基于XCFG的BPEL数据流属性分析与验证

吉顺慧,李必信,邱 栋
(东南大学计算机科学与工程学院,江苏南京 211189)

摘 要: BPEL组合服务实现了Web服务的复用和增值,但其复杂性带来了一定的挑战.例如,BPEL流程中正确的数据流对确保服务组合的正确性是十分重要的,然而现有的研究很少关注这类问题.本文提出一种基于扩展控制流图(XCFG)的BPEL流程数据流属性验证方法,利用XCFG对BPEL流程进行形式建模,设计相应的算法来分析和验证典型的数据流属性,如定义-使用一致性,无死锁和可达性.理论分析和实验均表明该方法是有效的.

关键词: Web服务组合; 扩展控制流图 (XCFG); 数据流属性; 验证
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2013)07-1365-06
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.07.019

XCFG-Based Analysis and Verification for Data Flow Property of BPEL

JI Shun-hui, LI Bi-xin, QIU Dong
(*Department of Computer Science and Technology, Southeast University, Nanjing, Jiangsu* 211189, *China*)

Abstract: BPEL achieves reusability and value-adding of Web service through composing individual services. Meanwhile, it brings some challenges because of its complexity. For example, guaranteeing the data flow correct is very important for the correctness of BPEL process. However, few literatures focus on the data flow related analysis and verification. In this paper, we proposed an XCFG based technique, which models BPEL with XCFG and designs corresponding algorithms to analyze and verify the typical data flow property of BPEL, such as the consistency of define-use, deadlock-free and reachability. Both theoretical and experimental study validates the effectiveness of the XCFG based technique.

Key words: Web service composition; extended control flow graph (XCFG); data flow property; verification

1 引言

随着SOA(Service Oriented Architecture)结构体系的不断完善,基于Web服务的各种应用逐渐增多.BPEL(Business Process Execution Language)将现有基本的Web服务按照一定的应用逻辑组合成具有更复杂功能的组合服务,实现了服务的复用和增值[1].BPEL流程中控制流和数据流的正确使用是保障服务组合正确性的重要前提.目前大多数研究主要关注控制流的分析与验证[2~7],针对数据流的研究却很少.然而组合流程中数据的正确使用是十分重要的,因此有必要对组合服务的数据流进行分析并验证数据流相关的属性.

本文针对BPEL数据流问题提出一种基于扩展控制流图XCFG(eXtended Control Flow Graph)的属性验证方法,该方法首先对BPEL进行形式建模,能够比较完整地刻画BPEL中组件服务的交互流程,并且考虑了流程中的数据交互,以便进一步分析和验证BPEL的数据流属性.其次,定义了BPEL到XCFG的转换规则.最后,基于XCFG提出了简单有效的方法对定义-使用一致性,无死锁和可达性这三种数据流相关属性进行分析与验证.在此基础上,实现了原型工具XCFGV4BPEL,实验结果与分析说明了方法的有效性.

2 扩展控制流图XCFG

XCFG在传统的控制流图(CFG)基础上增加了对BPEL流程并发控制和同步依赖的描述,并通过对结点和边增加信息域的来记录相关数据信息,以便进行数据流属性的分析和验证.

定义: XCFG = (N, E, F),其中 $N = N_N \cup N_S \cup N_E \cup N_C$ 是结点集合,N_N, N_S, N_E, N_C 分别表示BPEL流程中基本活动,*sequence*,选择结构化活动(包括*if*, *pick*, *while*和*repeatUntil*)和*flow*;$E = E_C \cup E_L$ 是边集合,E_C 表示结

收稿日期:2012-07-17;修回日期:2013-01-15;责任编辑:蓝红杰
基金项目:国家自然科学基金(No.60973149);博士点基金(No.20100092110022);江苏省高校科研成果产业化推进项目(No.JHB2011-3);

