率相等。

表 5-3 列出了多种技术应用于 TravelAgency 实例的定位效率，在 92 个故障版本中，基于谓词切换技术且选择 LEFS 策略时可以定位出 83 个故障版本，选择 LELS 策略时也可以定位出 83 个故障版本。Tarantula 技术可以定位出 61 个故障版本。Code-Coverage 技术选用α=0.1 时能够定位出 52 个故障版本。选用α=0.01 时，可以定位出 55 个故障版本；选用α=0.001 时，可以定位出 60 个故障版本。通过 TravelAgency 实例可以看出，基于谓词切换技术的故障定

位效率要优于其它技术，且 LEFS 策略和 LELS 策略的定位效率相等。

表 5-4 列出了多种技术应用于 QuoteProcess 实例的定位效率，在 83 个故障版本中，基于谓词切换技术且选择 LEFS 策略时可以定位出 82 个故障版本，选择 LELS 策略时也可以定位出 83 个故障版本，定位效率为 82/83=98.8%。Tarantula 技术可以定位出 52 个故障版本。Code-Coverage 技术选用α=0.1 时能够定位出 49 个故障版本。选用α=0.01 时，可以定位出 47 个故障版本；选用α=0.001 时，可以定位出 45 个故障版本。通过 QuoteProcess 实例可以看出，基于谓词切换技术的故障定位效率要优于其它技术，且 LEFS 策略和 LELS 策略的定位效率相等。

从三个实例的试验结果来看，基于谓词切换的技术的故障定位效率保持在 90%以上，高于 Tarantula 技术和 Code-Coverage 技术的故障定位效率。

在图 5-3 到图 5-8 中列出了多种故障定位方法的对比图，其中横坐标表示故障定位的技术种类，纵坐标表示故障定位的效率。“P(b)”代表利用 LEFS 策略进行的谓词切换的故障定位效率，而“P(f)”代表利用 LELS 策略进行的谓词切换的故障定位效率。图 5-3 到图 5-5 展示了谓词切换技术利用 LEFS 策略的故障定位效率与 Tarantula 技术和 Code-Coverage 技术的对比，图 5-6 到图 5-8 展示了谓词切换技术利用 LELS 策略的故障定位效率与其他技术的定位效率的对比。

从图 5-3 到图 5-8 中可以看出：

(1) 基于谓词切换的故障定位技术的定位效率基本保持在 90%，而 Tarantula 技术保持在 60%左右，而 Code-Coverage 技术的故障定位效率则在 55%左右。因此基于谓词切换的故障定位技术的定位效率要高于 Tarantula 技术和 Code-Coverage 技术。

(2) Tarantula 技术的故障定位效率比 Code-Coverage 技术的故障定位效率稍好，在三个实例显示的数据上 Tarantula 技术的定位效率都高于 Code-Coverage 技术。

(3) 无论是利用 LEFS 策略还是 LELS 策略，基于谓词切换的故障定位技术的定位效率都要优于 Tarantula 和 Code-Coverage 方法。并且这两种策略的选择对定位的效率没有影响。

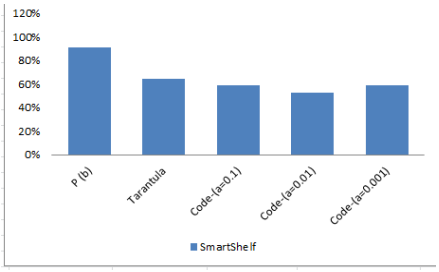

图 5-3 SmartShelf P(b) 故障定位效率对比图

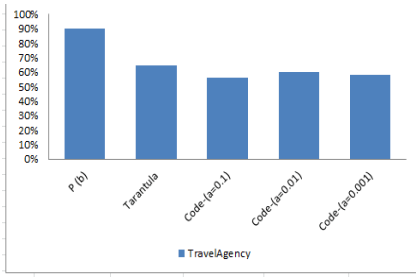

图 5-4 TravelAgecy P(b) 故障定位效率对比图

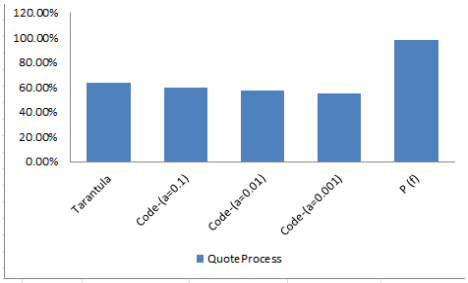

图 5-5 QutoProcess P(b) 故障定位效率对比图

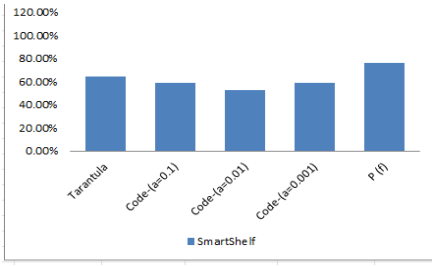

图 5-6 SmartShelf P(f) 故障定位效率对比图

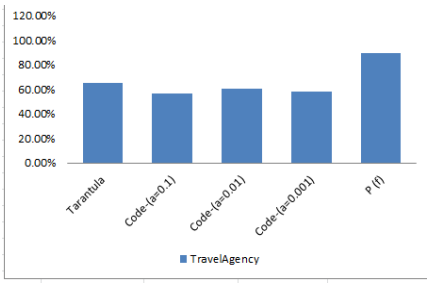

图 5-7 TravelAgency P(f) 故障定位效率对比图

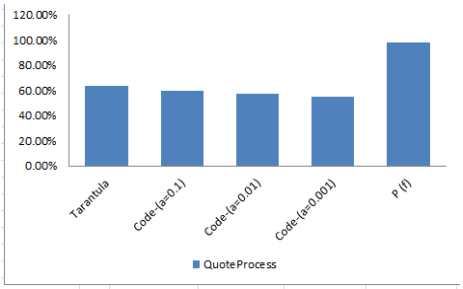

图 5-8 QutoProcess P(f) 故障定位效率对比图

5.3.2 故障定位精度

通过实验，SmartShelf、TravelAgency、QuoteProcess 的定位效率依次见表 5-5，表 5-6，表 5-7 所示。其中“Num”对应故障版本，“P(b)”表示基于谓词切换技术且选择 LEFS 策略进行故障定位的精度，“P(f)”表示基于谓词切换技术且选择 LELS 策略进行故障定位的精度，“Tarantula”对应 Tarantula 技术对应的定位进度，“LOP(α=0.1)”表示 Code-Coverage 技术选择α=0.1 时的故障定位精度，“LOP(α=0.01)”表示 Code-Coverage 技术选择α=0.01 时的

