第41卷 第12期 西 安 交 通 大 学 学 报 Vol.41 №12
2007年12月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Dec. 2007

基于更改域的工作流过程演变算法

李伟刚[1,2]，沈钧毅[1]
（1.西安交通大学电子与信息工程学院，710049，西安；2.西北工业大学软件与微电子学院，710075，西安）

摘要：针对工作流过程演变时过程实例的跃迁策略不易确定的问题，提出了一种表达灵活的工作流过程模型，并将过程模型的更改转化成若干更改域，更改域包含三类元更改操作.为了确定过程实例能否平滑地迁移至新的过程模型，又提出了通过更改域边缘查找更改点的方法.比较更改点位置与当前过程实例状态，确定过程演变策略，并给出过程演变算法，从而实现了过程实例迁移.实际应用表明，所提算法可自动导出工作流过程演变策略，并实现过程实例的批量跃迁.

关键词：工作流；过程演变；更改域
中图分类号：TP391 **文献标识码**：A **文章编号**：0253-987X(2007)12-1406-05

Process Evolution Based on Change Domains

Li Weigang[1,2], Shen Junyi[1]
(1. School of Electronics and Information Engineering, Xi′an Jiaotong University, Xi′an 710049, China; 2. College of Software and Microelectronics, Northwestern Polytechnical University, Xi′an 710075, China)

Abstract: Aiming at the problem that the migrating strategies of process instances are difficult to determine in process evolution, a workflow process model with flexible expressiveness was proposed, and the change of it is transformed into some change domains which contain three kinds of meta-change operations. In order to determine whether a process instances can be migrated to a new process model smoothly, an approach to find out all points of change (PoC) through confining the change domains was also proposed. The positions of PoCs were compared to the current states of process instances in order to decide the strategies of process evolution, and the process evolution algorithm was given and the migration of process instances was realized. The practical applications show that the proposed algorithm can deduce the process evolution strategies of workflow automatically, and the process instances migration can be carried out in batches.

Keywords: workflow; process evolution; change domains

工作流系统存在的一个主要问题是其难以适应动态的业务环境，主要是工作流系统的柔性、动态性和自适应性较差[1].工作流系统响应变化的能力称为柔性，这种变化主要是指系统在运行时工作流模型发生改变.若变更工作流模型定义的行为内容，会引起工作流实例的运行路径发生变化.因此，需研究运行的过程实例如何动态迁移到新的过程模型之上，即过程演变[2]或动态更改[1,3]，相应地也有不同的处理方法[2-4]，但是这些方法所设定的迁移条件过于简单，不能满足实际应用的需求.另外，有些研究偏重于动态演变原理[1]，缺乏具体算法支持，所以实用性受到限制.

本文基于一种强表达能力、高灵活性的工作流过程模型，对其更改划分为更改域，从而找出判断过程实例能否平滑迁移到更改后过程模型的更改点，确定过程实例的迁移策略.这种过程演变方法能够适应多分支的过程模型，使得每个过程实例按照各自所取道的分支独立地实施迁移，控制性、灵活性

收稿日期：2007-03-14. 作者简介：李伟刚(1972～)，男，博士后，讲师；沈钧毅(联系人)，男，教授，博士生导师. 基金项目：陕西省科技攻关资助项目(2004K05-G45)；西安市信息技术专项基金资助项目(ZX06033).

高，符合实际应用的需要.

1 工作流模型

工作流模型必需描述实际业务过程的功能、行为、信息、操作和组织等5个方面. 其中，行为描述最基本、最重要[5]，它描述活动的执行依赖关系(控制流). 控制流的变更对过程演变的实施影响最大，所以本文主要给出行为的建模方法.

定义1 工作流 W 定义为三元组 (A,Q,L)，其中 A 是活动集合，Q 是起止标记集合，L 是活动间的执行依赖关系(称为链接). W 满足以下关系：①$A\cap Q=\varnothing$，$A\neq\varnothing$；②$Q=\{b\}\cup F$，其中 b 为起始标记，F 为结束标记集合，$F\neq\varnothing$；③$L\subset A\times A\cup\{b\}\times A\cup A\times F$；④设 $\mathrm{dom}(L)=\{i\mid\exists j:(i,j)\in L\}$，$\mathrm{ran}(L)=\{j\mid\exists i:(i,j)\in L\}$，则 $\mathrm{dom}(L)\cup\mathrm{ran}(L)=A\cup Q$.

