文章编号：1001－9081(2008)S2－0129－03

一种 BPEL流程数据竞争检测方法

杨光超，陈 平，鲍 亮
(西安电子科技大学 软件工程研究所，西安 710071)
(chaoyg_fly@yahoo.cn)

摘 要：对使用业务流程执行语言(BPEL)组织流程中存在的数据竞争问题进行了研究，根据 BPEL特性提出了一个静态分析和动态监控相结合的有效检测方法。实验结果表明，该方法能够准确检测出流程中存在的数据竞争，提高了流程的可靠性。

关键词：业务流程执行语言；数据竞争；静态分析；动态监控
中图分类号：TP311 文献标志码：A

Data race detection approach for BPEL process

YANG Guang-chao CHEN Ping BAO Liang
(Software Engineering Institute Xidian University Xi'an Shaanxi 710071 China)

Abstract: As to the potential data race problem when composing business processes with Business Process Execution Language (BPEL), an efficient detection method was proposed according to the characteristics of BPEL, which combined static analysis and dynamic monitor techniques. Experimental results show that the proposed method can detect data race in business processes precisely, which also raises the reliability of business processes.

Key words: Business Process Execution Language (BPEL); data race; static analysis; dynamic monitor

0 引言

面向服务的架构(SOA)正在逐渐成为新的软件开发范型，其中软件功能以服务的形式通过网络发布，软件真正以面向接口的方式开发，复用层次也达到前所未有的高度。BPEL[1]作为SOA协议栈的主要组成成分，将网络上的服务组合以形成业务流程，并且流程可以通过网络发布为更大规模的服务。BPEL通过 flow和 link提供活动的并发和同步，因此 BPEL流程和传统的多线程程序一样面临着数据竞争问题。

目前的相关研究主要集中在 BPEL流程的验证和测试方面[2－3]，专门针对流程数据竞争检测的研究工作则相对较少。但在传统程序检测方面已有大量研究成果可供借鉴，例如 happens-before方式[4]根据程序执行偏序来检测竞争，可以适用于任何同步方式，而 lock-set[5]只能适用于基于锁的同步方式。

由于 BPEL的特殊语法(flow link)和语义(无效路径删除，"DPE")，对 BPEL的竞争检测需要采用不同技术。本文提出的检测方法首先对流程进行建模，结合图论来确定两个活动是否能够并发执行；然后根据静态分析的结果使用动态监控方法来检测流程中的数据竞争。该方法将静态分析和动态测试结合起来，既避免了传统静态分析中遇到的状态爆炸问题，又提高了检测效率和准确。

1 相关概念与定义

1.1 BPEL流程建模

为了确定两个活动之间能否并发执行，需要对流程进行建模。文献[6]针对流程的建模提出了 BAG(BEPL Segment Graph)模型。BAG是一个有向图，该模型与流程在结构上等价。本文的流程建模是在 BAG的基础上稍作修改而成。在静态分析之前为每个存在变量访问的活动以及存在变量访问的 link元素分配一个唯一 id来标识它们，id是一个由 1开始顺序递增的正整数。BAG图不包含任何变量信息以及迁移条件，只是包含活动以及活动之间的依赖关系。

定义 1 分支 b_n。把 BPEL流程运行时由同一个线程执行的活动序列称为分支。由同一线程执行的活动之间是不可能存在数据竞争的，通过给这些活动指定同一分支号可以避免不必要的检测，提高检测效率。

如果在 BAG中有一对节点 a和 a'，从 a到 a'不可达且 $a.b_n = a'.b_n$，则有 $a.id \| a'.id$。在这里可达指从 a到 a'要至少经过节点 a和 a'之外的另一个节点。

1.2 数据竞争

为了便于检测，我们对运行时变量访问定义了访问事件 event，运行中对每个变量访问都产生一个 event，运行结束后得到一个 event集合。最终通过对这个集合分析得到数据竞争检测结果。

定义 2 event。event是一个四元组(v，threadID，type，id)，其中：

v是访问的变量；

threadID是访问该变量的线程 ID；

type∈{R，W}，R表示对当前变量进行读操作，W表示对当前变量进行写操作；

id是访问该变量的当前活动或 link的 id。

对于两个访问事件 e_i，e_j定义：

$$IsRace(e_i, e_j) = (e_i.v == e_j.v) \wedge (e_i.threadID \neq e_j.threadID) \wedge (e_i.type == W \vee e_j.type == W) \wedge$$