点之间顺序,选择或并发的控制依赖关系(SEQUENCE,CHOICE 或 CONCURRENCY 类型),E_L 表示并发之间的同步依赖 *link*;F 记录信息域:对于 XCFG 的结点,F = (*id*, *name*, *jc*, *outEdges*, *outLinks*, *inLinks*, *readVars*, *writeVars*),其中 *id* 为结点唯一的编号,*name* 和 *jc* 为对应 BPEL 活动的名称和 *joinCondition*,*outEdges* 和 *inEdges* 为结点的出边集合和入边集合,*outLinks* 和 *inLinks* 分别记录以该活动结点为起始活动和目标活动的 *link* 集合,*readVars* 和 *writeVars* 分别记录该活动结点使用和定义的 *variable* 集合;对于 XCFG 的边,F = (*id*, *source*, *target*, *type*, *name*, *tc*),其中 *id* 为边唯一的编号,*source* 和 *target* 记录边的起始结点和目标结点,*name*, *tc* 表示 E_L 对应 *link* 的名称和 *transitionCondition*, *type* 表示 E_C 的类型.

构造 BPEL 服务组合流程的形式化模型 XCFG 需要四个步骤:(1)将 BPEL 文档解析为 BOM;(2)为 BPEL 活动创建 XCFG 结点;(3)根据 BPEL 结构化活动中子活动的执行顺序,创建相应的 XCFG 边 E_C;(4)针对并发活动间存在的同步依赖关系 *link*,创建相应的 E_L. 为了简化形式化建模,我们利用开源包 *org. apache. ode. bpel. compiler. com*[8] 将 BPEL 文档解析为 BPEL 对象模型(BOM),可以通过调用 BOM 类相应的成员函数来抽取 BPEL 组合流程的相关信息. 针对 BPEL 基本活动和结构化活动,XCFG 的构造需要制定不同的策略. 每个基本活动创建一个 XCFG 基本结点 N_N,结构化活动的 XCFG 构造则只考虑结构本身,创建其与子活动之间的控制边,对 *flow* 分支之间的 *link* 创建相应的 E_L,然后基于 BOM 描述结点和边的信息域 F. 由于空间限制,XCFG 具体的构造算法可参考[9].

XCFG 实际上是 BPEL 的一种控制流图,与传统程序的 CFG 类似且具有很好的扩展性,随着服务应用规模的增大,可比较容易地通过增加 XCFG 的结点和边来描述. 转换的可行性可得到保障. 至于 XCFG 模型的正确性证明需要从语义层次来判断它是否能够描述 BPEL 的所有执行,可以借鉴 CFG 相同的思路证明转换的正确性[10],这不是本文的研究重点,因此不做详细讨论.

3 BPEL 流程的数据流属性及验证

BPEL 流程中与数据流相关的元素主要有 *variable* 和 *link* 两种变量. *variable* 在 *process* 或 *scope* 中进行声明,在它们包含的活动中定义和使用,用于存储流程与组件服务之间交互的数据信息. *link* 只能在 *flow* 活动中定义和使用,用来实现并发活动之间的同步依赖. 本文重点关注 BPEL 流程中三种典型的数据流,讨论如何对它们进行分析和验证.

3.1 定义 – 使用一致性

与传统软件一样,BPEL 流程中使用未初始化的变量会引发错误,定义未使用的变量会导致冗余. 另外,当并发活动对同一个变量定义时,之后的使用只取其中一个值,其余的都丢失了,造成并发的定义 – 定义冲突. 当并发活动分别对同一个变量定义和使用时,运行时变量使用的值是不确定的,导致并发的定义 – 使用冲突. 这些情况都属于 BPEL 流程中变量的定义 – 使用缺陷,应当避免.

属性 1 当且仅当 BPEL 流程中不存在变量使用未定义、定义未使用以及定义、使用冲突,我们认为该 BPEL 流程满足变量定义 – 使用一致性.

