软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(3):504−516 [doi: 10.3724/SP.J.1001.2012.04065]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

死路径语义下 BPEL 路径敏感性缺陷分析*

杨学红+, 黄俊飞, 宫云战

(北京邮电大学 计算机科学与技术系,北京 100876)

Path Sensitive Defect Analysis for BPEL Under Dead Path Semantic

YANG Xue-Hong+, HUANG Jun-Fei, GONG Yun-Zhan

(State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

+ Corresponding author: E-mail: yangxuehong@bupt.edu.cn

Yang XH, Huang JF, Gong YZ. Path sensitive defect analysis for BPEL under dead path semantic. *Journal of Software*, 2012,23(3):504−516. http://www.jos.org.cn/1000-9825/4065.htm

Abstract: Software defect is an important indicator for measuring the adequacy of software testing. In order to improve the reliability and robustness of Web service composition based BPEL, this paper proposes a defect detecting method which combines the dead path into a path sensitive analysis. Dead path is a special semantic of BPEL, which has no execution information, but can connect two executed path segments. Using the abstract value of variable and its interval arithmetic, this method can identify an unreachable path and dead path, and also merge the conditions of property state at join points. A case study about uninitialized variable detecting which is related to dead path and executing path is given throughout the analysis, and verification is implemented to illustrate the effectiveness of this approach.

Key words: BPEL; dead path elimination; path sensitive analysis; dataflow analysis; defect detection

摘 要: 软件缺陷是衡量软件测试充分性的一项重要指标,为了提高基于 BPEL 的组合 Web 服务流程的可靠性和健壮性,提出了一种死路径语义下路径敏感的缺陷检测方法.死路径是 BPEL 提供的特殊语义,不具有任何执行信息,但是可以连接两个可执行路径片段.为了避免死路径对检测精度的影响,将死路径和路径条件有机地结合起来,采用了变量的抽象取值范围来表示流程的执行状态,即属性状态条件.通过状态条件中变量抽象取值范围为空来识别不可达路径及死路径,并在汇合节点进行了属性状态条件的合并.采用一个既与死路径相关又与执行路径相关的未初始化变量的缺陷检测贯穿整个分析与验证过程,进而说明了该方法的有效性.

关键词: BPEL;死路径清除;路径敏感分析;数据流分析;缺陷检测

中图法分类号: TP311 **文献标识码**: A

随着 SOA 的日益普及,服务组合在集成松散的、异构服务中表现出超前的威力,有利于企业应用开发和服务重用.所谓服务组合,是将 Internet 中已经存在的简单 Web 服务按照一定的逻辑顺序集成在一起,从而完成某个特定任务,同时将其自身暴露为一个新的复杂 Web 服务.然而,组合后的服务并不总是表现的很完美,往往存

* 基金项目: 国家自然科学基金(91018002); 国家高技术研究发展计划(863)(2009AA012404)
收稿时间: 2011-03-09; 修改时间: 2011-05-06; 定稿时间: 2011-06-17

在一些设计或性能上的缺陷,很难满足稳定性和高可靠性要求,而这些直接决定了服务组合的满意度,因此,验证和测试是服务组合相关研究领域中日益引起关注的一个重要话题.BPEL[1]作为一种服务组合语言已得到广泛认可,现有的关于组合服务正确性的验证主要从模型检查和动态测试两个角度进行了深入研究.控制流和数据流作为构建 BPEL 流程的基本信息,其正确与否是流程正确性的前提保证,错误的控制结构以及数据不一致等故障往往源自于此.目前,有关 BPEL 控制流分析和验证技术主要有进程代数[2]、Petri 网[3−5]和自动机[6−9];对数据流的验证主要集中在消息匹配[10,11]、数据依赖[12,13]等方面.

众所周知,缺陷是导致软件故障和漏洞的重要原因,不同测试方法检测缺陷的能力也不相同.一个好的缺陷检测系统能够发现程序中可能存在的各种缺陷.BPEL 语法和语义特性使其除了传统缺陷(如死锁)以外还有自己独特的缺陷,如 link 的非法使用.所以,尽管已经出现了各种方法对 BPEL 流程正确性进行验证,但是当前的方法都不能完美地证明流程中不存在某些不期望的情况,如不可达活动、消息接收冲突、变量使用异常等.为此,文献[14]提出了一种针对 BPEL 缺陷的静态检测方法,采用状态机对缺陷进行建模,然后依据提出的改进数据流迭代检测算法进行缺陷分析,如果发现缺陷状态机进入错误状态,则报告一个相应的错误.

本文的主要贡献是,在已有工作的基础上,提出了一种死路径语义下路径敏感的缺陷检测算法:在进行属性状态变迁的判断时,根据死路径的语义定义了属性状态的 3 种可能形式,极大地减少了误报和漏报发生.通过在传统路径敏感分析方法中加入死路径信息,并在控制流汇合节点合并相同属性状态的条件信息,根据路径条件信息中变量抽象取值区间[25]为空识别不可达路径和可达路径上的死路径,避免了全路径分析的空间爆炸问题.

1 BPEL 语言

业务流程执行语言(BPEL)用来描述基于 Web service 的业务流程,它是一种基于图形结构和块结构的混合工作流语言,源自于 IBM 的 WSFL 和微软的 XLANG.活动是 BPEL 流程的基本语句,可以分为基本活动和结构化活动.通过基本活动调用外部伙伴服务或更改流程状态;结构化活动用来限定基本活动间的执行顺序.每个 BPEL 流程可以划分为 4 部分:与外部伙伴关系的声明(partnerlinks)、流程变量的声明(variables)、处理器声明(故障处理器、事件处理器等)以及具体流程行为的描述.

link 结构定义在 flow 中,每个 link 连接且只能连接一个源活动和一个目标活动,link 上可以携带一个具有默认值为 true 的变迁条件(transition condition,简称 tc),每一个目标活动还可以关联一个汇合条件(join condition,简称 jc),它是所有以该活动为目标活动的 link(记为输入 link)状态的逻辑表达式,默认情况下是所有输入 link 状态的逻辑或(OR).BPEL 具有这样的特殊语义:如果某一活动所有输入 link 的状态变量 *tc* 计算完毕(link 是一个具有三元值的状态变量:ture,false 和 unset,当 link 连接的源活动尚未成功完成之前,其值为 unset;否则,根据 *tc* 计算状态变量),才可以判断 *jc* 的值.如果为 true,则该活动正常执行;否则,活动不能执行,并且设置所有以该活动为源活动的 link(记为输出 link)的状态值为 false.该情况将一直传递下去,直到遇到 *jc* 为 true 的活动.这就是 BPEL 的死路径清除(DPE),有关死路径的详细信息可以参见文献[15].