故障定位精度，“LOP(α=0.001)”表示 Code-Coverage 技术选择α=0.001 时的故障定位精度。

表 5-5 Smartshelf 定位精度实验结果

Num	P(b)	Tarantula	LOP(α=0.1)	LOP(α=0.01)	LOP(α=0.001)	P(f)
1	0.00%	0.00%	20.30%	0.00%	0.00%	0.00%
2	0.00%	16.30%	21.70%	35.20%	19.70%	0.00%
3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4	0.00%	16.30%	35.50%	35.50%	0.00%	0.00%
5	0.00%	16.30%	0.00%	0.00%	0.00%	0.00%
6	0.00%	24.00%	35.20%	35.20%	35.20%	0.00%
7	0.00%	23.50%	0.00%	0.00%	24.40%	0.00%
8	0.00%	16.30%	9.30%	0.00%	9.30%	0.00%
9	0.00%	16.30%	31.60%	0.00%	15.40%	0.00%
10	0.00%	16.30%	35.20%	0.00%	8.80%	0.00%
11	1.56%	16.70%	22.70%	27.90%	0.00%	1.56%
12	1.56%	0.00%	9.30%	0.00%	9.30%	1.56%
13	1.56%	0.00%	0.00%	0.00%	0.00%	1.56%
14	1.60%	0.00%	24.40%	0.00%	24.40%	1.60%
15	1.60%	18.60%	0.00%	20.30%	0.00%	1.60%
16	1.60%	18.60%	22.70%	31.60%	0.00%	1.60%
17	1.62%	16.30%	0.00%	24.40%	35.20%	1.62%
18	1.62%	16.30%	9.30%	0.00%	31.60%	1.62%
19	1.62%	16.30%	0.00%	31.60%	0.00%	1.62%
20	1.62%	16.30%	0.00%	0.00%	24.40%	1.62%
21	1.62%	16.30%	8.80%	20.30%	24.40%	1.62%
22	1.70%	0.00%	31.60%	21.70%	31.60%	1.70%
23	1.70%	16.30%	0.00%	0.00%	0.00%	1.70%
24	1.80%	0.00%	8.80%	8.80%	8.80%	1.80%
25	1.80%	0.00%	0.00%	24.40%	23.50%	1.80%
26	1.80%	24.00%	27.90%	0.00%	0.00%	1.80%
27	1.80%	24.00%	0.00%	0.00%	0.00%	1.80%
28	1.80%	0.00%	0.00%	16.30%	25.40%	1.80%
29	1.80%	0.00%	20.30%	20.30%	20.30%	1.80%
30	1.80%	16.30%	31.60%	9.30%	31.60%	1.80%
31	1.80%	19.40%	24.40%	20.30%	24.40%	1.80%
32	1.80%	16.30%	20.30%	24.40%	0.00%	1.80%
33	1.80%	19.40%	31.60%	20.30%	31.60%	1.80%
34	1.80%	19.40%	0.00%	24.40%	24.40%	1.80%
35	1.80%	0.00%	21.70%	0.00%	0.00%	1.80%
36	1.80%	24.40%	0.00%	0.00%	18.60%	1.80%
37	1.80%	24.40%	0.00%	20.30%	0.00%	1.80%
38	1.80%	19.40%	31.60%	31.60%	31.60%	1.80%
39	1.80%	19.40%	0.00%	0.00%	24.40%	1.80%
40	1.87%	0.00%	0.00%	0.00%	0.00%	1.87%
41	1.87%	0.00%	0.00%	0.00%	0.00%	1.87%

表 5-5 Smartshelf 定位精度实验结果(续)

42	1.87%	16.30%	24.40%	24.40%	35.20%	1.87%
43	1.87%	16.30%	20.30%	20.30%	20.30%	1.87%
44	1.87%	0.00%	24.40%	0.00%	24.40%	1.87%
45	1.90%	0.00%	8.80%	8.80%	8.80%	1.90%
46	1.90%	16.30%	24.40%	24.40%	24.40%	1.90%
47	1.90%	16.30%	0.00%	15.90%	10.90%	1.90%
48	1.90%	0.00%	0.00%	0.00%	0.00%	1.90%
49	1.90%	16.30%	16.30%	21.60%	17.30%	1.90%
50	2.10%	24.00%	20.30%	0.00%	20.30%	2.10%
51	2.10%	0.00%	9.30%	24.40%	0.00%	2.10%
52	2.10%	24.00%	20.30%	0.00%	0.00%	2.10%
53	2.30%	24.00%	24.40%	0.00%	6.25%	2.30%
54	2.30%	16.30%	20.30%	35.20%	0.00%	2.30%
55	2.30%	16.30%	24.40%	0.00%	24.40%	2.30%
56	2.30%	0.00%	0.00%	31.60%	0.00%	2.30%
57	2.30%	16.30%	0.00%	0.00%	21.70%	2.30%
58	2.30%	16.30%	20.30%	27.20%	20.30%	2.30%
59	2.30%	16.30%	24.40%	28.30%	24.40%	2.30%
60	2.30%	0.00%	0.00%	0.00%	0.00%	2.30%
61	2.30%	0.00%	0.00%	27.20%	0.00%	2.30%
62	2.30%	0.00%	0.00%	0.00%	26.40%	2.30%
63	2.30%	24.00%	24.00%	24.00%	24.00%	2.30%
64	2.30%	16.30%	35.50%	35.50%	27.60%	2.30%
65	2.30%	24.00%	24.40%	24.40%	24.40%	2.30%
66	2.40%	18.60%	21.60%	21.60%	0.00%	2.40%
67	2.40%	18.60%	21.60%	21.60%	0.00%	2.40%
68	2.40%	0.00%	0.00%	0.00%	0.00%	2.40%
69	2.57%	16.30%	24.40%	0.00%	24.40%	2.57%
70	2.57%	16.30%	0.00%	0.00%	20.30%	2.57%
71	2.57%	0.00%	0.00%	0.00%	0.00%	2.57%
72	2.60%	23.10%	35.20%	35.20%	35.20%	2.60%
73	2.60%	0.00%	0.00%	0.00%	0.00%	2.60%
74	2.60%	23.10%	31.60%	31.60%	31.60%	2.60%
75	2.60%	23.50%	0.00%	0.00%	0.00%	2.60%
76	2.60%	23.50%	0.00%	31.60%	31.60%	2.60%
77	2.60%	0.00%	0.00%	0.00%	0.00%	2.60%
78	2.60%	24.00%	0.00%	0.00%	0.00%	2.60%
79	2.60%	0.00%	24.40%	24.00%	0.00%	2.60%
80	2.60%	0.00%	24.40%	35.50%	0.00%	2.60%
81	2.60%	24.00%	0.00%	35.50%	0.00%	2.60%
82	2.70%	15.40%	0.00%	24.40%	23.30%	2.70%
83	2.70%	15.40%	17.50%	21.60%	17.50%	2.70%
84	2.70%	0.00%	0.00%	21.60%	21.70%	2.70%
85	2.70%	15.40%	35.20%	0.00%	0.00%	2.70%
86	2.70%	0.00%	31.60%	0.00%	0.00%	2.70%
87	3.20%	16.30%	35.20%	0.00%	9.30%	3.20%
88	3.20%	16.30%	0.00%	31.60%	0.00%	3.20%