收稿日期：2008－07－18；修回日期：2008－10－06。 基金项目："十一五"国防预研项目(513060601)。

作者简介：杨光超(1983－)，男，硕士研究生，主要研究方向：SOA、面向对象技术、软件体系结构；陈平(1953－)，男，教授，博士生导师，博士，主要研究方向：电子信息系统、软件开发、面向对象技术、软件工程、逆向工程；鲍亮(1981－)，男，博士研究生，主要研究方向：SOA、面向对象技术、软件体系结构。

$(e_i.id \| e_j.id)$ (1)

如果 IsRace(e_i, e_j)为真，则可以确定 e_i, e_j存在数据竞争。

2 BPEL流程竞争检测方法

BPEL流程静态分析部分，主要分为3个步骤：为BPEL活动分配分支号；将BPEL流程转化为BAG；活动并发性分析。

2.1 为BPEL活动分配分支号

为了提高检测速度，需要对分支编号。以下算法为每个活动分配分支号，某个分支的编号等于该分支中任意一个活动的编号。

```
输入：当前活动 act
输出：当前活动的分支号 n
int assign_bn(activity act, branch number n)
switch(activity type of act)
case basic activity:
  assign n to act's branch number          //为基本活动分配分支号
case switch, pick, while, onMessage:
  for each child activity(ca) in act
    assign n to ca's branch number
        // switch, pick, while, onMessage各子活动分支号相同
case sequence:
  for each child activity(ca) in act
    assign_bn(ca, n)
case flow, scope:
  p = 0
  for each child activity(ca) of act or onMessage branch of scope's
    eventHandlers
    p += assign_bn(ca, n+p)
```

从算法中可以得知基本活动和 switch, pick, while, onMessage的孩子活动分支号相同。由于执行 switch, pick, onMessage活动时每次只能选择一个满足条件的孩子活动来执行，这些孩子活动之间不可能出现数据竞争，指定相同的分支号可以避免这些活动之间的检测。该算法是一个基于图的递归遍历的过程，时间复杂度为 $O(n+e)$，n为图节点个数，e为图的边数。

设流程的顶层活动是 rootAct那么调用 assign_bn(rootAct, 1)可以为流程的所有活动分配分支号。

2.2 将BPEL流程转化为BAG

因为直接对BPEL源文件进行分析比较困难，因此需要将BPEL流程转化成等价结构BPEL活动图(BPEL Activity Graph, BAG)。BAG定义如下：

定义3 BAG，BAG是一个四元组$\langle N, E, s, f\rangle$，其中：

1) N是节点集合。$N=\{n_i\}$，$1\leqslant i\leqslant p$，p是BAG中节点个数。$n_i\in\{NN, SN\}$，NN, SN的含义见表1。

2) E是有向边的集合。$E=\{e_{ij}\}$，$1\leqslant i\leqslant q$，q是BAG中边的个数。$e_i=\langle a, b\rangle$，$a\in N$，$b\in N$。

3) s是开始节点，$s\in N$。

4) f是结束节点，$f\in N$。

以下规则用来将BPEL流程转化为BAG：

规则1 把while活动和存在变量访问的基本活动映射为NN，不存在变量访问的基本活动映射为SN。

规则2 活动的顺序执行映射为有向边。

规则3 如果存在一个 link L不存在变量访问，活动 a_1是L的源，活动 a_2是L的目标，那么插入一条从 a_1到 a_2的有向边。如果L存在对变量访问，则在 a_1和 a_2之间插入一个NN节点，并且插入一条从 a_1到NN节点的有向边和一条从NN到 a_2的有向边。

规则4 将 sequence展开，其中每一个子活动按规则1到6进行映射。

规则5 <switch><switch>映射为SN-SN，第一个SN节点到每个孩子活动映射一条有向边，所有孩子活动到第二个SN节点映射一条有向边。switch活动的每个孩子活动按规则1到6进行映射。flow, pick相同处理。

规则6 <scope><scope>映射为SN-SN，在第一个SN到子活动、事件处理器的每个 onMessage插入有向边，在子活动、事件处理器的每个 onMessage到第二个SN节点插入有向边。子活动和事件处理器的活动按规则1到6进行映射。

