文章编号: 1001-506X(2009)01-0241-05

BPEL 静态流程切片技术研究

鲍 亮, 宋胜利, 陈 胜, 陈 平, 胡圣明, 崔伟勇
(西安电子科技大学软件工程研究所, 陕西 西安 710071)

摘 要: 提出了业务流程切片的概念和对应的静态切片算法。流程切片是根据给定的活动和活动引用的变量(称为切片准则),分析出在整个流程中能够影响切片准则的流程片段。介绍了静态切片算法在给定切片准则的前提下,首先构造 BPEL 控制流图,在此基础上生成活动依赖图,并对活动依赖图进行分析,从而确定流程中影响切片准则的活动序列(流程片段)。其结果已经在流程分析、优化和并行化等方面起到了重要作用,效果较好。

关键词: BPEL; 流程切片; 流程片段; 静态切片算法
中图分类号: TP 393 文献标志码: A

Research on static process slicing in BPEL

BAO Liang, SONG Sheng-li, CHEN Sheng, CHEN Ping, HU Sheng-ming, CUI Wei-yong
(*Software Engineering Inst., Xidian Univ., Xi' an 710071, China*)

Abstract: The concept of process slicing in business process execution language (BPEL) and related static slicing algorithm are proposed. Given an activity and variables referenced by this activity (called slicing criterion), one can analyze and obtain the process segment which influences the slicing criterion during process slicing. Based on the slicing criterion, a BPEL control flow graph (BCFG) and related activity dependency graph (ADG) are constructed respectively, and then the sequence of activity (process segment) is identified through the analysis of the ADG. The result of such analysis has been proved valuable in process analysis, optimization and parallelization.

Keywords: business process execution language; process slicing; process segment; static slicing algorithm

0 引 言

随着 SOA (service-oriented architecture)[1] 技术的不断发展,面向服务的计算逐渐成为开放异构环境复杂分布应用的主流计算模型。业务流程执行语言 (business process execution language, BPEL)[2] 是 SOA 协议栈中的核心元素,通过协作多个 Web 服务,提供一种基于标准的,能够构建灵活业务流程的方法。与传统的程序设计语言类似,BPEL 具有典型的控制结构,包括顺序、分支和循环等。除此以外,BPEL 还使用 flow 和 link 表达并发和同步行为。

随着越来越多的业务流程采用 BPEL 建模,对 BPEL 流程的分析、验证与测试就显得十分重要。目前的相关研究主要集中在 BPEL 流程的验证和测试方面[3-4],对流程分析的研究工作则相对较少。针对这种现状,本文提出了将传统程序切片技术[5] 应用到 BPEL 流程中的思路:文章首先给出 BPEL 静态流程切片的定义,并将 BPEL 流程建模为 BPEL 控制流图(BPEL control flow diagram, BCFG)。在得到 BCFG 的基础上,计算并添加控制依赖边与数据依赖边,形成活动依赖图(activity dependency diagram, ADG)。在给定切片准则(指定活动与变量集合)的情况下,使用静态流程切片算法对 ADG 进行分析,最终可以得到在指定活动执行时,所有影响某个变量集合的活动集合。其分析结果已经在流程分析和优化等方面起到了重要的实用价值。

1 相关概念与定义

1.1 BPEL 流程切片

给定一个 BPEL 流程 P, P 中包含的活动集合为 A, 定义的变量集合为 V, 有以下定义:

定义 1 流程切片准则(process slicing criterion):是一个二元组$<a, v>$,其中 $a\in A$, $v\subseteq V$。该定义的含义是:给定变量集合 v,需要观察从流程开始到活动 a 处 v 中变量状

收稿日期: 2007-07-17; 修回日期: 2008-04-29。
基金项目:"十一五"国防预研项目资助课题(513060601)
作者简介: 鲍亮(1981-),男,博士研究生,主要研究领域为 SOA,面向对象技术和软件体系结构。E-mail: baoliang@mail.xidian.edu.cn

态变化情况。

定义 **2** 流程切片(process slicing):是 A 的一个子集,该集合只包含符合切片准则的所有活动,即包含从起始节点到 $a(a\in A)$之间所有对变量集合 v 中变量进行修改的活动集合。