表 5-5 Smartshelf 定位精度实验结果(续)

89	3.20%	0.00%	35.20%	0.00%	8.80%	3.20%
90	3.20%	25.30%	35.10%	35.20%	0.00%	3.20%
91	3.57%	24.00%	9.30%	0.00%	9.30%	3.57%
92	3.57%	24.00%	0.00%	0.00%	0.00%	3.57%
93	3.57%	0.00%	8.80%	0.00%	0.00%	3.57%
94	3.57%	0.00%	0.00%	0.00%	0.00%	3.57%
95	4.50%	24.40%	20.30%	20.30%	20.30%	4.50%
96	4.50%	0.00%	24.40%	24.40%	24.40%	4.50%
97	4.50%	24.40%	0.00%	0.00%	18.60%	4.50%
98	4.70%	0.00%	0.00%	0.00%	0.00%	4.70%
99	4.70%	24.40%	20.30%	20.30%	20.30%	4.70%
100	4.70%	0.00%	24.40%	24.40%	24.40%	4.70%
101	4.70%	24.40%	0.00%	0.00%	18.60%	4.70%
102	4.90%	0.00%	24.40%	24.40%	24.40%	4.90%
103	4.90%	24.40%	0.00%	0.00%	18.60%	4.90%

表 5-5 列出了多种技术应用于 SmartShelf 实例的故障定位精度。基于谓词切换的故障定位技术选择 LEFS 策略时的定位精度，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 1.56%到 4.90%之间，利用 LEFS 策略和 LELS 策略的定位精度一致；Tarantula 技术的定位精度约 20%，最小值为 16.3%，最大值为 24.4%；Code-Coverage 技术用用于该实例，当α=0.1 时，定位的精度约 25%，最小值为 9.3%，最大值为 35.5%。当α=0.01 时，定位的精度约为 25%，最小值为 8.4%，最大值为 35.2%。当α=0.001 时，定位精度约为 25%，最小值为 8.8%，最大值为 35.2%。因此，在 SmartShelf 实例中，基于谓词切换的故障定位效率为的定位精度要优于 Tarantula 技术和 Code-Coverage 技术。

表 5-6 TravelAgency 定位精度实验结果

Num	P(b)	Tarantula	LOP(α=0.1)	LOP(α=0.01)	LOP(α=0.001)	P(f)
1	3.49%	25.10%	30.30%	25.10%	25.10%	3.49%
2	3.49%	24.30%	0.00%	22.70%	21.30%	3.49%
3	3.49%	24.30%	27.30%	33.30%	33.30%	3.49%
4	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
5	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
6	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
7	3.49%	33.10%	26.30%	24.30%	22.10%	3.49%
8	3.49%	30.30%	32.30%	24.30%	22.10%	3.49%
9	3.49%	33.30%	32.30%	24.30%	22.10%	3.49%
10	3.49%	33.30%	32.30%	33.30%	33.30%	3.49%
11	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
12	3.49%	33.30%	25.70%	33.30%	33.30%	3.49%

表 5-6 TravelAgency 定位精度实验结果（续）

13	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
14	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
15	3.49%	0.00%	0.00%	0.00%	0.00%	3.49%
16	3.49%	20.10%	25.70%	30.30%	30.30%	3.49%
17	3.56%	22.40%	33.30%	27.30%	23.20%	3.56%
18	3.56%	26.20%	25.70%	29.40%	28.70%	3.56%
19	3.56%	31.20%	33.30%	32.30%	32.30%	3.56%
20	3.56%	28.70%	0.00%	30.20%	28.10%	3.56%
21	3.56%	22.40%	0.00%	27.30%	23.20%	3.56%
22	3.56%	25.10%	0.00%	24.30%	24.30%	3.56%
23	3.56%	22.40%	20.30%	32.10%	32.10%	3.56%
24	3.56%	33.30%	30.30%	33.30%	33.30%	3.56%
25	0.00%	33.30%	31.30%	0.00%	0.00%	0.00%
26	3.56%	33.30%	19.60%	33.30%	33.30%	3.56%
27	3.56%	0.00%	0.00%	0.00%	0.00%	3.56%
28	3.56%	33.30%	19.60%	33.30%	33.30%	3.56%
29	3.56%	32.10%	20.10%	31.20%	30.10%	3.56%
30	0.00%	0.00%	20.10%	0.00%	0.00%	0.00%
31	3.56%	0.00%	20.10%	0.00%	0.00%	3.56%
32	3.79%	0.00%	0.00%	0.00%	0.00%	3.79%
33	3.79%	32.10%	30.20%	30.30%	31.20%	3.79%
34	3.79%	33.30%	0.00%	33.30%	33.30%	3.79%
35	3.79%	25.40%	30.20%	0.00%	0.00%	3.79%
36	3.79%	27.20%	30.20%	27.10%	22.50%	3.79%
37	3.79%	0.00%	0.00%	0.00%	0.00%	3.79%
38	3.79%	30.20%	0.00%	24.50%	24.50%	3.79%
39	3.79%	0.00%	0.00%	0.00%	33.30%	3.79%
40	3.79%	33.30%	25.40%	33.30%	33.30%	3.79%
41	3.79%	0.00%	25.40%	0.00%	0.00%	3.79%
42	0.00%	0.00%	0.00%	33.30%	31.20%	0.00%
43	0.00%	0.00%	25.40%	0.00%	0.00%	0.00%
44	3.79%	30.30%	33.30%	24.50%	21.70%	3.79%
45	3.79%	30.30%	27.30%	24.50%	21.70%	3.79%
46	3.79%	24.30%	27.30%	0.00%	0.00%	3.79%
47	3.79%	24.30%	0.00%	24.30%	24.30%	3.79%
48	3.79%	0.00%	0.00%	0.00%	0.00%	3.79%
49	3.79%	33.30%	33.30%	33.30%	33.30%	3.79%
50	3.79%	0.00%	0.00%	0.00%	0.00%	3.79%
51	3.79%	33.30%	31.60%	33.30%	33.30%	3.79%
52	3.79%	33.30%	31.60%	0.00%	0.00%	3.79%
53	3.79%	33.30%	31.60%	33.30%	33.30%	3.79%
54	3.79%	0.00%	31.60%	0.00%	0.00%	3.79%
55	4.02%	33.30%	0.00%	25.70%	25.70%	4.02%
56	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
57	0.00%	33.30%	0.00%	33.30%	33.30%	0.00%
58	4.02%	0.00%	0.00%	0.00%	25.70%	4.02%
59	4.02%	33.30%	33.30%	33.30%	0.00%	4.02%