表1 BAG中节点的定义

名字	缩写	作用描述
普通节点	NN	有变量访问的节点
同步节点	SN	无变量访问的节点

2.3 活动并发性分析

这个步骤主要是结合图论判断BAG中两个活动是否能够并发执行。通过查找从一个节点到另一个节点的可达性和分支号来判定这两个节点所对应的活动是否并发。

给定一个 $p\times p$的矩阵 M，其中 p是BAG中NN节点个数，M_{ij}表示从 id为 i的活动 a_i到 id为 j的活动 a_j的可达性，如果 $M_{ij}=0$则 a_i到 a_j不可达。N'表示NN节点的 id集合，该集合可以通过静态分析得到；如果两个活动能够并发执行，则这两个活动的 id组成一个活动对，S_m就是活动对集合。并发性分析算法如下。

```
输入：BAG图开始节点 s
输出：并发活动对集合 Sm
初始化：初始化 Sm为空集
初始化矩阵 M，对角线元素置－1，其他元素置0
for each node ni∈N'
  for each node nj∈N'
    if从节点 ni到节点 nj可达
      if节点 ni和节点 nj是同一节点
        存在死锁，停止检测
      else Mij=1, Mji=-1
for each element Mst
      if Mst=0∧ as.bn≠at.bn为真
        则将(s,t)添加到 Sm中
```

在上述算法中，条件 $M_{st}=0\wedge a_s.bn\neq a_t.bn$表示 id分别为 s和 t的两个活动 a_s, a_t之间不可达且分支号不同，说明这两个活动是可以并发执行，则有 $s\| t$。经过并发性分析最终得到 S_m。在分配分支号算法中可以用一张二维表保存遍历过程中一个节点到另一个节点是否可达信息，本算法复杂度为 $O(n^2)$，其中 n为 N'元素数量。

2.4 动态监控

为了监控变量访问事件，本文采用一个修改过的ActiveBPEL引擎来执行流程。通过在每个活动的实现代码中植入Aspect代码来对变量访问进行监控并生成相应的event。例如在执行 invoke活动的开始先拦截该活动的 inputVariable变量和 outputVariable变量。根据BPEL规范可以确定变量的 type，通过 Thread.currentThread.getId()获取当前线程的标识 threadID，id则根据上下文来获取当前活动或

link的 id。每次对变量访问都会产生一个 event。当流程执行完以后结合静态分析结果 S_m和式(1)对 event集合进行分析可以得到数据竞争报告。

3 实例分析

图 1给出了一个简单网上购书订单处理流程示例。如图 1所示，在 flow活动中有两项任务并发执行。第一个任务是根据客户信息和订购金额计算该客户享受的折扣以及客户最终付款金额。另一个任务是根据订购金额计算赠送的代金券金额。在流程中，活动 InvokeDiscount的 joinCondition属性是默认设置，也就是如果 link的状态为真则活动 InvokeDiscount执行否则跳过。由图 1可见，活动 AssignDiscount和活动 InvokeVIP可以并发执行且都对变量 discount进行写操作，因此，这两个活动存在数据写-写竞争。另外，活动 AssignDiscount与 link存在对变量 discount写-读竞争，活动 InvokeFinalSum和活动 InvokeCoupons也存在对变量 sum读-写竞争。

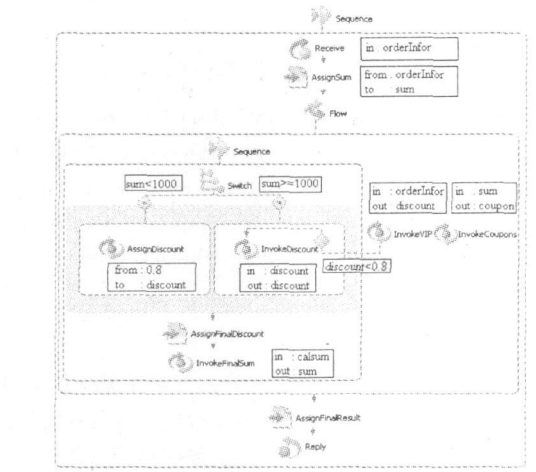

图 1 购书订单处理流程示例

图 2给出了示例流程转化之后的 BAG，由并发性分析算法可以确定如下活动对是可以并发执行的：AssignDiscount与 InvokeVIP，AssignDiscount与 InvokeCoupons，InvokeDiscount与 InvokeCoupons，AssginFinalDiscount与 InvokeVIP，AssginFinalDiscount与 InvokeCoupons，InvokeFinalSum与 InvokeVIP，AssginFinalDiscount与 InvokeCoupons。