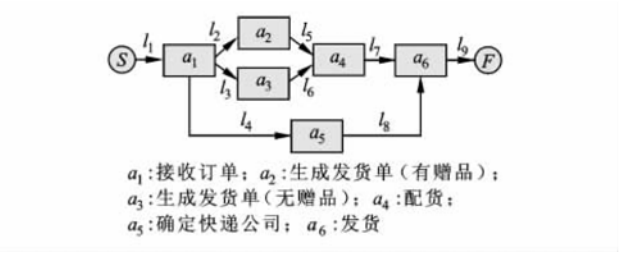
定义2 活动 $a_i\in A$ 定义为多元组 $(I,O,A_g,R,E,R_B,A_c,S,c)$. 其中：$I$ 和 O 分别是输入和输出的数据集合；A_g 和 R 分别是执行 a_i 的代理集合和所需资源的集合；E 是 a_i 运行期间接收事件的集合；R_B 为商业规则集合；A_c 是 a_i 上定义的动作集合；S 为 a_i 可能达到的状态集合；c 是补偿路径.

I 和 O 描述了 a_i 在运行时读取和写入的工作流变量集合. 就 W 而言，存在关系 $\bigcup_{i=1}^{n}(I(a_i)\cup O(a_i))\subseteq V$，其中 $n\in\mathbf{N}$ 为 W 中的活动数，V 是 W 中定义的工作流变量集合. E、R_B、A_c 组合起来可描述一种类似于ECA(Event Condition Action)规则的事件响应机制，称为事件-规则-动作(ERA)，它可灵活描述活动对外部事件的响应行为，实现过程与外部应用、分布式过程之间的细粒度通信和协调控制. ERA中的规则 R 比ECA中的条件具有更丰富的语义，适合描述企业中的复杂业务约束.

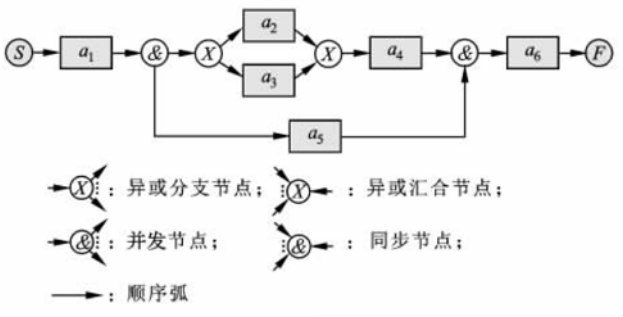
定义3 链接 $l_i\in L$，l_i 记作 $h\xrightarrow{e_s,C_i,A_c}t$，表示活动之间的执行依赖关系，定义为五元组 (h,t,e_s,C_i,A_c). 其中：h、t 分别表示 l_i 的头部和尾部，且 $h,t\in A\cup Q$；e_s 是 h 的一个状态转变事件，记作 $e(h.\mathrm{state}=s)$，$s\in S$ 为 h 的状态；C_i 是 l_i 上控制点或控制点分量组成的有序位置集合，表示为拟序集 $\langle C_i,\prec\rangle$；$A_c\in A_C$ 表示当 h 的状态变成 s 时，若满足 C_i 确定的控制依赖约束规则，在尾部 t 上进行的元操作(动作)的集合，A_C 是 W 中定义的动作集合.

