计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180037
56(4):895-905, 2019

面向 WS-BPEL 程序的变异测试优化技术

孙昌爱[1,2] 王 真[1] 潘 琳[1]
[1](北京科技大学计算机与通信工程学院 北京 100083)
[2](宇航智能控制技术重点实验室 北京 100854)
(casun@ustb.edu.cn)

Optimized Mutation Testing Techniques for WS-BPEL Programs

Sun Chang'ai[1,2], Wang Zhen[1], and Pan Lin[1]
[1](*School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing* 100083)
[2](*Science and Technology on Aerospace Intelligent Control Laboratory, Beijing* 100854)

Abstract Business process execution language for Web service (WS-BPEL) is an executable XML-based and process-oriented service composition language. Due to unique features of Web services, such as dynamics, loose coupling, and open deployment and execution environment, it is an important issue how to assure the quality of WS-BPEL programs. Although mutation testing has a strong fault detection capability, it fails to be widely practiced due to the large number of mutants, the long execution period, and the high computation cost. In order to improve the practicability of mutation testing, we investigate how to decrease the cost of mutation testing for WS-BPEL programs, and propose two kinds of optimization techniques from the perspectives of second-order mutation and prioritization of operators. We also develop an integrated tool named μBPEL to support the mutant generation, optimization, and execution of the proposed optimization techniques. Finally, an empirical study has been conducted where six representative WS-BPEL programs are used to validate and evaluate the effectiveness of the proposed optimized mutation testing techniques. Experimental results show that the proposed optimization techniques for WS-BPEL programs are able to reduce the number of mutants without significantly jeopardizing their fault detection effectiveness and thus improve the efficiency of mutation testing.

Key words WS-BPEL; mutation testing; performance optimization; second-order mutation testing; prioritization of mutation operators

摘 要 WS-BPEL(business process execution language for Web service)是一种基于 XML 的、面向过程的可执行服务组装语言.由于 Web 服务的动态性、松耦合特性、部署与运行于开放的网络环境,如何保证 WS-BPEL 程序的可靠性尤显重要.尽管变异测试具有较强的故障检测能力,但由于变异体数量大、执行时间长、计算开销大,不利于在实践中广泛应用.为了增强变异测试的实用性,针对如何降低面向 WS-BPEL 程序的变异测试开销问题,从二阶变异和变异算子优先级角度提出了 2 种面向 WS-BPEL

收稿日期:2018-01-16;**修回日期**:2018-08-14
基金项目:北京市自然科学基金项目(4162040);国家自然科学基金项目(61872039,61370061);航空科学基金项目(2016ZD74004);中央高校基本科研业务费专项资金项目(FRF-GF-17-B29)
This work was supported by the Beijing Natural Science Foundation (4162040), the National Natural Science Foundation of China (61872039, 61370061), the Aeronautical Science Foundation of China (2016ZD74004), and the Fundamental Research Funds for the Central Universities (FRF-GF-17-B29).

程序的变异测试优化技术,开发了相应的集成化支持工具 μBPEL. 使用 6 个 WS-BPEL 程序实例对提出的优化技术的有效性进行验证. 实验结果表明:提出的优化技术可以有效地减少变异体数目而并不显著降低变异测试有效性,提高了变异测试的效率.

关键词 WS-BPEL 语言;变异测试;性能优化;二阶变异测试;变异算子优先级

中图法分类号 TP311

面向服务的架构(service-oriented architecture, SOA)已经成为分布式应用程序的主要开发范式[1]. 由于单一的 Web 服务提供的功能有限,无法满足复杂需求,因此需要将多个服务组装以实现复杂的业务流程. WS-BPEL(business process execution language for Web service)是一种基于 XML 的服务组装语言[2],可以将多个不同的 Web 服务编制起来实现复杂的业务流程. 由于被组装的 Web 服务的动态性、松耦合性、不确定性以及互联网环境的开放性,如何保证 WS-BPEL 程序的可靠性成为一个挑战性问题[3].

变异测试是一种基于故障的软件测试技术[4],具有较强的故障检测能力,广泛用于评估测试用例集的完备性和测试技术的有效性[5]. 然而,由于变异测试产生的变异体数量庞大、等价变异体识别困难、缺乏相应自动化支持工具等原因,变异测试难以在实际中广泛应用. 在 WS-BPEL 变异测试方面,人们提出面向 WS-BPEL 程序的变异算子[6],为 WS-BPEL 程序的变异测试提供基础[7]. 在课题组前期研究工作中[8-10],我们提出了一种面向 WS-BPEL 程序的变异测试框架和支持工具,评估了变异算子的有效性和不同变异算子模拟的故障被检测的难易程度,发现了部分变异算子之间的包含关系.

在前期工作基础上,本文进一步研究如何降低面向 WS-BPEL 程序的变异测试的开销问题,从二阶变异测试和变异算子优先级 2 个方面探索面向 WS-BPEL 程序的变异测试优化技术. 本文的主要贡献有 3 个方面:

1) 提出了 2 种面向 WS-BPEL 程序的变异测试优化技术,即面向 WS-BPEL 程序的二阶变异优化技术和基于变异算子优先级的优化技术.

2) 开发了面向 WS-BPEL 程序的变异测试集成化支持工具 μBPEL,支持 WS-BPEL 程序变异测试的全过程,同时支持本文提出的变异测试优化技术.

3) 使用 6 个 WS-BPEL 程序实例验证并评估提出的优化技术的有效性.

1 相关工作

介绍变异测试优化和 WS-BPEL 测试相关的研究工作.