该属性只针对 BPEL 流程中的 *variable* 变量,定义包括 *receive* 的 *variable*, *invoke* 的 *outputVariable*, *assign* 的左边变量等,使用包括 *reply* 的 *variable*, *invoke* 的 *inputVarialbe*, *assign* 的右边变量等. 定义 – 使用一致性的验证如算法 1 所示,其中调用的 BEFORE( ), AFTER( )和 PARAL( )函数分别计算 XCFG 中在结点之前,之后以及与之并发执行的结点集合,都具有 $O(n)$的时间复杂度,其中 n 表示 XCFG 的结点数. 变量 *variable* 的定义结点和使用结点分别存储在集合 *defSet* 和 *useSet* 中,分析该变量是否存在定义 – 使用缺陷需要三个步骤:(1)分析该变量是否存在使用未定义. 语句 12 – 25 先找出 *useSet* 中第一次使用该变量的活动结点,然后分析该结点之前以及与之并发的结点集合中是否存在该变量的定义结点,若存在则说明该变量不存在使用未定义;(2)分析该变量是否存在定义未使用. 语句 26 – 39 对每个定义结点查找它的使用结点,若存在,判断这对定义 – 使用结点中间的活动结点是否存在对该变量的定义,存在则说明该定义结点未使用就被覆盖了;(3)分析该变量是否存在并发的定义使用冲突. 语句 40 – 50 对该变量的某个定义结点计算它的并发结点集合 *paralSet*,若 *paralSet* 中的某个结点也对该变量进行定义,则存在并发的定义 – 定义冲突;若 *paralSet* 中的某个结点使用该变量,则要进一步分析在该并发定义结点之前执行的结点集合中是否有对该变量的定义,若有则存在并发的定义 – 使用冲突. 只有排除了这三种情况,算法才能返回 *true*.

算法 1 ***isDefineUseConsistent*(*xcfg*)**

```
1:  nodeSet = xcfg. getNodeSet( ); // xcfg 所有的结点集合
2:  int nodeNum = nodeSet. size( );
3:  variableSet = xcfg. getVariableSet( ); // xcfg 所有的变量集合
4:  for(variable)∈ variableSet{
//计算定义变量集合 defSet 以及使用变量的集合 useSet
5:  for(node∈ nodeSet){
6:      if(variable∈ node. getWriteVariables( ))
7:          defSet. add(node);
8:      elseif(variable∈ node. gerReadVariables( ))
9:          useSet. add(node);}
```

```
//判断是否存在使用未定义
10:  if(! useSet.isEmpty()){
11:    int useNum = useSet.size();
12:    int usex, i = 0
13:    for(i = 1; i < useNum; i ++){
14:      beforeSet = BEFORE(useSet[usex]);
15:      if(useSet[i] ∈ beforeSet)
16:        usex = i;}
17:    befSet = beforeSet(useSet[usex]) ∪ PARAL(useSet[usex]);
18:    if(defSet.isEmpty())
19:      return false;
20:    int defNum = defSet.size();
21:    for(i = 0; i < defNum; i ++){
22:      if(defNum[i] ∈ befSet)
23:        break;}
24:    if(i = = defNum)
25:      return false;}
//判断是否存在定义未使用
26:  for(node ∈ defSet){
27:    afterSet = AFTER(node);
28:    commonSet = afterSet ∩ useSet;
29:    if(commonSet.isEmpty()){
30:      return false;}
31:    int commonNum = commonSet.size();
32:    int usex, i = 0
33:    for(i = 0; i < commonNum; i ++){
34:      if(commonSet[usex] ∈ BEFORE(commonSet[i])
35:        usex = i;}
36:    middleSet = afterSet ∩ BEFORE(commonSet[i]);
37:    if(! middleSet.isEmpty())
38:      if(defSet \ {node} ∩ midddleSet! = ∅)
39:        return false;}
//判断是否存在访问冲突
40:  for(node ∈ defSet){
41:    paralSet = PARAL(node);
42:    int prNum = paralSet.size();
43:    int(! prNum.isEmpty()){
44:      if(paralSet ∩ defSet! = ∅)
45:        return false;
46:      if(paralSet ∩ useSet! = ∅){
47:      beforeSet = BEFORE(node);
48:      if(defSet \ {node} ∩ beforsSet! = ∅)
49:        return false;}}}
50:  return true;
```

3.2　无死锁

BPEL 流程中 *link* 和 *variable* 不恰当的设计都会导致死锁.由 *link* 引发的死锁分析可参考文献[9].若 *variable* 约束的数据依赖关系与控制流形成相互依赖的环,同样也会引发死锁.当存在两个活动 A、B,B 控制依赖于 A 且 A 数据依赖于 B,那就构成了死锁环.*variable* 的不当设计引起的三种死锁环如图 1 所示:(a)仅由 *variable* 约束的数据依赖关系组成的环;(b)由 *variable* 约束的数据依赖关系和 *link* 约束的同步依赖关系组成的环;(c)由 *variable* 约束的数据依赖关系、有向边约束的顺序依赖关系以及 *link* 约束的同步依赖关系组成的环.