在链接中可描述过程路由语义，当 $e_s\equiv e(h.\mathrm{state}=\mathrm{completed})\wedge A_c=\{\mathrm{start}\}$ 时，链接就表示活动顺次执行. 另外，链接还可以描述活动动态行为，即将一个活动运行状态的变化，与对其他活动行为的控制关联起来，这样就可为细粒度、高度灵活活动间的控制依赖关系提供一种建模方法，提高模型的表达能力. e_s、C_i 和 A_c 仍然构成了ERA机制，与活动中定义的ERA一样，由工作流引擎统一处理. 动作集合 A_C 开发成单独的组件库，供工作流引擎在运行时调用. A_C 可扩展，以进一步提高模型的表达能力.

以链接为核心定义工作流模型的行为，虽然描述分支、汇合关系不够直观，但是易于执行工作流引擎，简化工作流系统. 图1给出了一个订单处理过程的例子.



(a)用本文定义的链接描述的过程模型



(b)与图1a等价的用过程路由语义节点定义的过程模型

图1 过程模型示例

2 过程结构动态更改

工作流系统处于不确定的技术环境和动态的经营环境，导致其运行不可能与预定义的过程模型完全一致，而且企业的核心业务过程，如新产品开发过程等，一般具有数月甚至数年的寿命周期，很难在开始时做出完善或完整的定义. 这样，就要求工作流系统在运行时能适应过程模型的变更，即支持过程模型动态更改，这也是柔性工作流系统的重要特征.

更改过程模型的行为描述会导致过程结构调整，如果此时系统中存在对应的运行中的过程实例，就称之为过程结构动态更改，简称结构更改. 结构更改分为实例级更改和模型级更改，实例级更改是临时调整个别过程实例的执行路径，这种更改不会对

过程模型产生影响.模型级更改则是过程模型发生永久性的变更,这就必须处理与其对应的正在运行的经过程实例动态适应更改后的过程模型,有如下几种策略[2-3].

前进策略　所有当前运行中的实例按照旧过程模型执行,新启动的过程实例按照新模型执行.这种方法对运行中的过程实例不能立刻生效,所以对某些过程,特别是长周期过程,往往它是不可接受的.

重新开始策略　所有运行中的实例都被取消,并重新按新模型执行.这种方法抛弃了已有的工作成果,对多数业务过程它是不可接受的.

过程演变策略　运行中的过程实例迁移到新的过程模型,依过程实例当前的状态采取不同的迁移方案.系统中的过程模型一般存在 2 个甚至是更多的版本,所以一个过程实例可能同时从属于几个版本.

过程演变是实施过程结构动态更改时必须处理的难题,其关键是判断过程实例能否平滑地迁移至更改后的过程模型,若过程实例能够无条件地迁移,则称其兼容于新模型.下面引入跃迁点的概念.

定义 4　设 W 和 W' 分别为语法正确的工作流模型,$W'=M(W)$,M 是更改算子.假如 p 是 W 的过程实例,且 p 依据 W 执行到活动 a,其后继活动开始迁移到 W',则称 a 为 p 的跃迁点,记作 P_{PoT}.

工作流模型很可能存在并发分支,所以过程实例的一次迁移,其跃迁点不一定是惟一的.

3　更改域和更改点

对过程结构的更改是通过更改操作完成的.设 $W'=M(W)$,$A,L,F\subset W$,$A',L',F'\subset W'$ 分别为 W 和 W' 中的活动、链接和结束标记集合,当未进行更改操作时,则有 $W'=W$.本文定义了 4 种更改操作,其中 3 种是元操作,一种是组合操作.

操作 1　删除,分为删除链接、删除活动和删除结束标记.

删除链接:$\forall W$,$\exists l\in L$,删除链接 l 记作 $\text{delete}^W(l)$,使得 $l\notin L'$.

删除活动:$\forall W$,$\exists a\in A$,删除活动 a 记作 $\text{delete}^W(a)$,使得 $a\notin A'\wedge\{^{|\rightarrow|}a\}\subset L'\wedge\{a^{|\rightarrow|}\}\subset L'$,其中 $^{|\rightarrow|}a$ 和 $a^{|\rightarrow|}$ 分别表示 a 在 W 中是直接前驱链接和直接后继链接.