2 缺陷分析及相关定义

2.1 一个误报的例子

不同于其他程序设计语言,BPEL 中定义的变量使用查询操作进行访问,每个查询访问变量的一个特定位置.实际上可以采用格来形式化表示变量查询子元素之间的关系,格中每个元素代表变量的一次查询,格上的操作表示这些查询操作之间的包含关系.可以定义格上的比较操作符为$\sqsupseteq$,则 $n \sqsupseteq m$ 表示 n 是 m 的父节点,并且 n 和 m 都是变量的查询子元素.格中的最大元素代表整个变量,最小元素用$\perp$表示.例如,变量$\$var$ 是具有如图 1 所示的复杂类型,其中,$price \sqsupseteq amount, var \sqsupseteq car, var \sqsupseteq price$ 等.假设存在 3 个逻辑上顺序执行的赋值操作 w_1,w_2,w_3 分别对该变量的不同部分$\$var,\$var/car,\$var/car$ 进行赋值,则 w_3 的操作将覆盖 w_2 的赋值,而 w_2 和 w_3 都将覆盖掉 w_1 的部分值($\$var/car$ 被覆盖,而$\$var/price$ 的值不受影响),因此最终变量$\$var$ 中的值是 w_1 和 w_3 合并后的值.

这说明只要同一变量的多个查询操作之间不存在包含(⊒)关系,则 BPEL 变量支持多个活动同时对同一变量的读写操作.这就使 BPEL 的未初始化变量检测不同于传统程序语言,它需要关注变量的整体赋值或部分赋值.

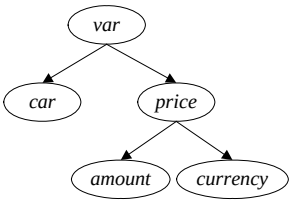


Fig.1 Complex variable type

图 1 复杂变量类型

BPEL 执行时的死路径清除(DPE)语义,给静态分析带来了更大挑战.图 2 为一个 DPE 影响检测精度的例子,图形左右两边分别给出了执行活动 if 前传统路径敏感静态分析和 DPE 语义下路径敏感分析的比较.两者均采用了变量的抽象取值(即区间)表示程序执行信息[16],其中,init,decl 分别表示变量的初始化和声明.

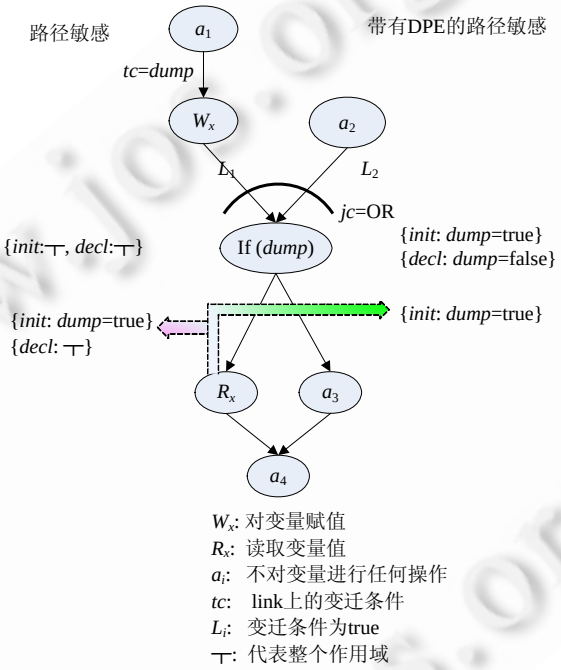


Fig.2 An example of false positive

图 2 一个误报的例子

传统路径敏感分析考虑了不同路径上的条件信息(分支条件)[17,18]以及分支间的组合关系,能够区分控制流图上的不同路径信息,因此属性状态机在 if 活动处可能的状态集合为{*init*:⊤},{*decl*:⊤},说明变量 *x* 的 *decl* 状态能够到达 if 的 true 分支上(即活动 R_x).在带有 DPE 语义的扩展路径敏感分析中,需要考虑同步边(link)上 *tc* 表达式中变量的取值情况,当 *dump*=false 时,W_x 不执行(W_x 标识一条死路径的开始,并设置 l_1 的状态值为 false),因此隐含有{*decl*:*dump*=false}{*init*:*dump*=true}.由于 if 活动的 join condition 为其输入 link 的逻辑或(OR),a_2 一定能够执行(l_2 的状态值为 true,a_2 没有对变量进行任何操作,因此有{*decl*:⊤}),导致 if 活动总会被执行.当到达 if 时,通过相同状态的属性条件合并,使得 *decl* 的属性状态条件为 *dump*=false.变量 *x* 的 *decl* 状态不会到达 if 的真

分支(*dump*=true)上,消除了传统路径敏感分析造成的误报.

同样,如果 *jc*=AND(所有输入 link 状态的逻辑与),在具有 DPE 语义的路径敏感分析中,该流程片段是安全的.因为当 *dump*=true 时,一定能够执行 W_x,所以 R_x 是安全的;而当 *dump*=false 时,从 W_x 开始直到 $a_4(W_x \rightarrow \text{if} \rightarrow R_x(a_3) \rightarrow a_4)$都处在死路径上,因此不具有任何执行信息.此时,{*decl*:*dump*=false}会沿着路径一直传递下去.

死路径可以看作是程序执行路径上一段特殊的路径片段,在该路径片段上,程序不具有任何执行信息(此时,任何的状态改变都是无效的),但其可以连接两个可执行的路径片段,将进入死路径前的程序执行信息传递到死路径结束后的活动上.不可达路径是指不同分支条件的组合,使某些活动永远得不到执行.通过上面的分析可知,除了传统意义上的不可达路径,死路径对静态缺陷检测精度也有着不可忽视的影响.本文针对 BPEL 的特殊性提出了一种具有 DPE 语义的路径敏感静态缺陷分析方法,该方法同时解决了影响 BPEL 缺陷检测精度的两个因素——死路径和不可达路径,减少了后续分析中的误报和漏报.

2.2 BPEL缺陷描述

软件缺陷是指对软件产品预期属性的偏离现象,发现并排除软件缺陷需要大量的人力和时间花费.基于缺陷的软件测试在国外早已成为一种潮流,并已蔓延到国内.该方法可以解决白盒测试和黑盒测试所面临的不确定性问题,具有针对性强、对小概率事件检测精度高、测试效率高、缺陷定位准确等优点.