表 5-6 TravelAgency 定位精度实验结果（续）

60	4.02%	33.30%	21.60%	0.00%	30.20%	4.02%
61	4.02%	33.30%	19.70%	33.30%	30.20%	4.02%
62	4.02%	0.00%	0.00%	0.00%	33.30%	4.02%
63	4.02%	25.10%	21.60%	25.70%	24.60%	4.02%
64	4.02%	24.30%	33.30%	0.00%	0.00%	4.02%
65	4.02%	24.30%	30.30%	0.00%	24.30%	4.02%
66	4.02%	0.00%	0.00%	33.30%	33.30%	4.02%
67	4.02%	0.00%	0.00%	0.00%	33.30%	4.02%
68	4.02%	0.00%	0.00%	33.30%	33.30%	4.02%
69	4.02%	0.00%	0.00%	0.00%	0.00%	4.02%
70	4.02%	33.10%	33.30%	29.30%	28.70%	4.02%
71	4.02%	30.30%	0.00%	0.00%	0.00%	4.02%
72	4.02%	33.30%	23.20%	25.60%	24.30%	4.02%
73	4.02%	33.30%	0.00%	0.00%	0.00%	4.02%
74	4.02%	0.00%	33.30%	33.30%	30.20%	4.02%
75	4.02%	33.30%	33.30%	33.20%	30.50%	4.02%
76	4.02%	0.00%	31.60%	26.30%	27.90%	4.02%
77	4.02%	0.00%	31.60%	0.00%	0.00%	4.02%
78	4.02%	0.00%	31.60%	35.30%	33.30%	4.02%
79	4.76%	20.10%	0.00%	33.30%	33.30%	4.76%
80	4.76%	22.40%	0.00%	0.00%	0.00%	4.76%
81	4.76%	26.20%	0.00%	33.30%	33.30%	4.76%
82	0.00%	31.20%	0.00%	33.30%	33.30%	0.00%
83	0.00%	28.70%	33.30%	28.60%	27.90%	0.00%
84	4.76%	22.40%	21.60%	33.30%	33.30%	4.76%
85	4.76%	25.10%	0.00%	0.00%	19.70%	4.76%
86	4.76%	22.40%	27.30%	33.30%	33.30%	4.76%
87	0.00%	33.30%	21.60%	33.30%	33.30%	0.00%
88	4.76%	33.30%	33.30%	0.00%	0.00%	4.76%
89	4.76%	33.30%	0.00%	33.30%	0.00%	4.76%
90	4.76%	0.00%	0.00%	33.30%	27.90%	4.76%
91	4.76%	25.10%	0.00%	33.30%	33.30%	4.76%
92	4.76%	24.30%	33.30%	0.00%	0.00%	4.76%

表 5-6 列出了多种技术应用于 TravelAgency 实例的故障定位精度。基于谓词切换的故障定位技术选择 LEFS 策略时定位精度介于 3.49%到 4.76%之间，利用 LEFS 策略和 LELS 策略的定位精度一致；将 Tarantula 技术应用于 TravelAgency 程序时其定位的精度为 30%左右浮动，最小值为 20.1%，最大值为 33.3%；Code-Coverage 技术应用于该实例，当α=0.1 时，定位的精度约为 25%，最小值为 19.7%，最大值为 33.3%。当α=0.01 时，定位的精度约为 25%，最小值为 22.7%，最大值为 33.3%。当α=0.001 时，定位的精度约为 25%，最小值为 21.3%，最大值为 33.3%。因此，在 TravelAgency 实例中，基于谓词切换的故障定位效率为的定位精度要优于其它技术。

表 5-7 QuoteProcess 定位精度实验结果

Num	P(b)	Tarantula	LOP(α=0.1)	LOP(α=0.01)	LOP(α=0.001)	P(f)
1	0.00%	18.40%	18.40%	0.00%	18.40%	0.00%
2	1%	45.70%	45.70%	46%	45.70%	1%
3	1%	0.00%	0.00%	46%	15.20%	1%
4	1%	45.70%	45.70%	0.00%	45.70%	1%
5	1%	15.20%	45.70%	46.90%	45.70%	1%
6	1%	0.00%	0.00%	46.90%	0.00%	1%
7	1%	9.50%	9.50%	0.00%	46.40%	1%
8	1.70%	32.60%	32.60%	0.00%	0.00%	1.70%
9	1.70%	8.60%	0.00%	0.00%	8.60%	1.70%
10	1.70%	0.00%	25.50%	25.50%	0.00%	1.70%
11	1.70%	32.60%	32.60%	32.60%	14.40%	1.70%
12	1.70%	45.70%	45.70%	0.00%	45.70%	1.70%
13	1.70%	0.00%	0.00%	0.00%	0.00%	1.70%
14	1.70%	15.40%	45.70%	45.70%	45.70%	1.70%
15	1.70%	0.00%	0.00%	0.00%	0.00%	1.70%
16	1.70%	15.40%	45.70%	0.00%	0.00%	1.70%
17	1.70%	9.50%	9.50%	9.50%	9.50%	1.70%
18	1.70%	15.40%	15.40%	0.00%	15.40%	1.70%
19	1.70%	9.50%	9.50%	9.50%	9.50%	1.70%
20	1.70%	46.40%	46.40%	0.00%	46.40%	1.70%
21	1.70%	15.40%	45.70%	0.00%	0.00%	1.70%
22	1.70%	25.70%	45.70%	0.00%	17.50%	1.70%
23	1.70%	0.00%	0.00%	45.40%	0.00%	1.70%
24	1.70%	45.70%	45.70%	0.00%	45.70%	1.70%
25	1.70%	0.00%	0.00%	0.00%	0.00%	1.70%
26	1.70%	0.00%	0.00%	0.00%	0.00%	1.70%
27	1.70%	9.50%	9.50%	9.50%	9.50%	1.70%
28	1.70%	23.40%	16.30%	0.00%	16.30%	1.70%
29	1.70%	10.50%	10.50%	10.50%	0.00%	1.70%
30	1.80%	17.20%	17.20%	46.90%	0.00%	1.80%
31	1.80%	0.00%	0.00%	0.00%	0.00%	1.80%
32	1.80%	45.70%	0.00%	0.00%	45.70%	1.80%
33	1.80%	45.70%	45.70%	0.00%	19.40%	1.80%
34	1.80%	18.40%	18.40%	0.00%	18.40%	1.80%
35	2.10%	17.20%	17.20%	0.00%	17.20%	2.10%
36	2.10%	0.00%	0.00%	25.70%	17.20%	2.10%
37	2.10%	0.00%	0.00%	25.70%	0.00%	2.10%
38	2.10%	45.70%	45.70%	0.00%	0.00%	2.10%
39	2.30%	9.50%	9.50%	27.60%	9.50%	2.30%
40	2.30%	0.00%	0.00%	27.60%	0.00%	2.30%
41	2.30%	0.00%	0.00%	0.00%	0.00%	2.30%
42	2.50%	18.40%	28.70%	27.60%	0.00%	2.50%
43	2.50%	45.70%	45.70%	15.70%	19.40%	2.50%
44	2.50%	16.90%	16.90%	30.60%	0.00%	2.50%
45	2.50%	29.40%	29.40%	0.00%	29.40%	2.50%