1.2 BPEL 控制流图

为了能够对 BPEL 流程进行分析,必须对流程进行建模。文献[3—4]针对流程测试需求,分别定义了各自的分析模型。本文采用 BPEL 控制流图这种中间模型支持流程分析。该模型对控制流图(control flow graph, CFG)进行扩展,加入一些表示 BPEL 特定流程结构、并发和同步语意的描述。

定义 **3** BPEL 控制流图(BPEL control flow graph, BCFG)。

BCFG 是一个有向图 $G=<N,E,S,F>$,其中:

(1) N 是结点集合,$N=\{n_i\}$,$1<=i<=p$,p 是 BCFG 中的节点数量,其中 $n_i\in\{SN, AN, MN, DN, FN, JN, EN\}$,$SN, AN, MN, DN, FN, JN, EN$ 的含义在表 1 中详细描述;

(2) E 是有向边的集合,$E=\{e_j\}$,$1<=j<=q$,q 是 BCFG 中的边数量,$e_j=<a,b>$,$a,b\in N$ 且 $e_j\in\{CE, PE, SE\}$,CE 表示控制边,PE 表示并发边,SE 表示同步边,如表 1 所示;

(3) S 是流程开始节点的集合;

(4) F 是流程终止节点的集合。

表 1 BPEL 控制流图中的元素定义

名称	结点和边描述		
	元素缩写	定义	图例
开始节点	*SN*	流程开始节点集合,S0 表示整个流程的开始,每个 flow 活动内部开始节点从 1 开始依次编号	S
结束节点	*EN*	流程结束节点集合,E0 表示整个流程的结束,每个 flow 活动内部的结束节点从 1 开始依次编号	E
活动节点	*AN*	流程中的基本活动集合,只有一个入边和一个出边。(节点中可以注明活动名称和基本信息)	
消息节点	*MN*	pick 活动 OnMessage 元素,有一个入边和两个互斥的出边	M
判断节点	*DN*	条件判断,有一个入边和两个互斥的出边	D
Fork 节点	*FN*	flow 活动的开始,有一个入边和多个并发的出边	F
Join 节点	*JN*	flow 活动的结束,有多个并发的入边和一个出边。只有当所有的入边都到达时,出边才能开始	J
控制边	*CE*	流程中的控制流	→
并发边	*PE*	流程中的并行流	⇢
同步边	*SE*	flow 活动中的同步关系,活动按照先后顺序执行	⇢

对于一个具体的 *BPEL* 流程,整个流程的开始节点记为 *S*0,结束节点记为 *F*0,流程中可能有多个 *flow* 活动,每个 *flow* 活动的开始节点和结束节点从 1 开始依次编号。除此之外,*flow* 活动的开始和结束由一对 *Fork* 节点和 *Join* 节点表示;对于流程中 *pick* 活动的消息接收部分,采用消息节点表示;流程中的判断活动使用判断节点表示;其余活动一律用普通节点表示。*BCFG* 中有三类边:对于流程中顺序执行的控制流,采用控制边表示;*flow* 活动中引入的并发关系采用并发边表示;*flow* 活动中的 *link* 结构采用同步边表示。

1.3 活动依赖图

与多线程程序依赖图(*threaded program dependency graph*, *TPDG*)[6] 的定义类似,活动依赖图(*activity dependency graph*)是 *BPEL* 控制流图的一个变形,在保持 *BPEL* 控制流图中控制流边和并行流边的基础上,加入控制依赖边、数据依赖边、冲突依赖边和同步依赖边[7]。

定义 **4** 活动依赖图(*activity dependency graph*, *ADG*)。

ADG $G'=(N', E', S', F')$是 *BCFG* $G=(N, E, S, F)$的一个变形,结点和边满足如下关系:

$N'=N$, $S'=S$, $F'=F$, $E'=E\cup E_{dl}\cup E_{cd}\cup E_{id}\cup E_{sd}$

其中,E_{dl} 表示数据依赖边,E_{cd} 表示控制依赖边,E_{id} 表示冲突依赖边,E_{sd} 表示同步依赖边。

2 静态流程切片算法

在前面给出的相关定义和模型的基础上,图 1 给出了静态流程切片算法的主要步骤:

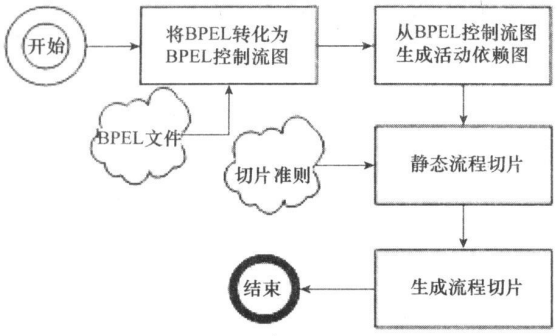

图 1　静态流程切片算法主要步骤

从图中可以看出, 静态流程切片算法的输入是一个合法的 *BPEL* 文件和对应的切片准则, 主要分为三个步骤: 第一步是将 *BPEL* 流程描述转化为 *BPEL* 控制流图; 第二步是从 *BPEL* 控制流图中生成活动依赖图; 第三步是输入切片准则, 进行静态流程切片, 最后生成符合切片标准的流程切片(影响切片准则中定义的变量集合的活动序列)。

2.1　将 BPEL 流程转换为 BCFG

算法的第一步要将 *BPEL* 流程转换为 *BPEL* 控制流图。映射过程包括基本活动(*receive*, *reply*, *invoke*, *assign*, *throw*, *terminate*, *wait*, *empty*)的映射和结构化活动(*sequence*, *flow*, *switch*, *while*, *pick* 等)的映射。具体转换规则如下所示:

规则 **1**　流程中的基本活动映射为 *BCFG* 中的活动节点 AN;

规则 **2**　流程中的结构活动映射为 *BCFG* 中的节点和边。根据不同的活动类型, 有以下的子规则:

规则 **2.1**　*switch* 和 *pick* 活动。将 *switch* 中的每个 *case* 部分映射为一个 DN 节点, 用控制边 CE 连接。*pick* 活动与 *switch* 类似, 只是将消息到达条件映射为对应的 MN 节点, 用控制边 CE 连接;

规则 **2.2**　*while* 活动。假设循环执行 0 次或 1 次, 将 *while* 活动的条件部分映射为一个 DN 节点, 用控制边 CE 连接;

规则 **2.3**　*flow* 活动。首先生成一个 FN/JN 节点对, 然后对于 *flow* 中的每一个分支活动, 生成一个 SN/EN 节点对, 分支内的活动包含在 SN/EN 之间, 最后从 FN 节点向每一个 SN 节点引出一条并发边, 每个分支的最后一个活动都向 JN 引出一条并发边;

规则 **2.4**　对 *link* 的处理。在静态分析时始终认为 *link* 的变迁条件为真, 这样 *link* 的语意就简化为同步关系, 直接从 *link* 的源活动端向目标活动端引出一条同步边。

2.2　将 BCFG 转换为 ADG

在得到 *BPEL* 控制流图 *G* 后, 算法的第二步需要将其转换为活动依赖图 G': 首先根据文献[8]中介绍的算法计算出 E_{cd} 和 E_{dd}, 然后根据文献[6]中介绍的算法, 计算出 *G* 中的冲突依赖边集 E_{id}, 再对于 *BCFG* 中的每个同步边, 生成对应的同步依赖边, 并且在 G' 中保留 *G* 中的控制流和并行流边, 加入 E_{cd}、E_{dd} 和 E_{id}, 最终生成 *ADG*。

2.3　根据切片准则进行静态流程切片

在介绍算法之前, 首先给出一些注记和辅助函数的定义[6]:

定义 **5**　*BPEL* 控制流图中的线程(通过 *flow* 活动创建)集合 $\Theta=\{\theta_0, \theta_1, \cdots, \theta_n\}$, 其中 θ_0 表示流程执行的主线程。对于图 3 描述的 *BCFG*, $\Theta=\{\theta_0, \theta_1, \theta_2\}$, 其中 θ_0 代表主线程, θ_1 代表执行 S1/E1 之间活动序列的线程, θ_2 代表执行 S2/E2 之间活动序列的线程。

定义 **6**　对于 *ADG* 中的每一个节点 n, 函数 $\theta(n)$, $n\in N'$ 返回节点 n 所在的最内层线程。例如, $\theta(A1)=\theta_0$, $\theta(A4)=\theta_1$, $\theta(A6)=\theta_2$。

定义 **7**　对于 *ADG* 中的每一个节点, 函数 $\Theta(n)$, $n\in N'$ 返回对于节点 n 来说, 不能并行执行的线程集合。例如 $\Theta(A2)=\{\theta_1, \theta_2\}$, $\Theta(A4)=\varnothing$。