删除结束标记:$\forall W$,$\exists f\in F$,删除结束标记 f 记作 $\text{delete}^W(f)$,使得 $f\notin F'\wedge\{^{|\rightarrow|}f\}\subset L'$,其中 $^{|\rightarrow|}f$ 表示 f 在 W 中是直接前驱链接.

操作 2　插入,分为插入链接和插入活动.

插入链接:$\forall a_1,a_2\in A$,在 a_1 和 a_2 之间插入链接 $l:a_1\rightarrow a_2$,记作 $\text{insert}^W(l)$,使得 $l\in L'$.

插入活动:向 W 中插入活动 a,记作 $\text{insert}^W(a)$,使得 $a\in A'$.

操作 3　修改,分为修改链接和修改活动.

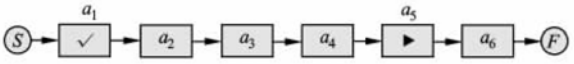
修改链接:$\forall l\in L$,将 l 修改成 l',记作 $\text{update}^W(l)$,使得 $l'\in L'$.

修改活动:$\forall a\in A$,将 a 修改成 a',记作 $\text{update}^W(a)$,使得 $a'\in A'$.

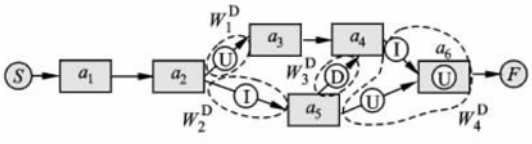
以上 3 种更改操作称为元更改操作,它们无法保证更改后的过程模型 W' 语法正确,所以不能单独使用.元操作需要组合起来构成区域更改操作,才能实际应用.

操作 4　区域更改,它是指对过程模型 W 进行了若干元更改操作,更改后 W' 与 W 相比,在某几个区域发生了变化,且不存在语法错误.对 W 的区域更改操作,记作 $\text{DM}^W(W^{\text{D}})$,其中 W^{D} 为更改了的过程模型元素集合,显然 $W^{\text{D}}\subset W\cup W'$.$\text{DM}^W(W^{\text{D}})=\{\text{delete}^W(A^{\text{d}}),\text{delete}^W(L^{\text{d}}),\text{delete}^W(F^{\text{d}}),\text{insert}^W(A^{\text{i}}),\text{insert}^W(L^{\text{i}}),\text{update}^W(A^{\text{u}}),\text{update}^W(L^{\text{u}})\}$,其中 $A^{\text{d}}\subset A$、$L^{\text{d}}\subset L$、$F^{\text{d}}\subset F$、$A^{\text{i}}\subset A'$、$L^{\text{i}}\subset L'$、$A^{\text{u}}\subset A$、$L^{\text{u}}\subset L$ 分别为被删除、插入和更新的模型元素集合.

将区域更改操作所涉及的模型元素标识于过程模型之中,得到过程模型的更改视图 W^{C},过程结构更改和更改域如图2所示.可见,W^{D} 在过程模型的

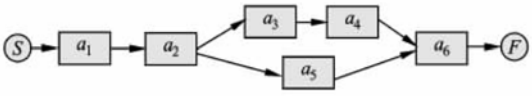


(a)原过程模型 W

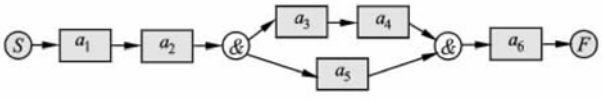


Ⓓ、Ⓘ、Ⓤ分别表示删除、插入和修改的模型元素

(b)过程模型的更改视图 W^{C} 和更改域



(c)更改后的过程模型 W'



(d)用过程路由语义节点定义的 W' 的等价过程模型

图 2　过程结构更改和更改域

更改视图上由若干连通的区域 W_i^D 构成，即 $W^D=W_1^D\cup W_2^D\cup\cdots\cup W_i^D$，将 W_i^D 称为更改域. 由更改域的连通特性知，$W_1^D\cap W_2^D\cap\cdots\cap W_i^D=\varnothing$.

为了判断过程实例的兼容性，需研究跃迁点与更改域的关系，为此应明确更改域的边缘.