BPEL 规范中给出了 95 种需要静态检测的缺陷,因此有必要对 BPEL 建立一种基于缺陷的检测方法.根据经验,表 1 给出了已经总结出的缺陷描述.当然,更多的缺陷还需要在实践中进一步总结.

Table 1 Description of defects

表 1 缺陷描述

缺陷名	缺陷描述
未初始化变量(UIV)	在使用一个消息变量整体或部分(如 WSDL 消息类型变量的某部分)时还没有被赋值.
变量子属性定位失败(VLF)	在 assign 活动或函数中使用查询操作定位变量的某属性时找不到相应的节点,该缺陷主要针对复杂类型的消息变量.
死锁(DL)	并发执行的活动循环等待或由于某些原因导致流程无法继续执行.BPEL 通过 link 结构指定并发活动之间的控制依赖,如果 link 之间形成控制依赖环则发生了死锁;或一个流程等待另外一个流程的消息,但该消息永远不会到达.
死循环(DC)	循环条件不当,导致循环执行的活动无法结束;BPEL 通过 while,repeatUntil 实现循环,则当遗漏循环终止条件或循环体与循环终止条件无关时,会导致死循环.
数据竞争(DR)	并发执行的两个或多个活动之间至少存在一个活动对同一变量的同一部分进行写操作(此缺陷需要对消息变量进行细粒度划分,即变量的具体查询操作).
Foreach 循环完成条件冲突(RCC)	Foreach 活动中的 completionCondition 属性值大于循环本身执行条件的限制值即 completionCondition>finalCounterValue-startCounterValue+1(具体含义参见 BPEL2.0 规范).
Link 非法使用(IUL)	Link 连接的源活动和目标活动分别位于同一分支活动的不同分支上,而分支活动不可能同时执行,使 link 连接的目标活动始终得不到执行.
路径不可达(PUR)	例如 if 条件永远为 true,导致 else 分支永远得不到执行.
元素属性滥用(IUA)	元素属性是描述活动行为的重要依据,如果属性设置非法,则会导致重要的流程行为异常,甚至是流程异常中断.例如在具有多个开始活动的流程中,必须有一个相同的关联集合属性值,并且所有关联集合的 initiate 属性设置为 join.否则将会导致流程执行中断.
分支活动相同(BAS)	由于设计失误,导致流程的不同分支执行相同的活动,使流程行为难于理解.
并发活动不可达(CAUR)	由于汇合条件 *jc* 的影响,使活动永远得不到执行.
消息接收冲突(MCR)	存在两个或两个以上的活动等待接收同一消息.即两个或两个以上的初始化活动(receive,pick 和 event handler)具有相同的伙伴链接(partner link)、端口类型(port type)、操作(operation)以及关联集合(correlation set)属性.
资源泄露(RL)	BPEL 引擎将接收到的消息保存在一个消息队列中,直到相应的流程实例结束,才释放所有保留的消息资源,但由于 BPEL 支持长时间运行的业务流程,过多的输入消息会导致内存不足(即不存在 receive,pick 或 event handler 与相应的输入消息匹配).

Table 1 Description of defects (continue)

表 1 缺陷描述(续)

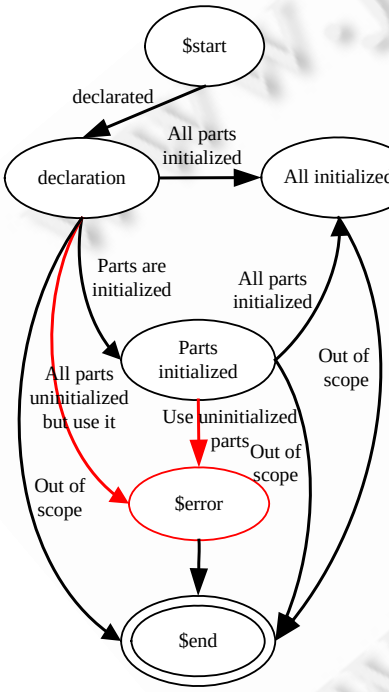
消息不兼容(MUC)	流程与被调用服务的接口的不相容,可以体现在以下 6 个方面: (a) 提供者的消息中包含接收者需要的多余消息; (b) 提供者提供的消息中缺少接收者需要的某些消息; (c) 提供者分多次发送接收者需要一次整体接收的消息的不同部分; (d) 提供者一次发送接收者希望多次分批接收的不同消息; (e) 请求者希望无论什么情况都能够收到接收者的返回消息,但提供者只有在特定的条件下才返回消息; (f) 请求者希望只有在一定条件下才接收接收者的返回信息,但提供者没有进行任何条件限定.
关联集合消息不匹配(CMUM)	BPEL 运行过程中,可能同时初始化多个流程实例,关联集合属性用来匹配返回的消息实例,如果消息与设定的关联集合不匹配,则导致该消息无法被任何流程实例接收.
伙伴服务不存在(PSUE)	调用一个不存在的伙伴服务.

2.3 缺陷建模及BPEL控制流图

时序安全属性(temporal safety properties)描述“某些坏事情不会发生”的属性,可以采用状态机进行建模.

定义 1. 缺陷状态机 EFSM 包含一个状态集合 D、状态转换集合 T 和迁移条件集合 $Conditions$.其中,$D=\{\$start,\$error,\$end\}\cup others$,$T:D\times Conditions\rightarrow D$.$\$start$,$\$error$ 和$\$end$ 分别代表初始、错误和结束状态,$others$ 是可能的其他状态.图 3 是 BPEL 未初始化变量缺陷状态表示,省略了相应的 XML 定义.

该状态机共包含 6 个状态(椭圆形表示),并给出了相应状态之间可能存在的状态转换(连接椭圆形的箭头).在不同的条件下,同一状态可能变迁到不同的下一状态,如 declaration 状态.图形右面给出了相应的状态转换描述.可见,如果消息变量未被赋值或仅部分赋值,都存在未初始化变量缺陷的风险,而这很难通过人工或动态测试方法发现.通过沿着该状态机状态变迁对源代码进行检测,便可以检测出流程中隐藏的此类缺陷.



序号	状态转换	引起状态变迁的动作
0	start→declaration	变量声明
1	declaration→All initialized	变量被全部初始化
2	declaration→Parts initialized	变量被部分初始化
3	parts initialized→All initialized	变量被全部初始化
4	declaration→error	对声明变量进行引用
5	declaration→end	超出作用域
6	parts initialized→end	超出作用域
7	All initialized →end	超出作用域
8	parts initialized→Error	引用未赋值的变量

Fig.3 EFSM of uninitialized variable

图 3 未初始化变量缺陷状态机

受 DPE 的影响,在对状态机属性状态进行分析时可能出现 3 种情况,即(possible,disabled,invalid):possible 代表当前的属性状态是有效地;disabled 代表活动或 link 进入死路径前状态机的属性状态,死路径上的状态更改

对 disabled 状态没有任何影响,当死路径结束后,该 disabled 状态可以变为 possible,继续对后续分析起作用;而 invalid 代表该属性状态已成功变迁到另一可能状态.为了对死路径上的状态变迁进行跟踪,使用布尔变量 *mbd* 对死路径进行标识,从而判断某一活动或 link 是否处于死路径片段上(其中,true 代表死路径,false 代表正常路径).