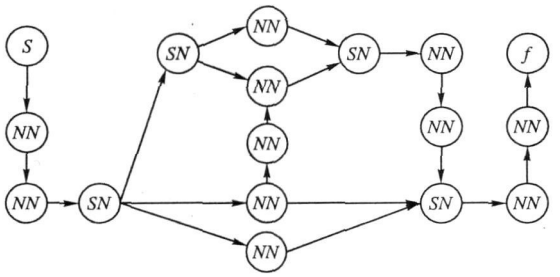

图 2 示例相关的 BAG

经过静态分析，可以知道两个活动是否可以并发。将 BPEL流程部署在修改过的 ActiveBPEL引擎中，给定测试用例，经过运行共产生 21个 event元素。结合静态分析结果和式(1)最终确定三对 event存在数据竞争，也就是上述提到的三个数据竞争被检测出来了。

为了进一步说明本文所述方法的有效性，本文从实际应用中选取了六个流程作为测试流程，分别是银行贷款流程(LA)、航空售票流程(AT)、工具集成流程(TI)、在线购书流程(OBP)、汽车装配流程(CA)、石油炼制流程(ORC)。由于外部服务对于竞争检测没有影响，为了加快执行速度，采用直接调用本地虚拟服务的方法执行流程。流程规模和测试结果信息如表 2，其中监控前时间用未修改版本 ActiveBPEL引擎执行流程所用的时间，监控后时间是指用修改版本 ActiveBPEL引擎执行流程所用时间。

表 2 流程测试结果

流程	活动数量	分支数量	event数量	竞争数量	监控前时间/ms	监控后时间/ms
LA	75	11	137	9	685	705
AT	83	5	153	8	758	789
TI	36	5	55	3	432	447
OBP	151	13	249	14	1321	1369
CA	305	24	514	31	2114	2207
ORC	521	33	789	38	2946	3088

从表 2可以看出，本文所述方法是切实可行的，并且检测结果准确度很高，检测出来的竞争变量确实存在多个线程竞争读写。由于在执行过程中只是对变量访问做记录，所以额外增加的时间开销很小，表 2中监控前时间和监控后时间差相对于流程实际应用中执行时间而言完全可以忽略不计。为了尽可能多地检测数据竞争，本文测试用例采用文献[7]中的方法来生成。

4 结语

BPEL目前已经是描述服务组合流程的工业标准语言，对 BPEL流程的可靠性分析具有重要的实际意义。本文在传统程序数据竞争检测方法的基础上，提出了一个针对 BPEL流程数据竞争的检测方法。该方法结合了静态分析和动态监控两方面优点，提高了检测的准确性和实用性。

文中动态监控部分还有待优化。进一步工作主要是优化监控变量访问集合，在不影响检测效果的前提下缩小 event集合，提高检测速度。其次是 BPEL中的高级特性有待深入分析。

参考文献：

[1] Business Process Execution Language for Web services (BPEL4WS) [DB/OL]. [2008-09-28]. http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[2] YUAN YUAN, LI ZHONG-JI, SUN WEI. A graph-search based approach to BPEL4WS test generation[C] // ICSEA'06: Proceedings of the International Conference on Software Engineering Advance. Papeete: IEEE Computer Society, 2006: 14-14.

[3] LI ZHONG-JI, SUN WEI, JIANG Z B, et al. BPEL4WS unit testing: framework and implementation[C] // ICSEA'05: Proceedings of the IEEE International Conference on Web Services. Florida: IEEE Computer Society, 2005: 103-110.

[4] CHRISTIAENS M, BOSSCHERE K D. TRaDe, a topological approach to on-the-fly race detection in Java programs[C] // Proceedings of the Java Virtual Machine Research and Technology Symposium. California: Usenix Association, 2001.

[5] SAVAGE S, BURROWS M, NELSON G, et al. Eraser: A dynamic data race detector for multi-threaded programs[C] // ACM Transactions on Computer Systems. New York: ACM Press, 1997: 27-37.

[6] 陈胜，鲍亮，陈平，等. BPEL流程数据竞争和死锁检测算法[J]. 西安电子科技大学学报：自然科学版，2008，35(6)：1-8.

[7] YAN JUN, LI ZHONG-JI, YUAN YUAN, et al. BPEL4WS unit testing: test case generation using a concurrent path analysis approach[C] // ISSRE'06: 17th International Symposium on Software Reliability Engineering. Carolina: IEEE Computer Society, 2006: 75-84.