1.1 变异测试优化技术

变异测试是一种基于故障的软件测试技术[4]. 对待测程序 P 植入符合语法规则的错误,将错误版本的程序称为变异体,植入的故障类型称为变异算子. 对于给定的变异体 M,如果存在某个测试用例,使得 P 和 M 展现出不同的执行行为(通常为不同输出结果),则称这个变异体 M 被"杀死". 如果对于任意的测试用例,P 和 M 的执行结果均相同,则称该变异体 M 为等价变异体.

变异测试具有较强的故障检测能力[5],可以产生较好的测试效果[11]. 然而,变异测试存在的主要不足有 3 个方面:1)变异体数量庞大导致的计算开销;2)等价变异体识别困难;3)缺乏自动化支持工具. 人们主要从变异体选择和变异体执行 2 个角度研究变异测试的优化技术[5].

变异体选择优化关注如何从生成的大量变异体中选择出典型的变异体. Mathur 和 Wong[12] 提出一种变异体随机选择方法,对 Mothra 系统中的 22 种变异算子产生的变异体,采用不同的比例随机选择变异体. 该方法可以大幅度减少测试开销,同时变异评分并没有明显降低. King 和 Offutt[13-14] 提出了一种变异算子选择方法,根据 FORTRAN 语言变异算子的测试有效性对其进行选择,采用选择后的变异算子能产生数目较少且更难被杀死的变异体. Hussain[15] 根据测试用例的检测能力对所有变异体进行聚类分析,选择出变异体. Langdon 等人[16] 应用多目标方法指导生成高阶变异体,该方法可以有效地产生比一阶变异体更难检测的高阶变异体,同时产生等价变异体的概率更小,减少了测试开销. 在前期工作中,我们提出一种路径感知的变异体精简方法,利用程序的结构信息设计变异体精简策略,有效地减少了变异测试的开销[17]. 在 WS-BPEL 程序的变异测试

优化技术方面,我们提出一种基于包含关系的变异算子优化技术[10],分析 WS-BPEL 程序的变异算子产生的变异体检测的包含关系,进行变异算子约简.

变异体执行优化关注减少变异体的执行时间,主要包括变异体检测优化、变异体变异优化和并行执行变异体优化方法[5]. Krauser 等人[18]提出一种基于 SIMD(single instruction multiple data)计算机的并发执行变异体方法. 此方法对变异体的无变异部分执行 1 次,变异的部分进行并发执行,有效地减少变异体执行时间.

1.2 WS-BPEL 测试技术

与传统程序相比,WS-BPEL 程序具有 4 个新特点[19]:1)WS-BPEL 提供一种显式的集成机制组装 Web 服务,而这样的集成在传统程序中是隐式;2)WS-BPEL 程序的服务可以采用不同语言实现,而传统程序中的模块通常由同一种语言实现;3)WS-BPEL 程序表示为 XML 文件,不同于传统应用程序;4)WS-BPEL 通过流(flow)活动支持并发机制、通过连接(link)支持同步机制. WS-BPEL 程序的新特性,导致了 WS-BPEL 程序的故障类型不同于传统应用程序.

人们提出了多种面向 WS-BPEL 程序的测试技术. 文献[20]将基于 WS-BPEL 描述的 Web 服务组装转化为扩展的着色 Petri 网(extended colored Petri net, ECPN),提出了一种基于 ECPN 控制流和数据流结合的测试方法. 针对 WS-BPEL 程序的并发特点,我们提出了一种面向场景的 WS-BPEL 测试用例生成方法[21],并开发了相应的支持工具[22]. Lee 和 Offutt[23]探索了将变异测试应用到 Web 服务测试中. Boonyakulsrirung 等人[24]提出一种面向 WS-BPEL 的弱变异测试框架,通过牺牲变异得分来提高变异测试的效率. Estero-Botaro 等人[7]使用遗传算法来获得变异体集合的子集,选择出高质量的变异体.

目前,人们已经开发了多个面向 WS-BPEL 程序的变异测试支持工具. Domínguez-Jiménez 等人[25]开发了 WS-BPEL 程序变异测试支持工具 GAmera,支持变异体生成、测试用例执行和测试结果统计. Boonyakulsrirung 和 Suwannasart[26]开发了 WeMuTe,支持 WS-BPEL 程序的弱变异测试. 在前期研究工作中,我们开发了一个面向 WS-BPEL 程序的变异测试框架,并开发相应支持工具[8-9],支持变异体生成、变异测试的执行及结果分析.

2 WS-BPEL 程序变异测试优化技术

从二阶变异测试和变异算子优先级 2 个方面,分别提出了变异测试的优化技术 MuSOM 和 MuPri.

2.1 基于二阶变异的优化技术 MuSOM

一种减少变异测试中变异体数量的方法是高阶变异体优化技术,由高阶变异体替代一阶变异体进行测试. 一阶变异体指对原始程序应用一个变异算子且植入一处错误生成的变异体;高阶变异体指对原始程序植入多处错误生成的变异体. 一个高阶变异体可以看成由多个一阶变异体复合而成. 高阶变异体优化技术的提出基于 2 个推测[5]:1)执行一次 M 阶变异体等同于执行 M 个一阶变异体;2)高阶变异体比一阶变异体产生等价变异体的概率小. 基于这 2 个推测可以看出,高阶变异体优化技术主要是通过减少待执行变异体数目和等价变异体识别开销来降低变异测试的开销.