影响 *mbd* 的关键因素是 link 变迁条件上变量的可能取值,而变量依赖于特定应用相关的消息,由外部客户端的输入或伙伴服务的返回消息决定,通常其值域是无限的或者非常大,要在静态分析过程中取到值域内的每一个可能值是行不通的.采用了变量抽象取值范围即区间来表示可能的变量值,通过区间代数计算 *mbd* 的值.如图 4 所示的计算折扣的流程实例,虚线代表活动之间的同步依赖关系,if 活动的 true 分支使 *sum* 的区间为[0,999],而当计算活动 *C* 的执行条件时,首先计算 link 的状态,由于 link 上的变迁条件为 *sum*>=10000,与当前的变量集合进行交运算的结果为空 *null*,link 的状态为 false,因此,其 *mbd* 的值为 true.而活动 *C* 只有一个输入 link,这样,活动 *C* 始终得不到执行(即不可达活动),导致当 *sum*<10000,流程执行活动 *D* 时,出现变量 *price* 未被赋值的问题.

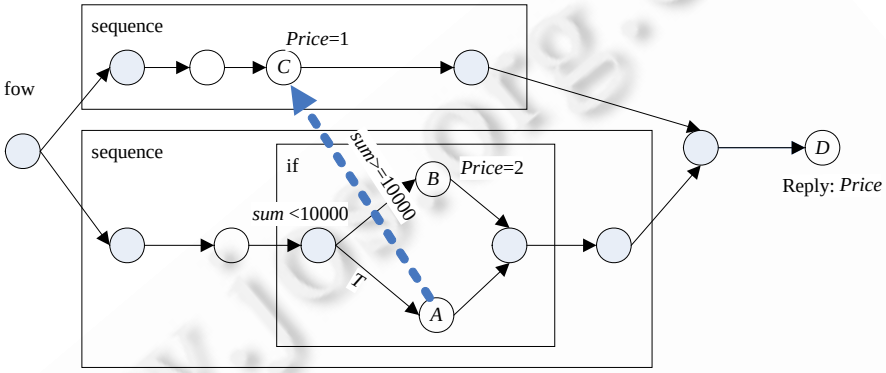


Fig.4 Segment of process

图 4 流程片段

另外,当遇到分支活动时,尤其是多个分支条件的组合,根据分支条件上变量的抽象取值区间范围为空,判断不可达路径(如果某一活动位于不可达路径上,则不需要计算).当遇到同步汇合节点(具有输入 link 的活动)时,可以根据输入 link 关联的 *tc* 中变量的抽象取值区间范围判定输入 link 的状态值,从而判断当前活动是否能够执行.如果该活动无法执行,则从该活动开始直到死路径结束这段路径上,属性状态的改变对后续分析没有任何影响,因此需要保留进入死路径前的状态信息(即 disabled 集合).

BPEL 作为一种基于 XML 的程序语言,直接对代码进行分析没有实际意义,需要对其进行抽象,建立一种中间模型,定义了能够表达并发和同步特性 BPEL 控制流图.

定义 2(BPEL 控制流图). 形式化地,BPEL 控制流图是一个四元组 $G=\langle A,E,s,f\rangle$,其中,A 是节点的集合代表活动,E 是边的集合代表活动之间的变迁,s 和 f 分别是流程开始节点和结束节点.其中,每个基本活动对应一个普通节点,结构化活动对应一个节点对(*head*,*tail*),分别表示结构化活动的头节点 *head* 和尾节点 *tail*.对于并发活动 *flow*,其 *head* 和 *tail* 相当于传统意义上的 *fork* 和 *join*.同时还定义了 3 种边(*ne*,*ce*,*se*),即普通控制流边 *ne*,并发边 *ce* 和同步边 *se*.其中,同步边 *se* 是一个三元组$\langle cn,nn,pr\rangle$,其中:*cn* 和 *nn* 分别代表变迁的当前节点和下一个节点;*pr* 是一个条件表达式,代表 link 上的条件变迁.直接嵌套在 *flow* 中的节点使用并发边 *ce* 与 *flow* 首尾节点对连接(用虚线表示).图 5 给出 BPEL 活动到 BPEL 控制流图的映射关系,分别包含了原子活动、结构化活动 if 的表示方法,最右面给出了银行贷款审批流程的 BPEL 控制流图.

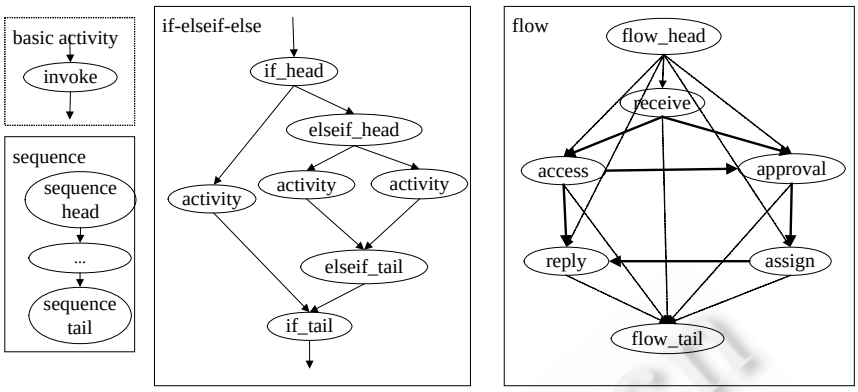


Fig.5 A map from segment to BPEL CFG

图 5 元素到 BPEL 控制流图的映射

3 死路径语义下路径敏感缺陷检测

3.1 检测框架

本文的焦点是关注 DPE 和不可达路径对缺陷检测精度的影响.整个缺陷检测过程可以描述为:采用 EFSM 建模 BPEL 缺陷,在控制流图的基础上抽象出 BPEL 的运行时语义(DPE),进而沿着状态机的状态变迁条件对 BPEL 源代码进行分析验证.

图 6 描述了整个缺陷检测框架,为提高检测精度,从路径抽象和近似的角度出发,在路径敏感性分析中增加了 DPE 语义来提高检测精度.