定义 **8**　算法使用的状态变量 $C=(n, (t_0, \cdots, t_{|\Theta|}))$

$$t_i=\begin{cases} n, \theta(s)=\theta_i \\ \phi, \text{其它} \end{cases}$$

, 其中 $n\in N'$。

借鉴文献[4]介绍的算法, 在得到 *ADG* 后, 对于输入的程序切片准则 $s=<a, v>$ (令活动 a 在 *ADG* 上对应的节点为 n), 流程静态切片算法主要分为三个步骤:

(1) 从节点 n 开始, 收集所有通过控制依赖边和数据依赖边能够到达 n 的节点集合 S1, 并记录每个节点所在的线程 t;

(2) 从节点 n 开始, 收集所有通过冲突依赖边能够到达 n 的节点集合 S2, 如果 S1 中的节点 n1 能够通过控制流边或并行流到达 n, 则加入 n1, 否则认为 n1 对于 n 来说是并行不可达的节点, 不加入;

(3) $S=S1\cup S2$。考察同步依赖边集合 E_{sd} 中的所有边, 对于 $e=(n1, n2)\in E_{sd}$, 若, $n1\notin S$, $n2\in S$, 则剔除 S 中的节点 n2。

输入：切片准则 s(*ADG* 上的一个节点 n)
输出：流程切片 S(*ADG* 上的节点集合)
初始化状态变量 $C=(s,(t_0,\cdots,t_{|\Theta|}))\Big|t_i=\begin{cases}s,\theta(s)=\theta_i\\ \phi,\text{其它}\end{cases}$，并记 $T=(t_0,\cdots,t_{|\Theta|})$。
初始化工作链表 w={C}，程序切片 S={s}。
do
　从 w 中取出一个元素 c=(x, T)
　for each y:y 通过控制依赖边和数据依赖边能够到达 x
　　T′=[y/θ(y)] T //将对应的线程分量取值为 y
　　if(θ(y)≠θ(x))
　　　for each t∈Θ
　　　　T'=[φ/t] T'
　　c'=(y, T')
　　if c' 没有被标记
　　　对 c' 进行标记
　　　w=w∪{c'}
　　　S=S∪{y}
　for each y:y 通过冲突依赖边能够到达 x
　　t=T[θ(y)]
　　if t=φ *OR* y 能够通过控制流边或并行流边到达 t(t≠y)
　　　c'=(y, [y/θ(y)] T)
　　　if c' 没有被标记
　　　　对 c' 进行标记
　　　　w=w∪{c'}
　　　　S=S∪{y}
while w≠Φ
foreache=(x, y), $e\in E_{sd}$
　if(y∈S∧x∉S)
　　S=S−{y}

3 实例分析

为了能够更好地说明算法的执行过程，给出一个具体的 *BPEL* 流程实例(记为 *P*)，如图 3 所示。

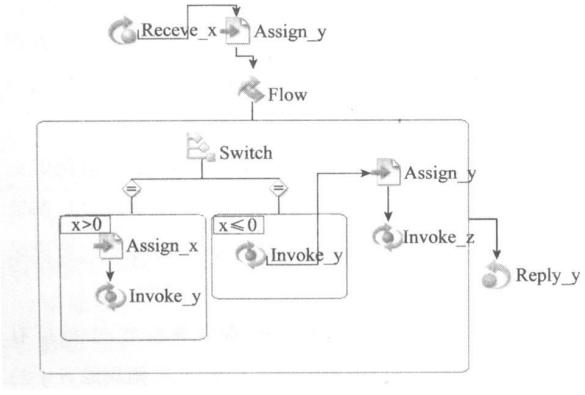

图 3 *BPEL* 流程实例 *P*

流程开始时首先接受一个变量 x，然后对变量 y 进行赋值，接着开始进行并发活动，其中一支首先是一个 *switch* 活动，判断 x 是否大于 0，如果大于 0，则对 x 进行赋值，然后以 y 为参数调用 *Web* 服务，如果 x 小于等于 0，则直接以 y 为参数调用 *Web* 服务。另外一支执行的前提是 *switch* 中 x 小于或等于 0 时对应的调用活动被执行，并且调用成功(由图中 *flow* 内部的 *link* 表示)，执行流程是对 y 赋值，并以 z 为参数调用 *Web* 服务，当所有的活动都执行完毕后，将变量 y 的值返回给流程调用者。