表 5-7 QuoteProcess 定位精度实验结果（续）

46	2.50%	46.40%	29.40%	17.90%	46.40%	2.50%
47	2.70%	45.70%	45.70%	17.90%	45.70%	2.70%
48	2.70%	45.70%	45.70%	18.60%	45.70%	2.70%
49	2.70%	0.00%	0.00%	18.60%	0.00%	2.70%
50	2.70%	45.70%	45.70%	25.70%	45.70%	2.70%
51	2.70%	0.00%	0.00%	25.70%	0.00%	2.70%
52	2.70%	0.00%	0.00%	18.60%	0.00%	2.70%
53	2.70%	45.70%	45.70%	25.70%	45.70%	2.70%
54	2.70%	0.00%	0.00%	25.70%	0.00%	2.70%
55	2.80%	0.00%	0.00%	17.90%	45.70%	2.80%
56	2.80%	46.40%	46.40%	15.70%	0.00%	2.80%
57	2.80%	45.70%	0.00%	17.90%	45.70%	2.80%
58	2.80%	0.00%	0.00%	0.00%	0.00%	2.80%
59	2.80%	45.70%	45.70%	22.50%	45.70%	2.80%
60	2.80%	46.40%	46.40%	15.70%	0.00%	2.80%
61	2.80%	45.70%	0.00%	17.90%	45.70%	2.80%
62	2.80%	0.00%	0.00%	0.00%	0.00%	2.80%
63	2.80%	45.70%	45.70%	22.50%	45.70%	2.80%
64	3.10%	0.00%	0.00%	15.70%	0.00%	3.10%
65	3.10%	0.00%	0.00%	15.70%	0.00%	3.10%
66	3.20%	9.50%	9.50%	30.50%	9.50%	3.20%
67	3.20%	0.00%	0.00%	0.00%	0.00%	3.20%
68	3.20%	9.50%	9.50%	0.00%	9.50%	3.20%
69	3.20%	45.70%	45.70%	30.60%	19.40%	3.20%
70	3.20%	0.00%	0.00%	22.50%	0.00%	3.20%
71	3.20%	0.00%	0.00%	22.50%	0.00%	3.20%
72	3.20%	9.50%	9.50%	30.50%	9.50%	3.20%
73	3.20%	0.00%	0.00%	0.00%	0.00%	3.20%
74	3.20%	9.50%	9.50%	0.00%	9.50%	3.20%
75	3.50%	0.00%	0.00%	17.90%	0.00%	3.50%
76	3.70%	0.00%	0.00%	0.00%	0.00%	3.70%
77	4.10%	46.40%	46.40%	0.00%	46.40%	4.10%
78	4.20%	9.50%	0.00%	18.60%	9.50%	4.20%
79	4.20%	0.00%	0.00%	0.00%	0.00%	4.20%
80	4.20%	0.00%	46.40%	0.00%	0.00%	4.20%
81	4.20%	46.90%	46.90%	46.90%	46.90%	4.20%
82	4.30%	0.00%	0.00%	17.90%	0.00%	4.30%
83	4.70%	45.70%	45.70%	0.00%	19.40%	4.70%

表 5-7 列出了多种技术应用于 QuoteProcess 实例的故障定位精度。基于谓词切换的故障定位技术选择 LEFS 策略时的定位精度，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介于 1%到 4.70%之间，利用 LEFS 策略和 LELS 策略的定位精度一致；Tarantula 技术的定位精度约 35%，最小值为 9.5%，最大值为 46.4%；Code-Coverage 技术应用于该实例，当α=0.1 时，定位的精度约为 30%，最小值为 9.5%，最大值为 45.7%。

当α=0.01 时，定位的精度约为 30%，最小值为 9.5%，最大值为 46.9%。当α=0.001 时，定位精度约为 30%，最小值为 9.5%，最大值为 46.9%。因此，在 QuoteProcess 实例中，基于谓词切换的故障定位效率为的定位精度要优于 Tarantula 技术和 Code-Coverage 技术。

图 5-9 到图 5-14 列出了所有技术的故障定位精度，其中“P(b)”指基于谓词切换的技术利用 LEFS 策略所得到的故障定位精度，“P(f)”指基于谓词切换的技术利用 LELS 策略所得到的故障定位精度。