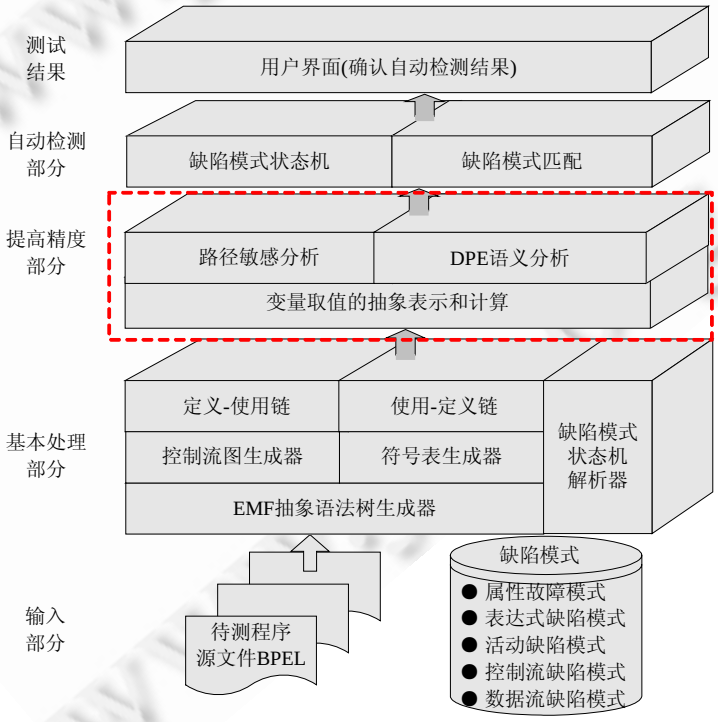


Fig.6 Framework of defect detecting

图 6 缺陷检测框架

路径敏感分析是静态缺陷检测分析过程中典型的提高精度的技术,可用于减少误报;DPE 语义分析是 BPEL 特有的提高检测精度的技术,可消除可达路径上未执行活动对分析结果的影响,对减少误报和漏报有很好的贡献.

3.2 抽象解释

整个缺陷分析过程采用正向数据流分析[19],将状态机看作数据流要分析的值.为此,需要计算到达每一个节点或 link 的属性状态集合以及经过每一节点或 link 后的属性状态集合.如果发现状态机进入 *error* 状态,则说明流程中存在相应缺陷.

受 DPE 的影响,每个缺陷状态机的属性状态对后续分析可能表现为不同的形式,即上面提到的可能(possible)状态、保留(disabled)状态和无效(invalid)状态.为了存储控制流图中每个节点上属性状态的不同形式,首先给出一些函数定义,A 和 L 分别代表活动和 link 的集合,fsm 代表状态机实例,D_{defect} 代表缺陷的集合.

函数 $poss_o$:$(A\cup L)\times D_{defect}\rightarrow \wp(fsm)$表示到达 x 时(x 可能是活动或 link),根据待检测缺陷产生的可能属性状态集合.函数 dis_o:$(A\cup L)\times D_{defect}\rightarrow \wp(fsm)$表示到达 x 时,根据待检测缺陷产生的当前保留属性状态集合.函数 inv_o:$(A\cup L)\times D_{defect}\rightarrow \wp(fsm)$表示到达 x 时,根据待检测缺陷产生的已正常变迁的属性状态集合.进行属性状态变迁计算时,需要判断当前活动或 link 是否位于死路径上.用布尔变量 mbd_o 表示到达 x 时的死路径信息,如果其值为 true,可能的属性状态进入保留状态;否则,其值为 false,状态成功迁移,进入无效状态.由于 inv_o 中的元素不会影响缺陷判断,因此实际计算时不需要保留其值. $poss_\bullet$,$dis_\bullet$,$inv_\bullet$ 和 $mbd_\bullet$ 分别对应 x 计算完毕后相应的属性状态集合及死路径标识.

定义 3(函数 fsm_o). $(A\cup L)\times D_{defect}\rightarrow poss_o\times dis_o\times \mathbb{B}$返回进入活动节点或 link 边时,所有属性状态元组集合,包括可能的属性状态、保留的属性状态和代表是否位于死路径的标识.$\mathbb{B}$($\mathbb{B}$是布尔集合{true,false}.

定义 4(函数 $fsm_\bullet$). $(A\cup L)\times D_{defect}\rightarrow poss_\bullet\times dis_\bullet\times \mathbb{B}$ 返回当前活动或 link 边处理完后,所有属性状态元组集合.

3.3 检测算法

为便于计算,在生成控制流图的同时对节点进行了编号,算法的主要任务是计算进入和离开活动或 link 时 fsm_o 和 $fsm_\bullet$ 的值,如果 $fsm_\bullet$ 中某一状态为 *error*,则说明程序可能存在此种类型的缺陷.

将属性状态元组 fsm 看作数据流要分析的值,考虑到路径条件和 link 完整语义(包括变迁条件 tc 和汇合条件 jc),当遇到汇合节点或同步节点时,需要对这些状态元组进行合并操作,具体的合并策略取决于当前处理的节点类型.如果是普通的分支间合并,则仅需要对这些分支按照相同的属性状态进行合并;但是如果当前的节点是同步汇合节点或并发的尾节点,则需要特殊处理.

假设流程沿着路径 p 执行到活动 a,相应的属性状态机的状态沿着 p 进行传递和变迁,在 a 处到达状态元组集合 σ.路径 p 上的条件谓词和赋值操作对到达 a 处的变量取值范围进行了限定,记为 τ,则 τ 即为状态 σ 的路径条件.但 DPE 会使路径上某些活动的执行条件无法满足而得不到执行,记为 τ',则 τ' 记为活动的执行条件.路径条件和执行条件共同决定了控制流图上某节点处的属性状态值.路径条件 τ 用来识别不可达路径,以减少误报,而执行条件 τ' 用来识别死路径,从而可以有效保留进入死路径前的状态信息.这里的 $\tau'(\tau)$ 都是一个累计值,它由沿着路径 p 分析活动执行(可达活动)需要满足条件的布尔表达式构成.例如图 4 中 C 的执行条件用区间表示为 [0,999]&&[10000,Max*Int*],其中,*MaxInt* 是系统能表示的最大整数值,C 的路径条件没有限制.

为了明确对控制流中 link 边 se 的访问,用对象 x 表示,d 代表当前检测的缺陷.首先计算到达每一个对象前的属性状态集合 fsm_o,其中:p 是 x 的顺序逻辑前驱;$tail(flow)$表示 $flow$ 的尾节点;l 表示一个同步 link;$s,t:(s,x,t)\in LR$ 表示 x 是一个 link,它关联一个源活动 s 和一个目标活动 t.如果 x 是并发活动的尾节点 $tail(flow)$,则仅合并那些没有输出 link 的前驱活动(用 $leaves[x]$表示)的属性状态信息,如伪算法 1.