属性 2　当且仅当 BPEL 中既不存在 *link* 引起的死锁环,也不存在 *variable* 引起的死锁环,我们认为该 BPEL 流程无死锁.

无死锁的验证如算法 2 所示.语句 7 – 10 分析 *link* 引起的死锁环,至于 *variable* 的死锁分析,针对每一个 *variable*,语句 16 – 20 找出它的最先定义结点,如果该结点是并发结点,则分析是否存在其他的并发结点使用该 *variable*.语句 25 – 26 针对图 1(a)的死锁环,如果定义和使用 *variable* 的两个并发结点分别对另一个变量使用和定义,会导致死锁.语句 27 – 28 针对图 1(b),(c)的死锁环,如果 *variable* 的定义结点控制依赖于它的使用结点,也会导致死锁.只有当 XCFG 中既不存在 *link* 引起的死锁环,也不存在 *variable* 引起的死锁环,才能返回 *true*.

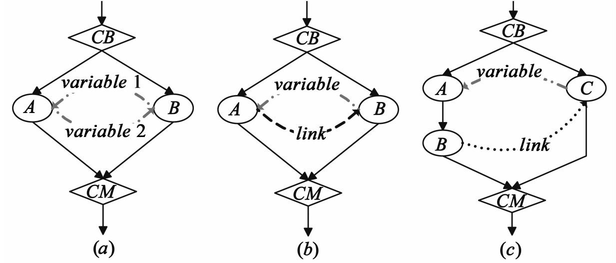

图1　*variable*引起的三种死锁环

算法 2　*isDeadlockFree*(*xcfg*)

```
1:  nodeSet = xcfg.getNodeSet();
2:  variableSet = xcfg.getVariableSet();
3:  linkSet = xcfg.getLinkSet(); //所有 link 集
4:  for(link ∈ linkSet){
5:      from = link.getSource();
6:      fromSet.add(from);} // fromSet:所有 link 的起始结点集合
//分析 link 可能引发的死锁环
7: if(! fromSet.isEmpty())
8:    for(node ∈ fromSet)
9:      if(node ∈ AFTER(node))
10:       return false;
11:  for(variable ∈ variableSet){
12:    for(int i = 0; i < nodeNum; i ++)
13:      if(variable ∈ nodeSet[i].writeVars)
14:        defSet.add(nodeSet[i]);
15:  int defNum = defSet.size();
16:  int defx = 0;    //记录最先定义变量的结点下标
17: for(int i = 1; i < defNum; i ++){
18:      beforeSet = BEFORE(defSet[defx]);
```

```
19:        if( defSet[ i ]∈ beforeSet)
20:          defx = i;}
21:  paralSet = PARAL( defSet[ defx ]);
22:  if(! paralSet. isEmpty())
23:    for( node∈ paralSet)
24:      if( variable∈ node. getReadVars){
25:        if( dfSet[ defx ]. getReadVars ∩ node. gerWriteVars! = ∅)
26:          return false;  //分析图 1(a)的死锁环
27:        elseif( defSet[ defx ]∈ AFTER( node))
28:          return false;}}  //分析图 1(b)(c)的死锁环
29:  return true
```

3.3 可达性

BPEL 中 *link* 不正确的设计会导致流程中某些活动是不可达的，如图 2 所示，(*a*)中 *link* 的起始活动和目标活动分别属于选择结构化活动的两个分支，由于选择结构化活动下只有一个分支活动能执行，而 *B* 活动又同步依赖于 *A*，因此 *B* 是不可达的. 图 2(*b*)中活动 *C* 的两个入 *link* 来自选择结构化活动的不同分支，因此 AND 操作的 *joinCondition* 永为 *false*，*C* 也是不可达的.

属性 3　当且仅当 BPEL 流程中不存在不可达的活动，我们认为该 BPEL 流程满足可达性.