本文提出一种面向 WS-BPEL 程序的二阶变异优化技术(简称为 MuSOM). 二阶变异体生成算法主要有 3 种,分别为 LastToFirst, DifferentOperators, RandomMix 算法[27]. 其中 LastToFirst 算法通过首尾组合 2 个一阶变异体的方式生成二阶变异体,生成的二阶变异体数量为一阶变异体数量的一半;DifferentOperators 算法通过组合不同变异算子生成的一阶变异体生成二阶变异体,生成的二阶变异体数量不小于生成变异体最多的变异算子产生的一阶变异体的数量;RandomMix 算法随机组合 2 个一阶变异体生成二阶变异体,生成的二阶变异体数量为一阶变异体数量的一半. 3 种算法的基本思想都是通过一阶变异体组合生成二阶变异体,但限制生成的二阶变异体数量的策略不同. 其中,LastToFirst 算法和 RandomMix 算法生成的二阶变异体数量相对确定(约为一阶变异体数量的 50%),而 DifferentOperators 算法生成二阶变异体数量不确定. 本文研究的 WS-BPEL 程序可应用的变异算子的种类较少、不同变异算子生成的一阶变异体数量不均衡,DifferentOperators 算法不适用. RandomMix 算法与 LastToFirst 算法相似,主要区别在于一阶变异体的选择次序. 因此,本文采用 LastToFirst 算法生成二阶变异体集合.

LastToFirst 算法[27]按照首尾依次组合 2 个一阶变异体的方式生成二阶变异体,过程如图 1 所示. 其中,P 指待测程序,M_1 至 M_n 代表待测程序 P 的

一阶变异体，$M_{1,n}$ 及 $M_{2,n-1}$ 等代表通过首尾依次组合 2 个一阶变异体生成的二阶变异体.

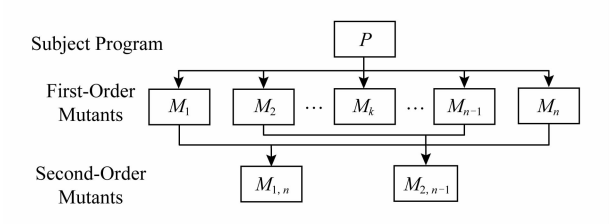

Fig. 1 Procedure of LastToFirst algorithm

图 1 LastToFirst 算法生成二阶变异体过程

MuSOM 描述的二阶变异体生成步骤如下：

1) 根据给定 WS-BPEL 程序 P 生成一阶变异体，得到的变异体集合表示为 $M_{1o}=\{M_1, M_2, \cdots, M_n\}$. 我们按照一阶变异体产生的顺序对变异体命名并编号为 1～n(n 为一阶变异体的总数).

2) 将 M_{1o} 中的一阶变异体首尾依次组合生成二阶变异体，得到二阶变异体集合为 $M_{2o}=\{M_{1,n}, M_{2,n-1}, M_{3,n-2}, \cdots\}$. 具体说来，将编号为 1 与编号为 n 的一阶变异体组合得到二阶变异体 $M_{1,n}$，编号为 2 与编号为 $n-1$ 的一阶变异体组合得到二阶变异体 $M_{2,n-1}$，将编号为 3 与编号为 $n-2$ 的一阶变异体组合得到二阶变异体 $M_{3,n-2}$，以此类推，生成二阶变异体集合. 当 n 为奇数时，将一阶变异体 $M_{(n+1)/2}$ 与 $M_{(n+1)/2+1}$ 组合生成二阶变异体.

2.2 基于变异算子优先级的优化技术 MuPri

在变异测试中，有些变异算子生成的变异体可以被绝大多数测试用例检测("杀死")，有些算子生成的变异体只能被特殊的测试用例检测. 若一些较难检测的变异体都能被给定的测试用例集检测，那么该测试用例集也能检测那些容易被测杀的变异体. 基于以上猜想，我们从变异体执行顺序的角度，提出一种基于变异算子优先级的变异测试优化技术，首先使用可以产生较难检测变异体的变异算子，然后再使用产生较易检测变异体的变异算子.

为了评价变异算子产生的变异体检测的难易程度，引入变异算子质量度量指标. 这里约定 P 为待测的 WS-BPEL 程序；TS 为测试用例集，$TS=\{t_1, t_2, \cdots, t_n\}$，其中 t_i 为测试用例集中第 i 个测试用例，n 为测试用例集的用例总数；MO 表示 WS-BPEL 的变异算子集合，$MO=\{O_1, O_2, \cdots, O_k\}$，其中 O_i 表示第 i 个变异算子，k 为变异算子总数.

将变异算子质量定义为

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

其中，NO_i 表示 O_i 变异算子产生的非等价变异体总数. 不难看出，上述定义可从故障检测率(fault detection rate, FDR)推导而来. 故障检测率 FDR 定义为测试用例检测变异体的比例，广泛用来衡量测试用例的故障检测能力：

$$FDR(M_j^i, TS) = \frac{NC(M_j^i, TS)}{|TS|}, \tag{2}$$

其中，M_j^i 表示 O_i 变异算子的第 j 个变异体，$NC(M_j^i, TS)$表示杀死变异体 M_j^i 的测试用例数目. FDR 值越大，说明测试用例集 TS 检测该变异体的概率越大，该变异体越容易被杀死. 进一步地，变异算子质量 Q_o 越大，则可以检测该变异算子产生的变异体的测试用例越少，该类变异算子的质量越好.

基于变异算子质量定义，我们提出一种基于变异算子优先级的优化技术(简称为 MuPri)，通过衡量变异算子质量，为变异算子分配测试优先级. 具体过程为：首先，对大量的 WS-BPEL 程序进行变异测试；然后，根据测试结果计算变异算子的质量，按照质量由高到低的顺序为其排序，质量好的变异算子在变异测试中分配较高的优先级，质量差的变异算子分配较低的优先级，即按照变异算子优先级顺序指导生成变异体集合.

算法 1. 测试用例排序算法.