定义 5 更改域边缘记作 $\uparrow W_i^D$，$\uparrow W_i^D\subseteq W_i^D$，满足

$$\forall a\in\uparrow W_i^D,\exists\{{}^{|\rightarrow|}a\}\notin W_i^D$$

$$\forall l\in\uparrow W_i^D,\exists{}^{|\cdot|}l\notin W_i^D$$

其中 ${}^{|\cdot|}l$ 表示链接 l 的直接前驱活动.

由删除操作的定义可知，被删除的 a 或 f 不会位于更改域边缘，因为必然存在 $\{{}^{|\rightarrow|}a\}\subset W^D$ 或 $\{{}^{|\rightarrow|}f\}\subset W^D$. 对于插入活动，有如下引理.

引理 1 设 $A\subset W$ 是更改前 W 中所有活动的集合，$\forall W$ 的区域更改操作为 $DM^W(W^D)$，如果 $\exists a,a\in W^D\wedge a\notin A$，执行操作 $insert^W(a)$，则 $\{{}^{|\rightarrow|}a\}\subset W^D$.

证明 由定义 1 可知，若执行操作 $insert^W(a)$ 后，使得 $a\in A'$，则 $\exists L^a\subset L'\wedge L^{a^*}\subset L'$，其中 $L^a=\{l_i:a_i\rightarrow a,a_i\in A,i=1,\cdots,m\}$，$L^{a^*}=\{l_j:a\rightarrow a_j,a_j\in A,j=1,\cdots,n\}$，因此 $\{{}^{|\rightarrow|}a\}=L^a$，$l_i\in\{{}^{|\rightarrow|}a\}$. 由于 $a\in W^D$，所以无论 a_i 是否属于 W^D，由链接的定义知，都有 $l_i\in W^D$，$i=1,\cdots,m$，即 $\{{}^{|\rightarrow|}a\}\subset W^D$.

可见，插入的 a 不可能位于更改域边缘. 所以，在确定更改域边缘时，可将它们排除在外.

为了使过程实例兼容于更改后的过程模型，其充分条件是保证跃迁点在其执行路由方向上不超过任何更改域边缘. 然而，在对过程模型进行区域更改操作后，可能产生多个更改域，每个更改域的边缘并不一定都会对兼容性判断起实质性作用，所以本文引入更改点 P_{PoC} 的概念.

定义 6 设 W 和 W' 分别为语法正确的工作流模型，$W'=M(W)$，$A,L\subset W$，$A',L'\subset W'$ 分别是 W 和 W' 中所有活动和链接的集合，那么如果 $\forall a_i\in A\wedge a_i\in A',a_i\in B_i\subset W\wedge a_i\in B_i'\subset W'$，$B_i$、$B_i'$ 分别为 W 和 W' 上的分支，$\exists\{{}^{\cdot}a_i|{}^{\cdot}a_i\in A\}=\{{}^{\cdot}a_i|{}^{\cdot}a_i\in A'\}\wedge\{a_i^{\rightarrow}|a_i^{\rightarrow}\in L\}=\{a_i^{\rightarrow}|a_i^{\rightarrow}\in L'\}\wedge(\{a_i^{|\cdot|}|a_i^{|\cdot|}\in A\}\neq\{a_i^{|\cdot|}|a_i^{|\cdot|}\in A'\}\vee\{a_i^{|\rightarrow|}|a_i^{|\rightarrow|}\in L\}\neq\{a_i^{|\rightarrow|}|a_i^{|\rightarrow|}\in L'\})$，$\{{}^{\cdot}a_i\}$ 和 $\{a_i^{\rightarrow}\}$ 分别是 a_i 在 B_i 和 B_i' 上的所有前驱活动集合和所有前驱链接集合，$\{a_i^{|\cdot|}\}$ 和 $\{a_i^{|\rightarrow|}\}$ 分别是 a_i 在 B_i 和 B_i' 上的直接后继活动集合和直接后继链接集合，则称 a_i 为更改算子 M 在分支 B_i 上作用于 W 的更改点，记作 P_{PoCi}. 更改点确定算法(算法 1)如图 3 所示.