算法 1. 进入每个对象前的状态元组集合.

procedure *calculateIN*(x)

begin

if x is the root process

$$fsm_{\circ}(x,d)=(\varnothing,\varnothing,\text{false})$$

if x is instance of activity

$$fsm_{\circ}(x,d)=\begin{cases}\bigcup\limits_{p\in pred[x]} fsm_{\bullet}(p,d)\cup joinLinks(a,d), & x\notin tail(flow)\\ \bigcap\limits_{k\in leaves[x]} fsm_{\bullet}(k,d), & x\in tail(flow)\end{cases}$$

if x is instance of link

$$fsm_{\circ}(x,d)=fsm_{\bullet}(s,d)\ \ \exists s,t:(s,x,t)\in LR$$

end

通过 *joinLinks* 对当前活动的输入 link 信息进行合并,$L_{in}(a)$表示活动 a 的输入 link,如伪算法 2.

算法 2. 合并具有输入 link 前驱信息.

procudure *joinLinks*(a,d)

begin

$$joinLinks(a,d)=\begin{cases}\bigcap\limits_{l\in L_{in}(a)} fsm_{\bullet}(l,d), & |L_{in}(a)|=1 \text{ or } jc=\text{AND}\\ \left(\bigcup\limits_{l\in L_{in}(a)} poss_{\bullet}(l,d)\cup dis_{\bullet}(l,d)\setminus \bigcap\limits_{l\in L_{in}(a)} dis_{\bullet}(l,d),\ \bigcap\limits_{l\in L_{in}(a)} dis_{\bullet}(l,d), mbd_{\circ}(a,d)\right), & \text{otherwise}\end{cases}$$

end

至此得到了进入对象前状态机实例的所有可能属性状态信息 fsm_o.根据该属性信息对当前对象进行属性状态变迁的检测,即计算 $fsm_{\bullet}$ 的值.其中,*gen* 代表的是经过状态变迁计算新产生的属性状态,即经过节点 x 后的可能的属性状态 $poss_{\bullet}(x,d)$.$mbd_{\circ}$ 表示 x 是否会引发 DPE 的产生或位于 DPE 上,如果其为 true,则说明当前的对象 x 位于死路径上;否则的话,说明当前活动处于正常的控制流路径上.如果 link 变迁条件上变量的取值区间为空(或者 link 链接的源活动没有执行),则 link 的状态值为 false,否则为 true.当活动所有输入 link 都获得状态值以后,便可以计算 jc 的值,从而判断当前的活动是否可以执行,见算法 3.

算法 3. 属性状态变迁的检测.

procedure *calculateOut*(x)

begin

if possible state is changed by x

$$fsm_{\bullet}(x,d)=\begin{cases}(gen(x,d),dis_{\circ}(x.d)\cup poss_{\circ}(x.d),\text{false}), & mbd_{\circ}(x,d)=\text{true}\\ (gen(x,d),dis_{\circ}(x.d),\text{false}), & mbd_{\circ}(x,d)=\text{false}\end{cases}$$

else

$$fsm_{\bullet}(x,d)=fsm_{\circ}(x,d)$$

end

有了以上对象处理算法后,在进行数据流迭代计算时,在控制流汇合节点合并相同属性状态的执行条件(即区间),能够避免全路径敏感分析的组合路径爆炸问题.其中,*mergeStateCondition* 是对相同属性状态的条件进行合并,*updateStateCondition* 是根据当前对象上的条件进行状态更新.具体的数据流迭代算法见算法 4.

算法 4. 缺陷检测数据流迭代算法.

for each $o\in A\cup L$, do $in[o]:=\varnothing$;

change:=true;

while *change* do begin

if o=*process* then

```
        in[o]={∅,∅,false}
    else
        in[o]=call calculateIn(o);
    if o∈merge then
        in[o]=mergeStateCondition(in[o]);
        oldout=out[o];
        out[o]=call calculateOut(o);
        updatadateStateCondition(out[o]);
    if out[o]≠oldout then change=true;
    end
end
```

死路径语义下路径敏感的静态缺陷检测算法在多项式时间复杂度的路径敏感分析算法中增加了 DPE 语义计算,因此也增加了算法复杂度.下面对 DPE 计算复杂度进行分析,主要包括死路径标识 *mbd* 的计算和 possible 和 disabled 集合的计算:*mbd* 的计算与流程中变量相关,变量抽象语义上的布尔操作的复杂度通常线性的; possible 和 disabled 则相对复杂.最坏情况下,flow 包含的所有活动都顺序执行,并且每个 link 都关联一个 *tc*,这样,对每一个活动进行计算之前都先要计算 link 上的状态集合,并将其拷贝到目标活动中.假设流程中包含 *n* 个活动,则复杂度为 $O(n^2)$.

3.4 对比分析

实验中采用了故障注入的方式进行测试.为了具体说明 DPE 是如何影响路径敏感测试结果的准确度,表 2 给出了图 2 流程片段具体的分析过程,分别对路径敏感和带有 DPE 的路径敏感进行了分析.根据给定算法的计算可见,当对 if 节点进行计算时,两者出现了不同的结果.

Table 2 Result of compare analysis

表 2 对比分析结果

节点名称		a_1	w_x	a_2	if	R_x	a_3	a_4
路径敏感分析	$fsm_\circ$	{*decl*, --, --}	{*decl*,--,--}	{*decl*, --,--}	{ *decl*,*init*), --,--}	{(*ecl*:[true], *i* *it*:[true]), --,--}	{(*decl*:[false], *init*:[false]), --,--}	{(*decl*:[false], *init*,*error*:[true]), --,--}
	$fsm_\bullet$	{*decl*, --,--}	{*init*:[true], --,--}	{*decl*, --,--}	{ *decl*,*init*), --,--}	{(*ror*:[true], *i* *it*:[true]), --,--}	{(*decl*:[false], *init*:[false]), --,--}	{(*decl*:[false], *init*,*error*:[true]), --,--}
DPE 语义下的路径敏感分析	$fsm_\circ$	{*decl*, --,false}	{*decl*,--,true}	{*decl*,--, false}	{(*ecl*:[false], *i* *it*:[true]), -,false}	{ *nit*:[true] - ,--,false}	{*decl*:[false], --,false}	{(*decl*:[false], *init*:[true]), --,false}
	$fsm_\bullet$	{*decl*, --,false}	{*init*:[true],*decl*: [false],false}	{*decl*,--, false}	{(*ecl*:[false], *i* *it*:[true]), -,false}	{ *nit*:[true], -,false}	{*decl*:[false], --,false}	{(*decl*:[false], *init*:[true]), --,false}
结论		带有 DPE 语义路径敏感分析能够进一步提高传统路径敏感分析的检测精度						