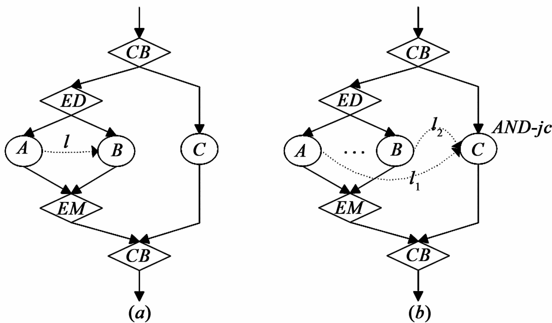

图2　活动不可达的两种情况

算法 3　*isReachable*(*xcfg*)

```
1:  linkSet = xcfg. getLinkSet();
//处理图 2(a)的情况
2:  for( link∈ linkSet){
3:    anc = findLCA( link. source, link, target);
4:    if( anc. getOutEdge(0). getType() == CHOICE);
5:      return false;}
//处理图 2(b)的情况
6:  for( link1∈ linkSet){
7:    for( link2∈ linkSet){
8:      if(( link1! = link2)( link1. target = = link2. target)){
9:      if( link1. target. jc 为 AND 操作){
10:       anc = findLCA( link1. source, link2. source);
11:         if( anc. getOutEdge(0). getType() == CHOICE)
12:           return false;}}}}
13:  return true;
```

可达性的验证如算法 3 示，其中调用的 *findLCA*() 函数计算两个结点的最先公共祖先结点，可用于判断这两个活动所属的结构化活动类型，其时间复杂度为 $O(n)$. 语句 2 – 5 处理图 2(*a*)的情况，对每一个 *link* 查找它的起始活动结点和目标活动结点，判断它们是否来自选择结构化活动的不同分支. 语句 6 – 12 处理图 2(*b*)的情况，对任意两个 *link* 检查它们是否具有相同的目标活动，并判断该目标活动结点的 *jc* 是否是 AND 操作，若满足则查找它们的起始活动结点，并分析其是否是选择结构化活动的不同分支. 只有排除了这两种情况才能返回 *true*.

4　实验分析

我们在前期理论和方法的基础上开发了 BPEL 建模和验证的原型工具 XCFGV4BPEL 来实现定义 – 使用一致性，无死锁和可达性这三种数据流属性的自动化验证，本节我们将讨论如何通过一些 BPEL 组合服务的实例来证明方法的有效性并分析方法的时间复杂度.

4.1　有效性分析

为了证明这三种数据流属性验证算法的有效性，本文选取了 92 个 BPEL 实例作为研究对象. 其中 45 个来自开源包 *apache ode* 的 *apache-ode-sources*-1.3.3/*bpel-test*，47 个搜集自互联网，并手动注入缺陷使它们不满足定义 – 使用一致性，无死锁，可达性中的一个或多个属性. 通过在原型工具 XCFGV4BPEL 上运行这 92 个实例，分析我们的验证方法是否总能正确地检测出不满足数据流属性的实例. 用 *act* 表示实际上不满足属性的 BPEL 实例数，*num* 表示实验分析出的不满足属性的 BPEL 实例数，数据流属性验证算法的有效性可以按照公式 $e = \frac{num}{act} \times 100\%$ 来计算.

表 1　实验结果统计

正确实例	属性 1 不满足	属性 2 不满足	属性 3 不满足	属性 1&2 不满足	属性 1&3 不满足	属性 2&3 不满足	属性 1&2&3 不满足
43	12	10	10	5	5	5	2

92 个实例的实验结果如表 1 所示，除了 45 个手动注入缺陷的实例不满足相关属性，还验证出“*temp*0000. *bpel*”和“*TestActivityFlow*. *bpel*”实例不满足定义 – 使用一致性. 经分析，它们确实存在变量并发的定义 – 定义冲突. 所有不满足属性的实例都正确地验证出来了，因此定义 – 使用一致性，无死锁，可达性验证的有效性都为 100% . 从而证明本文方法是有效的.

4.2　时间复杂度分析

通过理论分析，定义 – 使用一致性，无死锁验证算法的时间复杂度为 $O(n^2)$. 可达性验证算法的时间复

杂度为 $O(n)$,由于 92 个实例中大部分例子没有 *link*,导致实验数据不足难以得到准确的分析结果.下面将通过实验分析定义-使用一致性,无死锁验证算法的性能,其时间消耗趋势如图 3 所示,其中横轴表示 XCFG 的规模,竖轴表示验证算法消耗的时间.我们用 XCFG 模型的结点数来衡量它的规模,由于实例规模较小,每个验证算法消耗的时间很少,为了降低实验运行时的外在干扰,我们对每个实例执行 1000 次来记录运行时间.从图中可以看出,两个验证算法的时间消耗与 XCFG 规模之间都存在一个平方的多项式曲线关联,其拟合函数标识在相应的图中,从对应的判断系数可知该多项式曲线有较好的拟合优度,因此实验分析结果跟时间复杂度的理论分析是一致的,这样的时间消耗也是可以接受的.