输入：实例程序 P、变异算子集合 O、有序的测试用例集 $TS=\{t_1, t_2, \cdots, t_m\}$；

输出：排序后的测试用例集 TS'.

① 从 O 中选择适用于 P 的变异算子集合 OP；
② 对 OP 中变异算子按算子质量由高到低排序，得到变异算子序列 OP：$OP_1, OP_2, \cdots, OP_n$；
③ 令 $i=1$，$TS'=\varnothing$；
④ WHILE $i \leqslant n$ DO
⑤ 生成 OP_i 的一阶变异体集合 $MO_i(P)=\{m_1, m_2, \cdots, m_k\}$；
⑥ WHILE $MO_i(P) \neq \varnothing$ DO
⑦ 添加一个测试用例 t 到 TS'；
⑧ FOR ALL $m_j \in MO_i(P)$ DO
⑨ 以 t 测试用例执行 m_j；
⑩ IF m_j 被"杀死" THEN
⑪ 从 $MO_i(P)$ 中删除 m_j；
⑫ END IF
⑬ END FOR
⑭ END WHILE
⑮ $i=i+1$；
⑯ END WHILE
⑰ 输出测试用例集 TS'.

MuPri 技术不仅可以依据变异算子的优先级,优先使用质量好的变异算子生成变异体,提高变异体集质量,还可以为测试用例集合排序,得到故障检测效率更高的测试用例集,解决测试用例优先级问题. MuPri 实现的测试用例排序的过程如算法 1 所示.

3 支持工具 μBPEL

在前期工作中[8-9],我们开发了一个面向 WS-BPEL 程序的变异测试支持工具 μBPEL,支持变异体的生成、测试用例的执行和测试结果验证. 本文通过扩展 μBPEL 进一步支持本文提出的变异测试优化技术. 图 2 描述了 μBPEL 工具的系统结构. 各个组件的功能描述如下:

1) 变异体生成. 负责为待测 WS-BPEL 程序生成一阶或二阶变异体.

① WS-BPEL 解析. 解析 WS-BPEL 程序.

② 算子管理器. 针对各种变异算子的匹配与操作,实现对相应节点的变异处理.

③ XML 文件读/写. 负责 WS-BPEL 程序文件的读入和变异体输出.

④ 变异体生成. 生成一阶或二阶变异体.

2) 变异体优化. 支持变异体随机选择优化.

① 参数配置. 接收用户输入的待优化的变异体集合路径和变异体精简比例.

② 变异体获取. 根据变异体精简比例,从变异体集合中随机获取相应数目的变异体.

3) 测试用例生成. 根据 WS-BPEL 原始文件,输出期望的测试用例. 课题组前期研发了 2 种测试用例生成工具[22],场景用例生成将 WS-BPEL 程序转换为图模型,基于给定的覆盖准则生成测试场景和数据;随机用例生成根据用户输入的约束条件,随机生成满足条件的测试用例.

4) 变异测试执行. 执行测试用例并获取输出结果.

① 执行环境配置. 根据 WS-BPEL 的配置信息,获取 WS-BPEL 服务的端口号和操作名称配置执行环境.

② 程序选取. 通过用户输入的文件路径,依次获取原始程序和变异体程序文件.

③ 用例读取. 读入并解析相应的测试用例文件,获取用例输入变量的类型、数目、值及用例个数等信息.

④ 待测程序执行. 调用 WS-BPEL 引擎依次对原始程序和变异体执行测试用例集并输出结果.

5) 测试结果评估. 负责对输出结果进行统计分析,由结果统计和报告获取 2 个模块组成.

① 结果统计. 对执行相同测试用例的原始程序和变异体的输出结果逐一进行对比. 若二者结果不同,表明该测试用例将变异体“杀死”,标记为“T”;否则,记为“F”. 依次记录变异体被测杀的状态,并统计出针对变异体的每个测试用例集合的故障检测率信息.

②报告获取. 根据结果统计的输出结果,计算

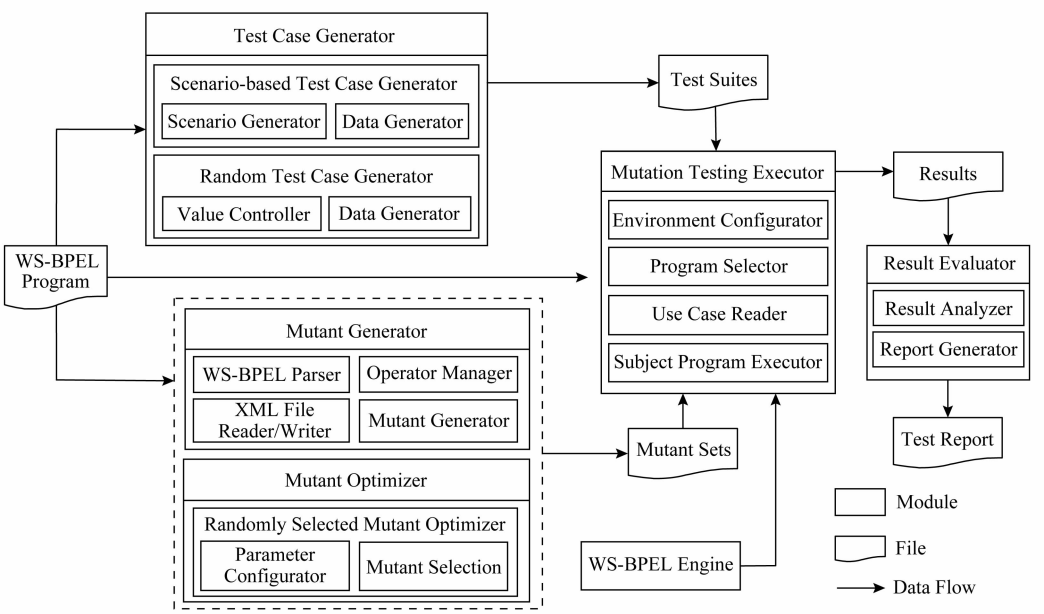

Fig. 2 Architecture of μBPEL

图 2 μBPEL 工具系统结构图

变异得分并生成报告,包括变异体数目、被杀死变异体数目和变异得分信息.