4 相关工作

有关 BPEL 的验证和测试,已经出现了各种各样的技术.Petri 网作为一种强有力的形式化方法,是基于工作流验证领域的首选静态分析方法,涌现出大量采用 Petri 验证组合 Web 服务的方法,通过将 BPEL 转化成 Petri 网,然后应用现有的分析技术进行静态检测,如自动化工具 WofBPEL 采用 Petri 技术对 BPEL 控制流进行静态分析,能够检测不可达活动、冲突消息接收和垃圾消息(即永远不被使用的消息)[20].但由于状态空间的限制,该方法很难应用于大规模的流程验证中,而且该方法没有考虑变量和条件对检测结果的影响.

在软件测试中,大部分的程序验证都源于程序的数据流分析,如可达性分析、定义/使用分析、数据竞争分析等.但传统的数据流分析算法并不能直接应用在 BPEL 这种基于消息的程序设计语言上,这是因为传统的数

据流分析算法不能直接分析出消息的定义和使用.不精确的数据流分析会导致检测结果中包含大量的误报和漏报,鉴于 BPEL 语言的特殊性,需要对其进行特殊数据流分析.董文莉提出了一种基于 BPEL 的 Web service 组合数据流分析方法 WSCTM,从 3 个视点分析数据流,即服务间、服务内部和服务实现构件间[21],使 Web service 组合的数据流测试可以在 3 个层面上得到实现.但该方法没有考虑 BPEL 的 DPE 语义对定义使用边的影响,使获得的定义/使用链中包含了大量的冗余边.为解决这种情况,减少冗余的定义使用边,Oliver 在数据流分析的同时考虑了 DPE 对写活动(能够对变量进行赋值的活动)的影响,采用知识推导的方法判断可能的死路径,很大程度上减少了冗余的写活动,但并没有去掉那些因分支间组合而产生的不可达路径上的写活动[22,23].本文提出的数据流迭代算法重筹兼顾,同时考虑了 DPE 和路径条件之间的关系.

表 3 通过对比的方式说明 DPE 语义下路径敏感的缺陷检测方法的好处(ASM:抽象状态机;FSM:有限状态机;PA:进程代数;PN:Petri 网.√代表支持,×不支持,两者都有代表只部分支持).

Table 3 The scopes of different models

表 3 不同模型适用范围

相关工作	技术支持	结构化活动	控制 link(DPE)	路径敏感	检测缺陷的完整性	流程间
Farahbod[24]	ASM	√	√/×	×	√/×	×
Ferrara[2]	PA	√	×	×	√/×	×
Fu[8]	FSM	√	√	×	√/×	√
C.Ouyang[20]	PN	√	√	×	√/×	√
Our work	EFSM	√	√	√	√	√

本文工作与以往工作的不同之处在于:采用状态机针对缺陷进行建模、分析和验证,一般而言,缺陷的状态不会超过 10 个;而其他采用状态机理论的验证方法都是对流程执行的可能状态进行建模,当流程较大时,会产生状态爆炸.实验室自主研发的基于缺陷模型的测试系统 DTS[25],并已应用到航空、航天、电力、电信和武装设备等领域的软件验证,取得了较好的效果.在该思想的基础上,本文提出的基于 BPEL 缺陷的通用检测方法采用数据流迭代算法对 BPEL 流程中的缺陷进行分析.通过上面的比较可以看出,我们的方法不仅检测的缺陷更加全面,而且由于在数据流迭代过程中考虑了 DPE 和路径信息,因此,与其他方法相比检测精度更高.

Kopp 采用的是知识推倒的方法计算死路径标识;Nakajima 采用谓词抽象的思想[26],对每一个条件变量引入辅助谓词变量(auxiliary predicate variables)来识别可能的死路径.本文引入了抽象解释思想,采用变量的抽象取值,即区间描述属性状态条件,应用区间代数识别死路径和不可达路径,避免了死路径和不可达路径上缺陷状态对后续分析的影响,有效提高了检测精度.经过缺陷分析的流程能够极大地改善性能,暴露隐含缺陷.

5 结束语

本文提出一种死路径语义下路径敏感的 BPEL 缺陷检测方法,将进入死路径前可能的缺陷状态转化为保留的;当死路径结束后,保留状态继续对后续分析有效,这样能够有效缓解死路径对检测精度的影响.为避免了全路径敏感分析的状态爆炸隐患,从路径抽象和近似的角度进行路径敏感性分析,采用变量的抽象取值计算识别不可达路径和死路径,在汇合节点合并相同属性状态的条件信息,将路径信息与死路径统筹考虑.最后,对路径敏感和带有 DPE 语义的路径敏感分析过程进行了对比,说明将 DPE 加入到路径信息中确实能够进一步提高检测精度.

将来的工作主要有两个方面:一是寻求一些技术,减少缺陷无关的条件信息,进一步提高检测效率;另一方面考虑到流程之间的通信,进行流程间上下文敏感的缺陷分析.

References:

[1] Web services business process execution language Version 2.0. 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.htm

[2] Ferrara A. Webservices: A process algebra approach. In: Proc. of the 2nd Int'l Conf. on Service Oriented Computing. New York: ACM Press, 2004. 242−251. [doi: 10.1145/1035167.1035202]

[3] Monakova G, Kopp O, Leymann F. Improving control flow verification in a business process using an extended Petri net. In: Proc. of the 1st Central-European Workshop on Services and their Composition (ZEUS 2009). 2009. 95−101.