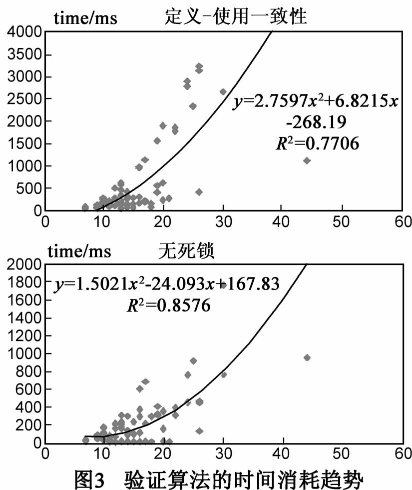

图3　验证算法的时间消耗趋势

5　相关工作比较

目前 Web 组合服务验证大部分采用模型检测技术,基于有限自动机[2,3]、进程代数[4,5]、Petri 网[6,7]的组合服务验证技术虽然已经形成了比较完善的理论和工具支撑,但这些研究大都是针对服务组合流程的控制流分析,很少有考虑到数据流的.

而考虑到数据交互的研究部分是针对数据相关的行为属性,Kazhamiakin 等[11]用 ground 模型和状态迁移系统建模 BPEL 组合服务的数据和控制流程来验证数据行为属性,Moser 等[12]和 Monakova 等[13]利用 CSSA 模型描述 Web 组合服务的数据流信息,分别在此基础上完善 Petri 网模型进行数据行为属性的验证[12],以及用逻辑断言来描述验证基并利用 SMT 工具 Yices 进行验证[13].这些研究都需要多个模型来描述组合服务的数据流和控制流,工作量十分大,而本文仅需 XCFG 模型就可解决这个问题.

针对数据流属性的验证也有不少工作,Yang 等[14]基于 BPEL 的控制流图分析流程中的数据依赖关系并检测变量的定义使用缺陷,但这一理论并未得到工具的支持实现自动化.本文的 XCFG 建模更细致,对 link 的描述更简单,另外本文还验证了与数据相关的死锁、不可达问题,且开发了原型工具 XCFG4BPEL 来实现属性验证的自动化.Lei 等[15]在 Petri 网加入数据库所来表示 BPEL 流程中的变量,分析模型中的每个可达标识是否存在变量使用未定义和并发的定义使用冲突.Xu 等[16]构造了一种包含数据信息的有色 Petri 网模型 WS-CPN,用 CPN ML 描述 BPEL 复杂的数据类型,并讨论如何验证数据的定义使用缺陷.相比较这两篇文献而言,本文对 Web 组合服务流程正确性的验证更全面些,但由于采用的形式化模型不同,无法实验比较验证算法的性能.

6　总结

本文主要对 BPEL 组合服务流程中数据流相关属性进行分析和验证,主要包含以下几个工作:(1)提供了一种形式化模型 XCFG 描述 BPEL 的控制流和数据流信息,并介绍 BPEL 到 XCFG 的转换规则;(2)提供了算法来分析和验证定义-使用一致性,无死锁和可达性三种数据流属性;(3)实现了原型工具 XCFGV4BPEL;(4)实验分析方法的有效性.下一步工作将包括完善 BPEL 复杂特性的建模(如异常处理机制),数据流行为属性和其他非功能属性的验证,以及基于 XCFG 的验证方法在 CPS 中的应用[17]等.

参考文献

[1] Alves A,Arkin A,Askary S.Web services business process execution language version 2.0[EB/OL].http://docs.oasis-open.org/wsbpel/20/OS/wsbpel-v2.0-OS.html.2007-4-11.

[2] Diaz G,Pardo J J,et al.Design and verification of web services compositions with timed automata[J].Electronic Notes in Throretical Computer Science,2006,157(2):19-34.