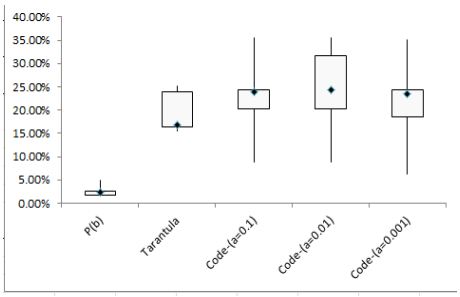

图 5-9 SmartShelf P(b)定位精度对比图

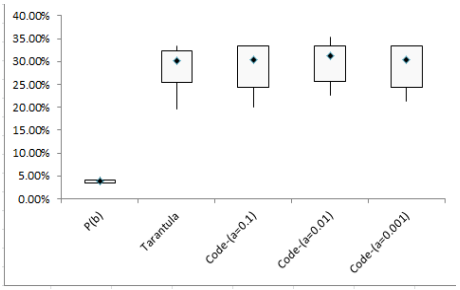

图 5-10 TravelAgency P(b)定位精度对比图

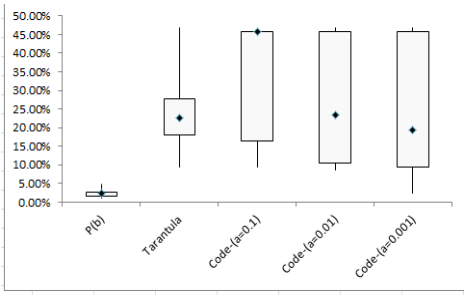

图 5-11 QuoteProcess P(b)定位精度对比图

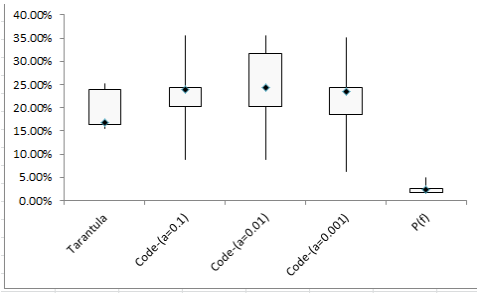

图 5-12 SmartShelf P(f)定位精度对比图

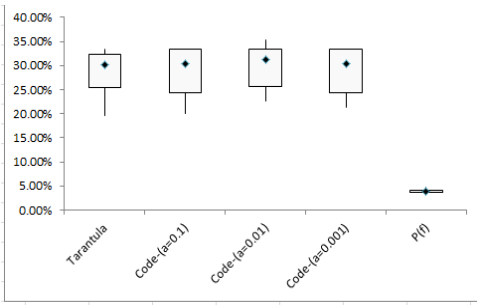

图 5-13 TravelAgency P(f)定位精度对比图

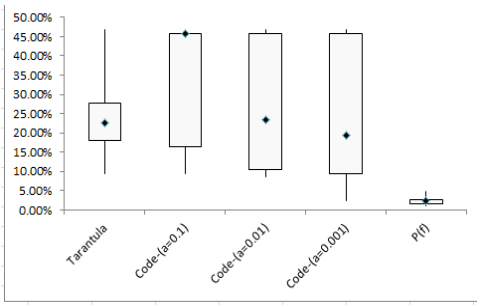

图 5-14 QuoteProcess P(f)定位精度对比图

从图 5-3 到图 5-8 中可以看出：

1) 基于谓词切换的故障定位技术的定位精度基本保持在 10%以内，而 Tarantula 技术保持在 20%左右，而 Code-Coverage 技术的故障定位效率

则在 30%左右。因此基于谓词切换的故障定位技术的定位精度要高于 Tarantula 技术和 Code-Coverage 技术。

2) Tarantula 技术的故障定位精度与 Code-Coverage 技术的故障定位精度稍好，在三个实例显示的数据上 Tarantula 技术的定位精度都高于 Code-Coverage 技术。

3) 无论是利用 LEFS 策略还是 LELS 策略，基于谓词切换的故障定位技术的定位精度都要优于 Tarantula 和 Code-Coverage 方法。并且这两种策略的选择对定位的精度没有影响。

图 5-15 到 5-17 列出了所有定位效率区间的版本分布，其中横坐标代表定位的效率，纵坐标指故障版本的个数。

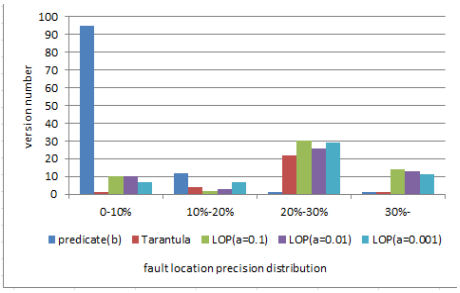

图 5-15 SmartShelf P(b)定位精度分布图

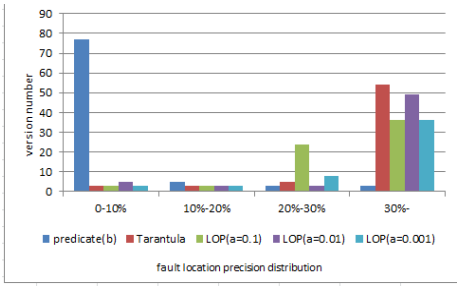

图 5-16 TravelAgency P(b)定位精度分布图

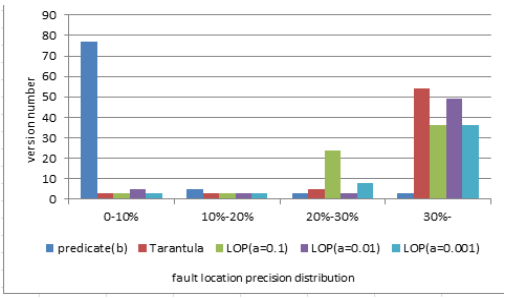

图 5-17 QuoteProcess P(b)定位精度分布图

从图 5-15 到 5-17 可以看出基于谓词切换技术的故障版本的定位效率主要集中在 0-10%区间内，而其他三种方法则分别分布在 10%-20%，20%-30%和 30%以上的区间中，与谓词切换技术的 10%相比，其定位的精度相对低。

基于谓词切换的技术中关键谓词排序策略的不同在平均精度和方差上不存在差异，表 5-8 展示了基于谓词切换技术利用两种排序策略的定位精度的平均值和方差。

表 5-8　各实例定位精度的方差与均值

instances / variables	SmartShelf		TravelAgency		QuoteProcess	
	P(b)	P(f)	P(b)	P(f)	P(b)	P(f)
average locating precision	2.46%	2.46%	3.90%	3.90%	2.42%	2.42%
standard deviation	1.03%	1.03%	0.4%	0.4%	0.86%	0.86%

两种策略的平均定位精度相等。因为在找到关键谓词的前提下，后续的定位步骤是一致的，因此在找到关键谓词的所有故障版本中，两种策略的定位精度是相等的。另外，LEFS 策略和 LELS 策略下的定位精度都趋于稳定，因为他们的方差相等。