[4] Sun P, Jiang CJ. Analysis of workflow dynamic changes based on Petri net. Information and Software Technology, 2009,51(2): 284−292. [doi: 10.1016/j.infsof.2008.02.004]

[5] Mukherjee A, Tari Z, Bertok P. Modeling of BPEL composite services using clustered coloured Petri-nets. In: Proc. of the World Conf. on Services—II. 2009. 55−62. [doi: 10.1109/SERVICES-2.2009.18]

[6] Fisteus JA, Fernández LS, Kloos CD. Formal verification of BPEL4WS business collaborations. In: Proc. of the 5th Int'l Conf. on Electronic Commerce and Web Technologies (EC-Web 2004), Vol.80. 2004. 76−85. [doi: 10.1007/978-3-540-30077-9_8]

[7] Foster H, Uchitel S, Magee J, Kramer J. Model-Based verification of Web service compositions. In: Proc. of the 18th IEEE Int'l Conf. on Automated Software Engineering. 2003. 152−161. [doi: 10.1109/ASE.2003.1240303]

[8] Fu X, Bultan T, Su JW. Analysis of interacting BPEL Web services. In: Proc. of the 13th Int'l Conf. on World Wide Web. ACM Press, 2004. 621−630. [doi: 10.1145/988672.988756]

[9] Lei LH, Duan ZH. An extended deterministic finite automata based method for the verification of composite Web services. Journal of Software, 2007,18(12):2980−2990 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2980.htm [doi: 10.1360/jos182980]

[10] Wombacher A, Fankhauser P, Mahlek OB, Neuhold E. Matchmaking for business processes based on choreographies. In: Proc. of the 2004 IEEE Int'l Conf. on E-Technology, E-Commerce and E2 Service. Washington: IEEE Computer Society, 2004. 359−368.

[11] Yang L, Liu X, Wang LZ, Chen X, Li XD. Scenario-Based analysis and verification for Web services message flows. Chinese Journal of Computers, 2009,32(9):1759−1772 (in Chinese with English abstract).

[12] Zheng YY, Zhou J, Krause P. Analysis of BPEL data dependencies. In: Proc. of the 33rd EUROM ICRO Conf. on Software Engineering and Advanced Applications. Washington: IEEE Computer Society, 2007. 351−358. [doi: 10.1109/EUROMICRO.2007.17]

[13] Khalaf R, Kopp O, Leymann F. Maintaining data dependencies across BPEL process fragments. Int'l Journal of Cooperative Information Systems, 2008,17(3):259−282. [doi: 10.1142/S0218843008001828]

[14] Yang XH, Huang JF, Gong YZ, Liu CC. Research on the static defect detecting in BPEL. Journal of Beijing University of Posts and Telecommunications, 2011,34(2):1−4 (in Chinese with English abstract).

[15] van Breygel F, Koshkina M. Dead-Path-Elimination in BPEL4WS. In: Proc. of the 15th Int'l Conf. on Application of Concurrency to System Design (ACSD'05). 2005. 192−201. [doi: 10.1109/ACSD.2005.11].

[16] Yang ZH, Gong YZ, Xiao Q, Wang YW. The application of interval computation in software testing based on defect pattern. Journal of Computer-aided Design & Computer Graphic, 2008,20(12):1630−1635 (in Chinese with English abstract).

[17] Xiao Q, Gong YZ, Yang ZH, Jin DH, Wang YW. Path sensitive static defect detecting method. Journal of Software, 2010,21(2):209−217 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3872.htm [doi: 10.3724/SP.J.1001.2010.03872]

[18] Das M, Lerner S, Seigle M. ESP: Path-Sensitive program verification in polynomial time. In: Knoop J, Hendren LJ, eds. Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 2002. 57−68. [doi: 10.1145/512529.512538]

[19] Aho AV, Lam MS, Sethi R, Ullman JD. Compilers Principles, Techniques and Tools. 2nd ed., New York: Addison-Wesley, 2006. 626−632.

[20] Ouyang C, Verbeek E, van der Aalst WMP, Breutel S, Dumas M, ter Hofstede AHM. Formal semantics and analysis of control flow in WS-BPE. In: Proc. of the Science of Computer Programming. Elsevier, 2007. 162−198. [doi: 10.1016/j.scico.2007.03.002]

[21] Dong WL, Hu JH. Test method for BEPL-based Web service composition based on data flow analysis. Journal of Software, 2009, 20(8):2102−2112 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/568.htm [doi: 10.3724/SP.J.1001.2009.00568]

[22] Kopp O, Khalaf R, Leymann F. Reaching definitions analysis respecting dead path elimination semantics in BPEL processes. Technical Report, University Stuttgart, IAAS, 2007.

[23] Kopp O, Khalaf R, Leymann F. Deriving explicit data links in WS-BPEL processes. In: Proc. of the IEEE SCC 2008. 2008. 367−376. [doi: 10.1109/SCC.2008.122]

[24] Farahbod F, Glasser U, Vajihollahi M. Abstract operational semantics of the business process execution language for Web services. Technical Report, SFU-CMPT-TR-2004-03, Burnaby: School of Computer Science, Simon Fraser University, 2004.

[25] Yang ZH, Gong YZ, Xiao Q, Wang YW. A defect model based testing system. Journal of Beijing University of Posts and Telecommunications, 2008,31(5):1−4 (in Chinese with English abstract).

[26] Nakajima S. Model-Checking behavioral specification of BPEL applications. Electronic Notes in Theoretical Computer Science, 2006,151(2):89−105. [doi: 10.1016/j.entcs.2005.07.038]

附中文参考文献:

[9] 雷丽晖,段振华.一种基于扩展有限自动机验证组合 Web 服务的方法.软件学报,2007,18(12):2980−2990. http://www.jos.org.cn/1000-9825/18/2980.htm [doi: 10.1360/jos182980]

[11] 杨璐,柳溪,王林章,陈鑫,李宣东.面向基于场景规约的 Web 服务消息流分析与验证.计算机学报,2009,32(9):1759−1772.

[14] 杨学红,黄俊飞,宫云战,刘传昌.BPEL 静态缺陷检测方法.北京邮电大学学报,2011,34(2):108−112.

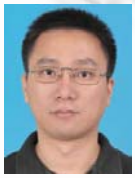
[16] 杨朝红,宫云战,肖庆,王雅文.基于缺陷模式的软件测试中的区间运算应用.计算机辅助设计与图形学报,2009,20(12):1630−1635.

[17] 肖庆,宫云战,杨朝红,金大海,王雅文.一种路径敏感的静态缺陷检测方法.软件学报,2010,21(2):209−217. http://www.jos.org.cn/1000-9825/3872.htm [doi: 10.3724/SP.J.1001.2010.03872]

[21] 董文莉,胡建华.基于 BPEL 的 Web Service 组合的数据流分析测试方法.软件学报, 2009,20(8):2102−2112. http://www.jos.org.cn/1000-9825/568.htm [doi: 10.3724/SP.J.1001.2009.00568]

[25] 杨朝红,宫云战,肖庆,王雅文.基于软件缺陷模型的测试系统.北京邮电大学学报,2008,31(5):1−4.

杨学红(1983－),女,河北唐山人,博士生,主要研究领域为软件测试,服务测试.



黄俊飞(1977－),男,博士,讲师,主要研究领域为软件测试.

宫云战(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试,软件工程.