算法在 W 的每条可达路径上通过查找最靠前的更改域边缘节点，再由更改点的定义获得更改点集合，由此保证了算法的有效性. 假设存在 n 个更改域，每个更改域包含 m 个活动或链接，则通过定义可确定更改域边缘的算法复杂度 $O(n^2)$. 同理，根据更改域边缘生成更改点集合的算法复杂度也是 $O(n^2)$，所以算法 1 的复杂度为 $O(n^2)$. 在实际应用中，更改域的数量、更改域包含的活动和链接数、可达路径数都较少，所以可保证算法的效率.

以图 2 为例，过程模型 W 通过重新组合 a_2 至 a_6 的路由顺序，模型 W' 产生了 4 个更改域 W_1^D、W_2^D、W_3^D 和 W_4^D(见图 2b)，它们的边缘分别为 $a_2\rightarrow a_3$、$a_2\rightarrow a_5$、$a_4\rightarrow a_5$、$\{a_4\rightarrow a_6,a_5\rightarrow a_6\}$，由于 W 中存在一条可达分支，所以通过算法 1 得到与之对应的更改点集合 $\{a_2\}$.

更改点确定后，可判断出过程实例与更改后的过程模型的兼容性，并确定过程实例的迁移策略.

输入：要更改的过程模型 W，过程更改定义 PML.
输出：更改点集合 $\{P_{PoC1},\cdots,P_{PoCm}\}$.

1. 由PML确定更改区域 $W_1^D,\cdots,W_n^D$；
2. FOR 每个更改区域 $W_j^D,j=1,\cdots,n$
3. 确定更改区域边缘 $\uparrow W_j^D$；
4. NEXT
5. $\uparrow W^D\leftarrow(\uparrow W_1^D\cup\cdots\cup\uparrow W_j^D\cup\cdots\cup\uparrow W_n^D)$；
6. FOR $\uparrow W^D$ 中的每个活动或链接 $e,e\in\uparrow W^D$
7. FOR W 的每条可达分支 $B_j,j=1,\cdots,m$
8. IF ${}^{|\cdot|}e\in B_j$
9. IF P_{PoCj}=NULL
10. $P_{PoCj}\leftarrow{}^{|\cdot|}e$；
11. ELSE IF $\{e\in{}^{\bullet}P_{PoCj},P_{PoCj}\}$
12. $P_{PoCj}\leftarrow{}^{|\cdot|}e$；
13. END IF
14. END IF
15. NEXT
16. NEXT
17. RETURN $\{P_{PoC1},\cdots,P_{PoCm}\}$

图 3 更改点确定算法

4 过程演变

由工作流引擎执行机制可知，过程实例中包含的活动实例在同一时刻只能处于 OPEN 或 CLOSED 2 种状态之一，如果有 2 个或 2 个以上的活动实例处于 OPEN 状态，则它们必定位于不同的执行路径. 假设 W 的任一实例 p 尚未发生过任何过程演变，p 中处于 OPEN 状态的 $a_1,\cdots,a_m$ 分别位于可执行路径 $B_1,\cdots,B_m$ 上，若 W 发生变更，使得 $B_1,\cdots,B_m$ 上的更改点分别为 $P_{PoC1},\cdots,P_{PoCm}$，则 $a_i,i=1,\cdots,m$ 与更改点 $P_{PoCi},i=1,\cdots,m$ 之间有如下 2 种

关系之一:①$a_i \in \{ {}^{\bullet}P_{PoCi}, P_{PoCi}\}$,$i=1,\cdots,m$;②$\forall a_i$,$\exists a_i \in \{P^{\bullet}_{PoCi}\}$,$i=1,\cdots,m$,其中$\{P^{\bullet}_{PoCi}\}$表示路径$B_i$上的更改点$P_{PoCi}$的后继活动集合.