4 实验评估

采用经验研究验证与评估本文提出的面向WS-BPEL程序的变异测试优化技术的有效性.

4.1 实验对象

实验对象包括6个WS-BPEL程序实例:SupplyChain实例[19](P_1)、SmartShelf实例(P_2)[19]、SupplyCustomer实例(P_3)[2]、LoanApproval实例(P_4)[2]、CarEstimate实例(P_5)①和TravelAgency[28]实例(P_6).表1总结这些程序实例的具体信息.

Table 1 WS-BPEL Programs

表1 WS-BPEL程序实例的基本信息

Program	Basic Functionality	Number of Services	Lines of Code
P_1	Management of supply chains	2	50
P_2	Management of commodity shelves	14	194
P_3	Management of project orders	5	122
P_4	Examination of loan application	3	120
P_5	Assessment of car repairs	7	121
P_6	Booking of travels	9	543

4.2 实验指标

本文使用变异得分(mutation score, MS)、故障检测率FDR(见式(2))、测试用例序列检测故障的平均百分比(average of the percentage of faults detected, $APFD$)这3个指标度量变异测试优化方法的有效性.

1) 变异得分MS.对于给定的测试用例集TS,杀死变异体数量与非等价变异体数量的比例[4],用来衡量测试用例集的充分程度.变异得分计算为

$$MS(P,TS)=\frac{N_k}{N_m-N_e}, \tag{3}$$

其中,P代表被测程序,N_k表示被杀死的变异体数量,N_m表示变异体总数量,N_e代表等价变异体的数量.变异得分越高,说明测试用例集"杀死"的变异体越多,测试用例集越有效.

2) 测试用例序列检测故障的平均百分比$APFD$.用于评价测试用例集的故障检测效率,常用于衡量不同测试用例优先级技术的排序效果[29].$APFD$的计算公式为

$$APFD(TS,P)=1-\frac{\sum_{i=1}^{m}reveal(i,TS)}{m\times n}+\frac{1}{2n}, \tag{4}$$

其中,TS表示具有特定序列的测试用例集,P表示待测程序,n表示测试用例的个数,m表示被检测故障的总数,$reveal(i,TS)$表示最早检测出第i个故障所执行的测试用例的位置.$APFD$量化了测试用例序列的效率和效能,即$APFD$的数值越大,测试用例集故障检测效率越高.本文采用$APFD$来度量序列化的测试用例的故障检测效率.

4.3 实验设置

讨论与2种变异测试优化技术评估相关的变异体生成和测试用例集.

1) 一阶和二阶变异体生成.针对表1中的WS-BPEL程序实例,采用本文开发的μBPEL工具生成一阶变异体集合(记为M_{1o})和二阶变异体集合(记为M_{2o}).表2总结了生成的一阶变异体和二阶变异体情况.

Table 2 First-Order and Second-Order Mutants of WS-BPEL Programs

表2 WS-BPEL程序实例的一阶和二阶变异体

Program	Number of Mutants		Number of Applied Operators
	M_{1o}	M_{2o}	
P_1	34	17	11
P_2	170	85	11
P_3	73	37	11
P_4	85	43	17
P_5	54	27	5
P_6	171	86	14

2) 测试用例生成.在前期工作中[10],我们采用等价类划分[30]、边界值分析[30]、面向场景测试技术[22]和随机测试技术[30]这4种技术生成测试用例.为了减少不同测试用例生成技术对变异测试结果的影响,最终生成5组测试用例集,分别记为T_x,T_y,T_z,T_u,T_w,如表3所示.$|T_x|$,$|T_y|$,$|T_z|$,$|T_u|$,$|T_w|$表示WS-BPEL程序实例对应的每组测试用例集的大小.需要指出的是,测试用例集的数量依赖于程序的规模和所使用的测试用例生成技术.为了保证实验的公平性,本文采用这些测试用例集评估优化技术的有效性.

① http://www.activevos.com/developers/sample-apps

Table 3 Test Suits of WS-BPEL Programs

表 3 WS-BPEL 实例的测试用例集合

Program	$\|T_x\|$	$\|T_y\|$	$\|T_z\|$	$\|T_u\|$	$\|T_w\|$
P_1	6	10	15	19	30
P_2	22	35	50	74	90
P_3	7	12	18	24	30
P_4	12	18	25	31	40
P_5	10	20	30	36	40
P_6	4	7	10	14	20

4.4 优化技术评估结果与分析

4.4.1 MuSOM 技术评估结果

MuSOM 技术有效性的评估过程描述如下:首先,采用测试用例集 T_x 对一阶变异体集合 M_{1o} 和二阶变异体集合 M_{2o} 进行变异测试,得到能够杀死所有变异体的测试用例集 TC;然后,采用测试用例集 TC 对一阶变异体集合 M_{1o} 进行变异测试,统计变异得分.

表 4 总结了生成的二阶变异体集合中的等价变异体情况;图 3 统计了一阶变异体集合和二阶变异体集合的变异得分情况.