5.3.3 影响因子分析

实验结果表明：不同的排序策略对故障定位的效率不存在影响，就平均比较次数而言，谓词切换的次数与故障存在的位置密切相关，例如：如果某个故障存在的位置靠近“ReceiveInput”块，存在的关键谓词的位置也将靠近“ReceiveInput”块，也就是说若谓词的集合为 $p=\{p_1, p_2, p_3, \ldots, p_n\}$，并且故障存在于 p_n 处，当利用 LEFS 排序策略进行谓词切换时仅需要 1 次即可，否则的话即为 n-1 次，因此对于所有的故障版本而言，其平均比较次数为 *n(n-1)/2* 次。表 5-9 展示了两种不同的排序策略的平均比较次数的对比情况。

表 5-9　各实例寻找关键谓词的平均切换次数

instance / variables	SmartShelf		TravelAgency		QuoteProcess	
	P(b)	P(f)	P(b)	P(f)	P(b)	P(f)
average switching times	2.10	2.12	2.30	2.67	1.93	2.06

表 5-9 表明：不同的排序策略的平均比较次数是不同的。这种差异是由版本的分布状态引起的。例如，假设在某个程序中有五个谓词，根据与输出位置的远近分别是 $d_i(i=1, 2, 3, 4, 5)$，如表 5-10 所示，在 d_1 处有 6 个故障版本，在 d_2 处有 3 个故障版本，…，在 d_5 处有 4 个故障版本。利用 LEFS 排序策略得到的平均比较次数为 3.2，而利用 LELS 得到的平均比较次数为 2.8。因此可以得出：平均比较次数与故障版本的位置有关系，而与切换的策略没有关系，也就是说不管是 LEFS 还是 LELS 对故障定位的效率和精度没有影响，

对平均切换次数没有影响，而故障版本的分布会影响平均切换次数。

表 5-10　不同切换策略平均比较次数

按距离排序(d_i)	版本数量(n_i)	LEFS 切换次数	LELS 切换次数
d1	6	5	1
d2	3	4	2
d3	4	3	3
d4	3	2	4
d5	4	1	5

5.4 小结

本节讨论了实验方案的设计及实验结果的分析。针对三个实例进行了多种技术的故障定位的验证。实验方案的设计包括测试用例的选择，故障版本的生成以及执行和记录。在实验结果的分析部分，从故障定位效率，定位的精度以及定位影响因子分析了基于谓词的故障定位技术的优势和不足。

6 结论

针对 BPEL 程序的特点及其特有的故障类型，本文将基于谓词切换的故障定位技术应用于 BPEL 程序的故障定位，提出了一个基于谓词切换的 BPEL 程序的故障定位框架，将能够检测出故障的测试用例集、BPEL 程序语句集合及预期输出作为输入，输出最有可能的故障语句块。根据该框架开发出了相应的故障定位支持工具。为了验证和评估基于谓词切换的故障定位技术的有效性，将基于谓词切换的故障技术的定位效率和精度与 Tarantula 和 Code-Coverage 故障定位技术的定位效率和精度进行了对比，并分析了不同策略选择对定位效率的影响。

本文取得的主要研究成果包括：

(1) **提出了一种基于谓词切换的 BPEL 程序故障定位方法**

该方法基于 BPEL 程序的语句块结构和能够检测出故障的测试用例，通过多次切换过程找到关键谓词，并分析与关键谓词相关的程序切片来寻找程序故障。

(2) **开发了基于谓词切换的 BPEL 程序的故障定位支持工具**

设计并实现了基于谓词切换的 BPEL 程序的故障定位支持工具 BPEL_PSLocator，实现了流程的自动化。

(3) **采用经验研究的方式评估了提出的故障定位的有效性**

采用了三个程序实例验证了提出的故障定位技术及支持工具的可行性；将基于谓词切换的故障定位技术的定位效率与精度与其他的定位技术的定位精度和有效性进行了比较。实验结果表明提出的基于谓词切换的故障定位技术更为高效。

需要进一步研究的工作：

(1) 本文应用了三个实例来验证了基于谓词切换的故障定位技术的有效性，在后期的研究中希望采用更通用更多的 BPEL 实例来验证该技术的有效性。

(2) 改进故障定位的支持工具，提供更好的用户体验。

(3) 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 所调用的外部服务，深入探索 BPEL 服务的故障定位问题。

7 参考文献

[1] 干忠兵，李长云．软件故障诊断技术综述[J]. 微计算机信．2010.30(34):161-163.

[2] 虞凯，林梦香．自动化软件故障定位技术研究进展[J]. 计算机学报．2011,20(08): 1411-1423.

[3] 鲍亮，宋胜利．BPEL 静态流程切片技术研究[J]. 系统工程与电子技术．2009, 31(1):241-249.

[4] 黄亮，姚放吾．Apache ODE 环境下 Web 服务组合技术的研究[J]. 计算机技术与发展. 2011,21(7):98-104.

[5] W3C. Simple Object Access Protocol Version 1.1 [EB/OL]. http://www.w3.org/TR/

[6] J. A. Jones．Fault Localization Using Visualization of Test Information[C]. //Proceedings of 26th International Conference on Software-Engineering (ICSE 2004), 2004:54-56.

[7] W. E. Wong, Y. Qi, L. Zhao and K. Y. Cai. Effective Fault Localization using Code Coverage[C]. //Proceedings of 31st Annual International Computer Software and Applications Conference, 2007:449-456.

[8] L. Zhao, Z. Zhang, L. Wang and X. Yin. Fault localization via noise reduction on coverage vector[C]. //Proceedings of the 23th International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), 2011:203-206.

[9] J. A. Jones and M. J. Harrold. Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique[C]. //Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), 2005:273–282.

[10] 鞠小林，姜淑娟，张艳梅，董国伟．软件故障定位技术进展[J]. 计算机科学与探索. 2012(06):481-493.

[11] J. D. Some. psychological evidence on how people debug computer programs[J]. International Journal of Man-Machine Studies, 1975, 7(2): 151-182.

[12] E. W. Dijkstra. Correctness Concerns and, among Other Things, Why They Are Resented [J]. Acm Sigplan Notices, 1978, 10(6):80-88.

[13] M. Weiser. Program Slicing [J]. IEEE Transactions on Software Engineering, 1984, 10(4):352-357.

[14] K. J. Ottenstein, L. M. Ottenstein. The program dependence graph in a

software development environment [C]. //Proceedings of ACM Sigplan Notices. ACM, 1984, 19(5): 177-184.

[15] 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.

[16] J. R. Lyle, D. R. Wallace, J. R. Graham, K. B. Gallagher, J. P. Poole, D. W. Binkley, Unravel project, URL: http://hissa.ncsl.nist.gov.