对于关系①,显然过程实例p与更改后的过程模型是兼容的,能够平滑地实现过程演变,此时B_i上的跃迁点P_{PoTi}可以选取a_i、P_{PoCi}或它们之间的在B_i上的任意活动节点.若$P_{PoTi}=a_i$,则称过程演变为即时演变;若$P_{PoTi}=P_{PoCi}$,则称过程演变为极限演变.本文采用即时演变策略,则有如图 4 所示的过程演变算法.

```
输入: 实施过程演变的过程实例p, 过程更改定义
      PML.
输出: 更改是否成功.
1.  suspend(p);
2.  获取p对应的过程模型W, 根据PML按照算法1
    确定更改点集合{P_PoC 1,…, P_PoC m};
3.  FOR W的每条分支B_i, i=1,…,m
4.    在p可执行的分支b_i上查找处于OPEN状态的
      活动a_i (b_i为对应B_i的分支实例);
5.    IF a_i≠NULL∧a_i∈{P•_PoC i}
6.      println ("过程实例不兼容!"); restart(p);
7.      RETURN false;
8.    END IF
9.  NEXT
10. W_temp←W;
11. 由PML定义的更改操作和更改内容修改W_temp;
12. 验证W_temp的语法和语义正确性, 保证过程结构
    中不存在冲突;
13. 将W_temp保存成W的一个新版本W';
14. 将p对应的过程模型改成W';
15. restart(p);
16. RETURN true;
```

图 4　过程演变算法

只有当所有分支上处于 OPEN 状态的活动在执行路径上都不超过更改点的情况下,过程演变算法才能保证过程实例跃迁,该算法复杂度为$O(n)$.若不符合上述条件,则一定满足关系②.

过程实例p由于存在运行位置超出更改点的执行路径,所以与更改后的过程模型不兼容.此时要想实现过程演变,必须取消并回滚或补偿某些已经完成的活动,使p兼容,至少达到极限演变.这需要用如图 5 所示的伪码来更改过程演变算法的第 5 至 8 行,其中方法 abort(a_i)用于取消活动实例,这将会放弃活动a_i的所有执行成果.若a_i上定义了补偿路径,方法 compensate(a_i)通过执行补偿路径,将a_i的执行上下文还原.

```
IF a_i∈{P•_PoC i}
  WHILE a_i≠P_PoC i
    abort(a_j); compensate(a_j); a_i←|a_i;
  END WHILE
END IF
```

图 5　更改过程演变算法的第 5 至 8 行的伪代码

5　结束语

现有过程演变方法不能很好地解决复杂多分支过程模型的动态更改问题,关键是对过程实例的兼容性研究不足.本文引入一种具有强表达能力的过程模型,通过界定更改区域并计算出更改点,进而确定适合过程实例迁移的跃迁点,以此解决多分支复杂过程的动态演变问题.本文方法在作者开发的柔性工作流管理系统 AdaptoBPM 中得到应用,实现了过程演变的自动化.应注意,本文给出的过程演变准则比基于轨迹仿真[2]的方法严格,若过程中存在并行分支,可能会使得语义上可实现跃迁的过程实例无法迁移,这将是下一步需深入研究的问题.

参考文献:

[1] van der Aalst W M P. How to handle dynamic change and capture management information: an approach based on generic workflow models [J]. International Journal of Computer Systems, Science, and Engineering, 2001, 16(5):295-318.

[2] Casati F, Ceri S, Pernici B, et al. Workflow evolution [J]. Data and Knowledge Engineering, 1998, 24(3): 221-238.

[3] Sadiq S. Handling dynamic schema changes in workflow processes [C]// Proceedings of the 11th Australian Database Conference. Los Alamitos, USA: IEEE Computer Society, 2000:120-126.

[4] 魏登萍, 姜新文. 工作流演进变化中迁移策略的自动生成 [J]. 计算机应用, 2006, 26(4):995-997.
Wei Dengping, Jiang Xinwen. Automatic building of transfer policy in workflow evolution changes [J]. Computer Applications, 2006, 26(4):995-997.

[5] van der Aalst W M P, ter Hofstede A H M. Verification of workflow task structures: a Petri-net-based approach [J]. Information Systems, 2000, 25(1):43-69.

(编辑　苗凌)