算法的第一步是根据流程描述生成 *BCFG*，首先加入整个流程的开始节点 *S*0，根据 2.1 节介绍的规则 1，将流程中的 *receive*, *assign*, *invoke*, *reply* 等基本活动映射为活动节点 AN(为了方便与图 3 的活动对应，保留活动名称)；根据规则 2.2，针对 *flow* 活动生成(*F*, *J*)节点对，对并发活动分支，依次构造起始和结束节点对(S1, E1)和(S2, E2)；对于 *switch* 活动，根据规则 2.1，加入一个判断节点 D，并将 *case* 子句对应的活动序列加入到 D 的出边上；根据规则 2.4，对流程中所有的 *link* 进行处理，加入对应的同步边；最后，加入流程的结束节点 E0。生成的 *BCFG* 如图 4 所示(记为 *G*)。

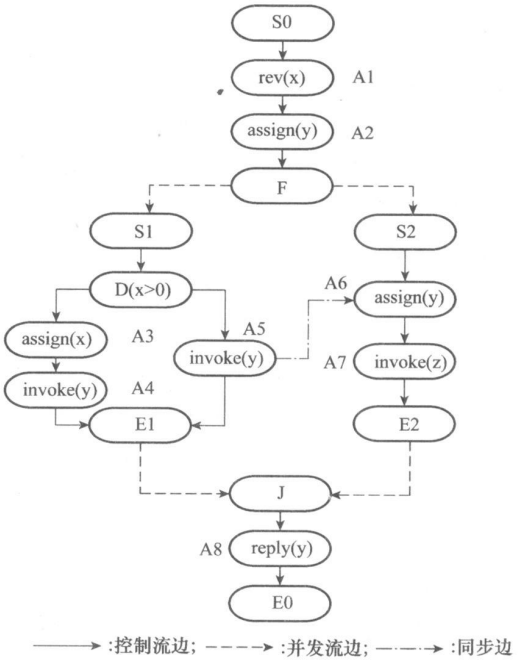

图 4 流程实例 *P* 对应的 *BPEL* 控制流图 *G*

算法的第二步是从 *G* 生成对应的活动依赖图 *G*′，文献[8－9]给出了得到控制依赖和数据依赖的算法：对于控制依赖，首先需要从控制流图中构造后向决定树(*post-dominator tree*)，然后通过对控制流图中的边和后向决定树上的相应节点进行分析即可得到控制依赖；对于数据依赖，可以采用构造有向无环图[8]的方式得到。在得到控制依赖和数据依赖关系后，利用文献[6]给出的算法计算出并发活动之间的冲突依赖关系，最后将 *G* 中的同步边转化为 *G*′中的同