[3] Cambronero M E,Diaz G,Valero V,et al.Validation and verification of web services choreographies by using timed automata[J].The Journal of Logic and Algebraic Programming,2011,80(1):25-49.

[4] Salaum G,Bordeaux L,Schaerf M.Describing and reasoing on web services using process algebra[A].Proceeding of IEEE International Conference on Web Servce[C].Washinton:IEEE Computer Society,2004.43-51.

[5] Peng Y,Ye L,Zheng Z,et al.Automatic service composition verification based on Pi-calculus[A].Proceeding of International Conference on E-Business and Information System Security[C].Wuhan:IEEE,2009.1-4.

[6] Hinz S,Schmidt K,Stah C.Transforming BPEL to petri-nets

[J]. Lecture Notes in Computer Science, 2005, 3649: 220 – 235.
[7] Song W, Ma X. Ye, C, Dou W, Lu J. Timed modeling and verification of BPEL processed using timed Petri nets[A]. Proceeding of the 9th International Conference on Quality Software[C]. Cheju: IEEE, 2009.92 – 97.
[8] Package org. apache. ode. bpel. compiler. bom[CP/OL]. http://www.jarvana.com/javana/view/or/apache/ode-bpel-compiler/1.3.4/ode-bpel-compiler-1.3.4-javadoc.jar! /org/apache/od.
[9] 张功源.BPEL组合服务并发结构的分析与验证[D].江苏南京:东南大学,2011.3.
Zhang Gongyuan. Analysis and Verification of Concurrent Construct for BPEL-Based Service Composition[D]. Nanjing, Jiangsu: Southeast University, 2011, 3. (in Chinese)
[10] Amighi A. Flow Graph Extraction for Modular Verification of Java Programs[D]. Stockholm: KTH Royal Institute of Technology, 2011.
[11] Kazhamiakin R, Pistore M. Static verification of control and data in web service compositions[A]. Proceeding of IEEE International Conference on Web Services (ICWS′06)[C]. Washington: IEEE Computer Society, 2006.83 – 90.
[12] Moser S, Martens A, et al. Advanced verification of distributed WS-BPEL business processes incorporating CSSA-based data flow analysis[A]. IEEE International Conference on Service Computing[C]. Salt Lake City: IEEE, 2007.98 – 105.
[13] Monakova G, Kopp O, et al. Verifying business rules using an SMT solver for BPEL processes[A]. Proceedings of the Business Process and Services Computing Conference (BPSC′09)[C]. Germany, 2009.81 – 94.
[14] Yang X, Huang J, Gong Y. Static data flow analysis and anomalies detection for BPEL[A]. Proceeding of International Conference on Test and Measurement (ICTM)[C]. Hong Kong: IEEE, 2009.18 – 21.
[15] Lei K, Zhang P. P, Lang B. Data access exception detecting of WS-BPEL process based on workflow nets[A]. Proceding of International Conference on Computational Intelligence and Software Engineering (CiSE)[C]. Wuhan: IEEE, 2010.1 – 6.
[16] Xu C, Qu W, Wang, H, etc. A Petri net-based method for data validation of web services composition[A]. Proceeding of 34 th International Computer Software and Applications Conference[C]. Seoul: IEEE, 2010.468 – 476.
[17] 朱敏,李必信,等.基于微分动态逻辑的CPS建模和属性验证[J].电子学报,2012,40(6):1126 – 1132.
Zhu Min, Li Bixin, et al. Transforming hybridUML to hybrid program for CPS property verification[J]. Acta Electronica Sinica, 2012, 40(6): 1126 – 1132. (in Chinese)

作者简介

吉顺慧 女,1987年出生于江苏南通,现为东南大学计算机科学与工程学院在读博士,主要研究方向为Web服务组合分析、测试与验证.
E-mail: shunhuiji@seu.edu.cn

李必信 男,1969年出生于安徽省庐江县,博士/博士后,现任东南大学计算机科学与工程学院教授、博士生导师,CCF高级会员,主要研究领域为软件分析、测试、验证,实证软件工程.
E-mail: bx.li@seu.edu.cn

邱 栋 男,1986年出生于江苏海门,现为东南大学计算机科学与工程学院在读博士,主要研究方向为软件分析与测试.
E-mail: dongqiu@seu.edu.cn