[17] H. Agrawal, R. DeMillo, E. Spafford. Debugging with dynamic slicing and backtracking[J]. Software Practice and Experience. 1993, 23(6): 589-616.

[18] H. Agrawal, J. Horgan, S. London, and W. Wong. Fault localization using execution slices and dataflow tests[C]. //Proceedings of Sixth International Symposium on Software Reliability Engineering. 1995,143-151.

[19] X. Y. Zhang and R. Gupta. Cost effective dynamic program slicing[C]. //Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Washington D. C. June 2004: 96-106.

[20] N. Gupta and H. He. Locating Faulty Code Using Failure-Inducing Chops[C]. //Proceedings of ACM International Conference on Automated Software Engineering. 2005:263—272.

[21] A. Zeller. Yesterday, my program worked. Today, it does not. Why?[C]. //Proceedings of Seventh European Software Engineering Conference/ Seventh ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE), 1999: 253-267.

[22] A. Zeller and R. Hildebrandt. Simplifying and isolating failure-inducing input [J]. IEEE Transactions on Software Engineering, 2002, 28(2):183-200.

[23] X. Y. Zhang, R. Gupta, Y. Zhang. Precise Dynamic Slicing Algorithms[C]. //Proceedings of 25th International Conference on IEEE Computer Society, 2003: 319-329.

[24] B. Korel and J. Laski. Dynamic Program Slicing [J]. Acm Sigplan Notices, 1990, 25(6):246-256.

[25] X. Y. Zhang. Locating Faults Through Automated Predicate Switching[C]. //Proceedings of the 28th International Conference on Software Engineering. ACM, 2006: 272-281.

[26] X. Y. Zhang. Experimental Evaluation of Using Dynamic Slices for Fault localization[C]. //Proceedings of the Sixth International Symposium on Automated Analysis-driven Debugging. ACM, 2005: 33-42.

[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 2002 International Conference on Dependable Systems and Networks. 2002:595-604.

[29] 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.

[30] R. Abreu, P. Zoeteweij, J. C. Gemund. On the accuracy of spectrum based fault localization[C]. //Proceedings of Testing: Academic and Industrial Conference, Practice and Research Techniques. 2007:89-98.

[31] C. Liu, X. F. Yan, L. Fei. SOBER：statistical model-based bug localization [J]. ACM SIGSOFT Software Engineering Notes. 2005, 30(5): 286-295.

[32] 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.

[33] 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.

[34] 张东源. 软件测试中的程序切片技术, 现代导航. 2012,03(6):231-238.

[35] Z. Zhang, W. K. Chan. Non-parametric statistical fault localization. Journal of Systems and Software[J]. Journal of Systems and Software. 2011, 84(6): 885-905.

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

[37] 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.

[38] J. A. Jones and M. J. Harrold. Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique[C]. //Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), 2005:273-282.

[39] W. E. Wong, Y. Qi, L. Zhao and K. Y. Cai. Effective Fault Localization using Code Coverage[C]. //Proceedings of 31st Annual International Computer Software and Applications Conference (COMPSAC 2007). 2007:449-456.

[40] L. Zhao, Z. Zhang, L. Wang, and X. Yin. Fault localization via noise reduction on coverage vector[C]. //Proceedings of the 23th International Conference on Software Engineering and Knowledge Engineering (SEKE 2011). 2011:203-206.

[41] E. Antonia, P. L. Francisco. Mutation Operators for WS-BPEL 2.0 [C]. //Proceedings of 21th International Conference on Software & Systems Engineering and their Applications. 2008: 143-151.

[42] E.Botaro, Antonia, 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] M. F. Worboys, H. M. Hearnshaw, et al. Object-oriented data modelling for spatial databases[J]. International Journal of Geographical Information System. 1990, 4(4): 369-383.

[44] 易昭湘, 慕晓冬, 赵鹏, 等. 基于代码检测的软件故障定位方法[J]. 计算机工程, 2007, 33(12): 82-83.

[45] J. Garc á-Fanjul, J. Tuya, et al. Generating test cases specifications for BPEL compositions of web services using SPIN[C]. //Proceedings of International Workshop on Web Services–Modeling and Testing (WS-MaTe 2006). 2006: 83.

[46] Y. S. Ma, J. Offutt, et al. MuJava: an automated class mutation system[J]. Software Testing, Verification and Reliability, 2005, 15(2): 97-133.

[47] J. J. Dom ínguez-Jim énez, A. Estero-Botaro, et al. GAmera: an automatic mutant generation system for WS-BPEL compositions[C]. //Proceedings of European Conference on Web Services. 2009: 97-106.

[48] A. Estero-Botaro, F. Palomo-Lozano, I. Medina-Bulo. Quantitative evaluation of mutation operators for WS-BPEL compositions[C]. //Proceedings of the Third International Conference on Software Testing. 2010: 142-150.

[49] 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.

[50] R. Abreu, P. Zoeteweij. On the accuracy of spectrum-based fault localization[C]. //Proceedings of Testing: Academic and Industrial Conference Practice and Research Techniques-mutation, 2007: 89-98.

[51] W. E. Wong, T. Wei, Y. Qi, et al. A crosstab-based statistical method for effective fault localization[C]. // Proceedings of International Conference on Software Testing, Verification, and Validation. IEEE Computer Society. 2008:42-51.

[52] B. Liblit, M. Naik , et al. Scalable statistical bug isolation[C]. //Proceedings of ACM SIGPLAN Notices. ACM, 2005, 40(6): 15-26.

[53] D. Hao, L. Zhang, Y. Pan, et al. On similarity-awareness in testing-based fault localization[J]. Automated Software Engineering. 2008, 15(2): 207-249.

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
2008.9-2012.6	湖北大学信息管理与信息系统	本科

二、 在学期间从事的科研工作

- 中央高校基本科研业务费资助项目“面向 SOA 的新型软件测试技术与工具”（FRF-SD-12-015A）。

三、 在学期间发表的论文

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**
故障定位，关键谓词，调试工具，服务组装程序，BPEL	公开	TP311	004.4	
学位授予单位名称*		**学位授予单位代码***	**学位类别*** **工学**	**学位级别*** **硕士**
北京科技大学		10008	工学	硕士
论文题名*		**并列题名**		**论文语种***
基于谓词切换的 BPEL 程序故障定位技术与支持工具研究				中文
作者姓名*	郑彩云		**学号***	G20138526
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***
软件工程		软件调试	2.5 年	2016
论文提交日期*	2015 年 12 月 25 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**	
	朱岩			
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*	61 页			
共 33 项，其中带*为必填数据，为 22 项。				