步依赖关系，并保留控制流边和并行流边，最终生成的活动依赖图 G′如图 5 所示。

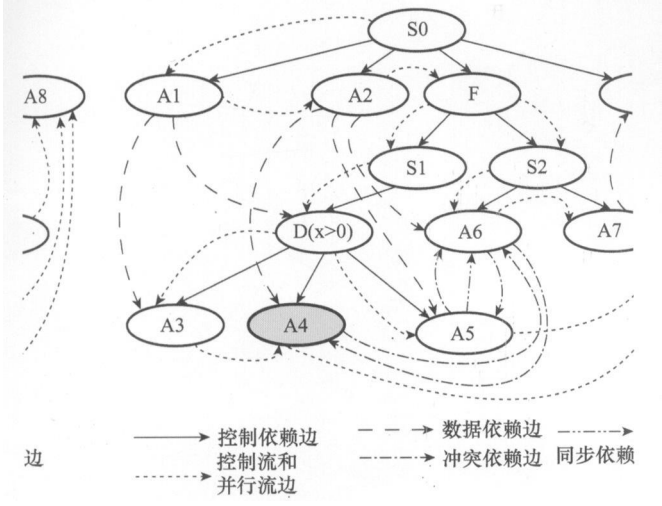

图 5　BPEL 控制流图 G 对应的活动依赖图 G′

算法的第三步是在得到 G′和给定切片准则的条件下进行静态切片。假设切片准则为＜*A*4，{*y*}＞，算法开始时初始化程序切片 *S*＝{*A*4}，通过数据依赖边和控制依赖边加入 *S* 的节点集合 *S*1＝{*D*，*A*2，*S*0，*A*1，*S*1，*S*2，*F*}，通过冲突依赖边加入 *S* 的节点集合 *S*2＝{*A*6}，注意虽然 *A*5 也能通过冲突依赖边到达 *A*6，但 *A*5 与 *A*4 在同一个线程中，且 *A*5 不能通过控制流边或并行流边到达 *A*4，因此 *S*2 中不能加入 *A*5。完成这些步骤后，*S*＝{*A*4，*D*，*A*2，*A*6，*S*2，*A*1，*S*0，*S*2，*F*}，由于 ADG 中同步依赖边集合 E_{sd}＝{＜*A*5，*A*6＞}，且 *A*5 ∉ *S*，*A*6∈*S*，需要在 *S* 中剔除节点 *A*6。最终满足切片标准的静态程序切片 *S*＝{*A*4，*D*，*A*2，*S*1，*A*1，*S*0，*S*2，*F*}，其中有效活动集合 *S*′＝{*A*4，*A*2，*A*1}，也就是说，对于活动 *A*4，在 *A*4 执行时，整个流程中只有 *A*2、*A*1 和 *A*4 自身能够影响变量 *y* 的取值。

4　结束语

Web 服务的组合体现了现代软件系统以业务为中心的组织模式，BPEL 目前已经是描述服务组合流程的工业标准语言，对 BPEL 流程的分析具有重要的实际意义。本文在分析借鉴现有程序切片方法的基础上，提出了 BPEL 流程切片的概念，并给出一个静态流程切片算法。在给定流程中的某个活动和一个关注的变量集合的情况下，算法能够得到整个流程中在执行该活动时所有能够影响集合中变量取值的活动集合，其结果能够用于流程优化和并行化等方面。

目前笔者及同事已经利用该算法对 BPEL 流程进行了死锁和数据竞争检测分析[10]，并对流程执行进行优化[11]，取得了较好的效果。后续的工作需要解决在流程规模较大的情况下，如何保证算法的效率，并结合约束求解等方法提高流程切片的精确度。

参考文献：

[1] Papazoglou M P，Georgakopoulos D. Service oriented computing [J]. *Communications of the ACM*，2003，46(10)：25－28.

[2] Business process execution language for Web service (BPEL4WS). http：//www.casisopen.org/committees/download.php/2046/BPEL%20V1－1%20Ma%y%205%202003%20Final.pdf.

[3] Yuan Yuan，Zhongjie Li，Wei Sun. A graph-search based approach to BPEL4WS test generation[C] //*Proceedings of the International Conference on Software Engineering Advance*，2006.

[4] Li Zhongjie，Sun Wei，Jiang Zhongbo，et al. BPEL4WS unit testing：framework and implementation[C] //*Proceedings of the IEEE International Conference on Web Services*，2005.

[5] Weiser M. Program slicing[C] //*International Conference on Software Engineering*，*San Diego*，*CA*，1981：439－449.

[6] Jens K. Static slicing of threaded programs[C] //*Workshop on Analysis for Software Tools and Engineering* (*PASTE*'98)，*Proc ACM SIGPLAN/SIGSOFT*，1998：35－43.

[7] Chen J. Slicing concurrent programs a graph-theoretical approach[C] //*Proceedings of the First International Workshop on Automated and Algorithmic Debugging*. *LNCS*，1999(749)：223－240.

[8] Ferrante J，Ottenstein K J，Warren J D. The program dependence graph and its use in optimization[J]. *ACM Transactions on Programming Languages and Systems*，1987，9(3).

[9] Kuck D J，Kuhn R H，Padua D A，et al. Dependence graphs and compiler optimizations[C] //*Conference Record of the Eighth ACM Symposium on Principles of Programming Languages*，1981：207－218.

[10] 陈胜，鲍亮，陈平，等. BPEL 流程数据竞争和死锁检测算法研究[J]. 西安电子科技大学学报(自然版)，2008，35(6)：1056－1062.

[11] Sheng Chen，Liang Bao，Ping Chen. OptBPEL：a tool for performance optimization of BPEL process[C] //*Proceedings of the 7th International Symposium on Software Composition*，*Springer Berlin/Heidelberg* 4954，2008：141－148.