Table 4 Equivalent Mutant of WS-BPEL Programs

表 4 WS-BPEL 程序实例的等价变异体情况

Mutant	Number of Equivalent Mutant of Program					
	P_1	P_2	P_3	P_4	P_5	P_6
M_{1o}	0	13	5	7	7	16
M_{2o}	0	0	0	0	0	0

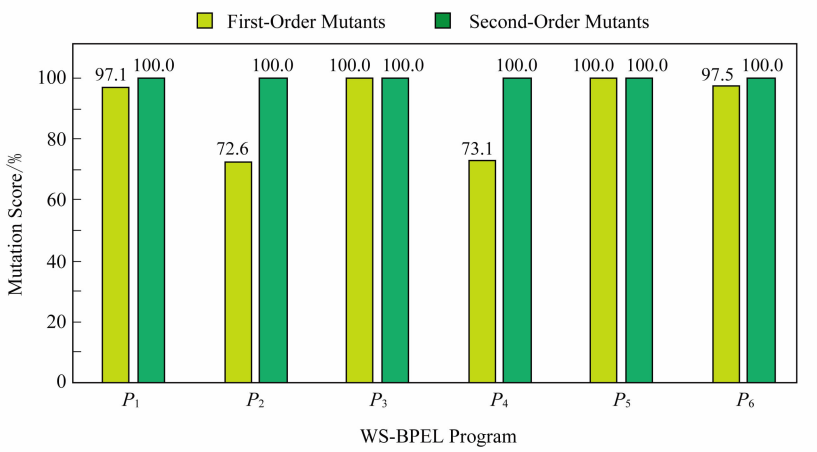

Fig. 3 Mutation score of WS-BPEL programs

图 3 WS-BPEL 程序的一阶与二阶变异体的变异得分

上述实验结果表明:

1) 在使用同样数目和种类的变异算子情况下,M_{2o} 集合数量约为 M_{1o} 的一半,相对于一阶变异测试,减少了约 50% 的待测变异体(如表 2 所示);M_{2o} 中的等价变异体数目(NE)总和为 0,而 M_{1o} 中存在 48 个等价变异体(如表 4 所示),主要原因是,与一阶变异体相比,二阶变异体中植入了多处错误,降低了等价变异体的生成概率.由此可见,MuSOM 技术极大地减少了等价变异体的出现概率(一阶变异测试中等价变异体约为 28%),大幅度降低等价变异体识别带来的计算开销.

2) 在采用相同的测试用例集情况下,二阶变异测试的变异得分均为 100%,而一阶变异测试的变异得分有所不同.其中,P_3 和 P_5 实例中的变异得分为 100%;P_1 和 P_6 实例中的变异得分分别是 97.1% 和 97.5%,接近于 100%;P_2 和 P_4 实例的变异得分分别是 72.6% 和 73.1%.主要原因是,二阶变异体中存在多处错误,更容易被检测出来.相应地,在相同测试用例集情况下,二阶变异测试的变异得分高于一阶变异测试.

综上所述,相对于传统的(一阶)变异测试而言,二阶变异测试技术可以减少约 50% 的变异体和减少约 28% 的等价变异体识别开销,同时并没有大幅度降低衡量测试用例集充分性的能力.

4.4.2 MuPri 技术评估结果

MuPri 技术依据变异算子质量对测试用例集进行排序.通过对比排序前后的测试用例集的 $APFD$ 评估 MuPri 技术的有效性.具体步骤有 3 个:

1) 变异算子优先级排序.首先采用测试用例集 T_x,T_y,T_z,T_u,T_w 分别执行实例程序和其一阶变异体集合 M_{1o},计算变异得分(MS)和故障检测率(FDR).需要说明的是,如下变异算子在实例程序

中不适用:EAA,EEU,ELL,EMD,EMF,AFP,AIS,AWR,AJC,APM,APA,XMC,XMT,XTF,XER,XEE;变异算子 ECC 和 EAP 产生的均为等价变异体.限于篇幅,我们不列出每个变异算子的变异得分和故障检测率(参考文献[9]).

表 5 列出了可适用的变异算子优先级排序结果.依据变异算子质量 Q_o 的平均值,对变异算子排序并分配优先级.优先级的数值越小,表示该变异算子的优先级越高.

Table 5 Priority of Mutation Operators

表 5 变异算子优先级

Mutation Operator	Q_o of Program/%						Average /%	Priority
	P_1	P_2	P_3	P_4	P_5	P_6		
XMF				100.0			100.0	1
ISV				88.6			88.6	2
ECN				82.9		71.9	77.3	3
CDE	50.0	54.7	50.0	75.8		50.0	56.1	4
CCO	50.0	54.7	50.0	75.8		50.0	56.1	5
CDC	50.0	54.7	50.0	75.8		50.0	56.1	6
EIU				59.4		50.0	54.7	7
EAN				59.4		50.0	54.7	8
ERR	40.0	49.7	58.0	73.2		40.0	52.2	9
AIE	41.8	54.5	50.8			50.0	49.3	10
ASF	29.1	11.3	24.6	88.6	0.0	42.9	32.7	11
AEL	30.9	32.0	48.1	30.5	0.0	32.3	29.0	12
CFA	30.9	32.0	27.9	12.5	0.0	32.3	22.6	13
EIN	0.0	9.3	24.6	57.7		0.0	18.3	14
ASI	14.6	14.5	22.9	0.0	0.0	18.7	11.7	15
ACI	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16

2) 测试用例集排序.针对每个 WS-BPEL 程序 P,首先得到变异得分 100% 的测试用例集 TS,使用变异算子优先级得到排序的测试用例集 TS',分别计算测试用例集 TS 和 TS' 的 $APFD$ 值.以 SupplyChain 程序为例,表 5 列出了适用的变异算子优先级序列为:CDE→CCO→CDC→ERR→AIE→ASF→AEL→CFA→EIN→ASI→ACI.采用测试用例排序过程(算法 1)对 SupplyChain 实例的测试用例集进行排序,结果如表 6 所示.表 6 中,"√"表示测试用例集 TS 中第 1 个将该变异体"杀死"的用例.最终得到测试用例集 TS 的顺序是"T_1→T_2→T_3".

Table 6 Results of Executing Test Suite (*TS*) on SupplyChain

表 6 SupplyChain 程序执行测试用例 *TS* 结果

Priority	Applicable Operator	ID of Generated Mutants	Test Suite (TS)		
			T_1	T_2	T_3
1	CDE	1	√		
		2		√	
2	CCO	1	√		
		2		√	
3	CDC	1	√		
		2		√	
4	ERR	1	√		
		2			√
		3	√		
		4		√	
		5	√		
5	AIE	1		√	
6	ASF	1	√		
		2	√		
7	AEL	1	√		
		2	√		
		3	√		
		4	√		
		5	√		
		6		√	
		7	√		
8	CFA	1	√		
		2	√		
		3	√		
		4	√		
		5	√		
		6		√	
		7	√		
9	EIN	1	√		
10	ASI	1	√		
		2	√		
		3	√		
		4	√		
11	ACI	1	√		

Note:"√" means the first test case that kills the mutant.

使用排序后的测试用例集 TS 执行没有排序的变异体集合,结果如表 7 所示.其中"√"表示第 1 个将变异体"杀死"的测试用例,Location 表示该用例在 TS 中的位置,"×"表示测试用例不能"杀死"

变异体,“~”表示测试用例没有执行.最终计算得到 $APFD$ 的值是 74.5%.类似地,我们可以得到其他 WS-BPEL 程序优化前后的 $APFD$ 值.

Table 7 Results of Mutation Testing

表 7 变异体测试结果

Applicable Operator	ID of Generated Mutants	Test Suite (TS)			Location
		T_1	T_2	T_3	
ERR	1	√	~	~	1
	2	×	×	√	3
	3	√	~	~	1
	4	×	√	~	2
	5	√	~	~	1
EIN	1	√	~	~	1
ACI	1	√	~	~	1
ASF	1	√	~	~	1
	2	√	~	~	1
AEL	1	√	~	~	1
	2	√	~	~	1
	3	√	~	~	1
	4	√	~	~	1
	5	√	~	~	1
	6	×	√	~	2
	7	√	~	~	1
AIE	1	×	√	~	2
ASI	1	√	~	~	1
	2	√	~	~	1
	3	√	~	~	1
	4	√	~	~	1
CFA	1	√	~	~	1
	2	√	~	~	1
	3	√	~	~	1
	4	√	~	~	1
	5	√	~	~	1
	6	×	√	~	2
	7	√	~	~	1
CDE	1	√	~	~	1
	2	×	√	~	2
CCO	1	√	~	~	1
	2	×	√	~	2
CDC	1	√	~	~	1
	2	×	√	~	2

Notes:“√” means that the test case kills the mutant; “×” means that the test case cannot kill the mutant; “~” means that the test case is not executed to kill the mutant.

3) 比较排序前后测试用例集的有效性.表 8 列出了每个 WS-BPEL 程序的变异算子优化前后的测试用例集的 $APFD$.

Table 8 Comparison Between *APFD* of Original Test Suite and that of Prioritized Test Suite Using MuPri

表 8 变异算子优化前后的 *APFD* 比较 %

Program	Prioritized $APFD$	Original $APFD$
P_1	74.5	69.6
P_2	80.9	71.9
P_3	94.1	94.1
P_4	71.5	52.8
P_5	50	50
P_6	70.2	61.3

上述实验结果表明:使用 MuPri 优化技术后得到的测试用例序列的 $APFD$ 值都大于或等于优化前的 $APFD$ 值.MuPri 优化技术优先使用较难被检测的变异算子生成变异体,采用这样的变异体集合为测试用例集排序,可以得到故障检测效率更高的测试用例集.因此,MuPri 技术通过对变异算子进行优先级排序提高了测试用例集的故障检测效率.

5 总 结

本文针对 WS-BPEL 程序的变异测试开销大的问题,提出了 2 种面向 WS-BPEL 程序的变异测试优化技术 MuSOM 和 MuPri.其中,MuSOM 从变异体数量精简角度,将二阶变异应用 WS-BPEL 测试;MuPri 提出了变异算子质量的概念,通过度量变异算子优先级指导变异体的使用顺序.开发了面向 WS-BPEL 程序的变异测试支持工具 μBPEL,该工具实现了本文提出的优化技术,有助于对 WS-BPEL 程序进行高效的变异测试.最后,采用 6 个 WS-BPEL 程序实例验证并评估了提出的变异测试优化技术的有效性.实验结果表明:本文提出的变异测试优化技术极大地降低了 WS-BPEL 程序的变异测试开销,提高了变异测试的效率.

未来将在 2 个方面进一步开展研究工作:1)与其他相关的优化技术进行比较,评估所提出的优化技术的效率;2)扩展验证的 WS-BPEL 程序集,特别地,目前的程序集适用的变异算子种类较少,需要采用更大规模的实例程序对优化技术进行更全面的评估.

参 考 文 献

[1] Papazoglou M P, Traverso P, Dustdar S, et al. Service-oriented computing: A research roadmap [J]. International Journal on Cooperative Information Systems, 2008, 17(2): 223-255

[2] Eviware. Web services business process execution language [OL]. 2012 [2017-06-11]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[3] Sun Chang'ai. On open issues on SOA-based software development [OL]. 2011[2017-06-11]. http://www.paper.edu.cn/releasepaper/content/201107-461 (in Chinese)
(孙昌爱. SOA 软件开发中的若干开放问题探讨[OL]. 2011 [2017-06-11]. http://www.paper.edu.cn/releasepaper/content/201107-461)

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

[5] Chen Xiang, Gu Qing. Mutation testing: Principal, optimization and application [J]. Journal of Frontiers of Computer Science and Technology, 2012, 6(12): 1057-1075 (in Chinese)
(陈翔, 顾庆. 变异测试: 原理、优化和应用[J]. 计算机科学与探索, 2012, 6(12): 1057-1075)

[6] Estero-Botaro A, Palomo-Lozano F, Medina-Bulo I. Mutation operators for WS-BPEL 2.0 [OL]. 2008 [2016-06-10]. https://www.researchgate.net/publication/255992637_Mutation_operators_for_WS-BPEL_20

[7] Estero-Botaro A, Palomo-Lozano F, Medina-Bulo I. Quantitative evaluation of mutation operators for WS-BPEL compositions [C] //Proc of the 3rd Int Conf on Software Testing, Verification, and Validation Workshops (ICSTW). Los Angeles, CA: IEEE Computer Society, 2010: 142-150

[8] Wang Qiaoling. Research on mutation testing technique for BPEL programs and its supporting tool [D]. Beijing: University of Science and Technology Beijing, 2015 (in Chinese)
(王巧玲. 面向 BPEL 程序的变异测试技术与支持工具研究[D]. 北京: 北京科技大学, 2015)

[9] Pan Lin. Research on optimization techniques of mutation testing for BPEL programs and their supporting tool [D]. Beijing: University of Science and Technology Beijing, 2017 (in Chinese)
(潘琳. BPEL 程序的变异测试优化技术与集成化支持工具研究[D]. 北京: 北京科技大学, 2017)

[10] Sun Chang'ai, Pan Lin, Wang Qiaoling, et al. An empirical study on mutation testing of WS-BPEL programs [J]. The Computer Journal, 2017, 60(1): 143-158

[11] Just R, Jalali D, Inozemtseva L, et al. Are mutants a valid substitute for real faults in software testing? [C] //Proc of the 22nd Symp on the Foundations of Software Engineering (FSE'14). New York: ACM, 2014: 654-665

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

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

[14] Offutt A J, Rothermel G, Zapf C. An experimental evaluation of selective mutation [C] //Proc of the 15th Int Conf on Software Engineering (ICSE'93). Los Angeles, CA: IEEE Computer Society, 1993: 100-107

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

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

[17] Sun Chang'ai, Xue Feifei, Liu Huai, et al. A path-aware approach to mutant reduction in mutation testing [J]. Information and Software Technology, 2016, 81(1): 65-81

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

[19] Sun Chang'ai, Zhai Yimeng, Shang Yan, et al. BPELDebugger: An effective BPEL-specific fault localization framework [J]. Information and Software Technology, 2013, 55(12): 2140-2153

[20] Mou Xiaoling. Research of Web service composition test based on extended colored Petri net [D]. Chongqing: Southwest University, 2012 (in Chinese)
(牟小玲. 基于扩展着色 Petri 网的服务组合测试研究[D]. 重庆: 西南大学, 2012)

[21] Sun Chang'ai, Zhao Yan, Pan Lin, et al. Automated testing of WS-BPEL service compositions: A scenario-oriented approach [J]. IEEE Transactions on Services Computing, 2018, 11(4): 616-629

[22] Zhao Yan. A scenario-based approach to automatic test case generation for BPEL programs and its supporting tool [D]. Beijing: University of Science and Technology Beijing, 2014 (in Chinese)
(赵彦. 基于场景的 BPEL 测试用例自动生成技术与工具研究[D]. 北京: 北京科技大学, 2014)

[23] Lee S C, Offutt J. Generating test cases for XML-based Web component interactions using mutation analysis [C] //Proc of the 12th Int Symp on Software Reliability Engineering (ISSRE'01). Los Angeles, CA: IEEE Computer Society, 2001: 200-209

[24] Boonyakulsrirung P, Suwannasart T. A weak mutation testing framework for WS-BPEL [C] //Proc of the 8th Int Joint Conf on Computer Science and Software Engineering (JCSSE'11). Piscataway Township, NJ: IEEE, 2011: 313-318

[25] Domínguez-Jiménez J J, Estero-Botaro A, García-Domínguez A, et al. GAmera: An automatic mutant generation system for WS-BPEL compositions [C] //Proc of the 7th European Conf on Web Services (ECOWS'09). Los Alamitos, CA: IEEE Computer Society, 2009: 97-106

[26] Boonyakulsrirung P, Suwannasart T. WeMuTe-A weak mutation testing tool for WS-BPEL [C] //Proc of the Int MultiConference of Engineers and Computer Scientists (IMECS'12). Hong Kong: Newswood Limited, 2012: 810-815

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

[28] Sun Chang'ai, Khoury E, Aiello M. Transaction management in service-oriented systems: Requirements and a proposal [J]. IEEE Transactions on Services Computing, 2011, 4(2): 167-180

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

[30] Myers G J. The Art of Software Testing [M]. 2nd ed. Hoboken, NJ: John Wiley and Sons, 2004

Sun Chang'ai, born in 1974. Professor at the School of Computer and Communication Engineering, University of Science and Technology Beijing. His main research interests include service-oriented computing and software testing.

Wang Zhen, born in 1994. PhD candidate at the School of Computer and Communication Engineering, University of Science and Technology Beijing. Her main research interests include service-oriented computing and software architecture.

Pan Lin, born in 1992. Master candidate at the School of Computer and Communication Engineering, University of Science and Technology Beijing. Her main research interests include service-oriented computing and software testing.