工作流活动多实例的调度控制*

孙瑞志[1+], 史美林[2]

[1](中国农业大学 信息与电气工程学院,北京 100083)

[2](清华大学 计算机科学与技术系,北京 100084)

Schedule of Activity Instances in Workflow Management System

SUN Rui-Zhi[1+], SHI Mei-Lin[2]

[1](College of Information and Electrical Engineering, China Agricultural University, Beijing 100083, China)

[2](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-62736755, Fax: +86-10-62736746, E-mail: sunrz@cau.edu.cn, http://www.cau.edu.cn

Received 2003-10-24; Accepted 2004-11-15

Sun RZ, Shi ML. Schedule of activity instances in workflow management system. *Journal of Software*, 2005,16(3):400−406. DOI: 10.1360/jos160400

Abstract: Supporting multiple instances of one activity can enable workflow management system to be flexible on handling workflow process. When handling multiple instances the main problem is the schedule of instances. After analyzing the assignment and the join of multiple instances, this paper proposes the context of workflow activities and then formally describes the semantic of a multi-instance activity. Base on the formal presentation, the concept of Shell is proposed to control the assignment and submission of the multiple instances. The Shell can control the multiple instances to run synchronically and control the progress of workflow process according to the semantics of activities. The Shell gives a solution to schedule and control of activity multi-instances in workflow process.

Key words: workflow; workflow management system; activity instances; Shell; synchronization

摘 要: 支持多实例的工作流管理系统为工作流过程处理带来极大的灵活性,活动多实例要解决的主要问题之一是多实例的调度控制.在分析了多实例的分配和汇聚等问题之后,针对过程中活动间不同活动语义的上下文,对活动多实例的活动属性进行了统一的形式描述,提出了活动多实例控制体 Shell,用于控制活动多实例的分配和提交.Shell可以根据不同的活动语义,处理多实例的同步并控制整个过程的运行.Shell的提出解决了工作流执行中一个活动多个执行实例的同步执行问题.

关键词: 工作流;工作流管理系统;活动多实例;控制体;同步

中图法分类号: TP311 **文献标识码**: A

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA113150 (国家高技术研究发展计划(863))

作者简介: **孙瑞志**(1964－),男,山东莱州人,博士,副教授,主要研究领域为计算机网络,计算机支持的协同工作;**史美林**(1938－),男,教授,博士生导师,主要研究领域为计算机网络,计算机支持的协同工作.

工作流活动的多实例是指多个参与者共同完成一个活动,活动多实例控制是工作流进行任务分配时经常遇到的问题,定义一个多实例活动可以极大地简化工作流程的定义,它可以从一个整体的角度去表示群体执行同样任务的情况.但这种表示上的灵活性对工作流执行中的调度分配、任务的结束和资源共享等一系列问题提出了控制要求,本文主要研究活动多实例的执行调度问题,在研究了不同活动语义的上下文之后,提出了一个活动多实例的控制体,解决了工作流执行中的上下活动执行转移问题.

1 相关的研究工作

工作流系统中,多实例不是一个复杂的概念,相关的概念有批处理式任务[1]、多任务[2]等.文献[3]在论述工作流的控制结构时,对多实例控制模式,根据多实例数目被确定的时机将其进行了分类:在设计阶段可以确定的、在运行阶段可以确定的和直到生成多实例时才能确定的共 3 种类别.文献[4]分析了工作流中多实例被触发的情况,从消息传递的角度将触发的模式分为同步/异步消息(messaging)触发、终止多实例流程,并采用独占方式(exclusive processes)来保证一个实例不与其他实例产生访问资源冲突.文献[5]从实体-关系图的角度出发,为了处理实体之间的多对多关系,提出了 proclet 的概念,一个 proclet 是一小段过程,多对多的关系可以通过 proclets 之间的多次消息(performative)触发来生成多实例.文献[6]在工作流建模中将活动的多实例作为一个要素加以了考虑.文献[7]研究了多工作流的合并机制,从工作流程间的相互关系出发,强调了多个过程的活动之间时间顺序上的协调.

在多实例的具体实现上,部分工作流产品实现了对多实例的支持.工作流产品 Forte(forte software)和 Verve 等利用循环和并行分支实现了多实例,Visual WorkFlo(FileNet)和 I-Flow(Fujitsu software)用子流程的方式实现了多实例.IBM 的 FlowMark 采用 Bundle 方式解决同步问题,即得到活动的实例化数目后,就由 Bundle 来创建一定的实例数目[3,5].本文从任务-用户的关系出发,研究了一个或密切相关的几个活动被多个用户执行的情况,提出了活动多实例控制体 Shell 和相应的控制算法,控制多个任务的同步和流程的执行.

2 活动多实例问题描述

2.1 活动多实例不同的表现形式

下面用图 1 来说明活动多实例问题,图 1 是一个工作流,由 4 个活动组成,不失一般性,将 4 个活动都看成是一般的原子活动[8].下面我们先给出几个相关概念.

定义 1. 工作流过程定义 W 中的一个活动 n 可以用一个 4 元组$\langle ID,U,T,V\rangle$表示,其中:ID 是活动 ID,U 是该活动所有执行者的集合,T 是活动属性的集合,V 是属性的值域.

定义 2. 一个过程实例 I 中的活动实例 a 是一个四元组$\langle id,n,u,t\rangle$,其中 id 是活动实例 ID,n 是活动实例 a 对应在过程定义 W 中的活动定义,$u\in U$ 是实际的执行者,$t:T\rightarrow V$ 是属性 T 的具体取值,表示活动实例的状态.

定义 3. 活动 n 被称为是多实例的,如果在同一个过程实例 I 中至少存在 2 个活动实例 $a_1=\langle id_1,n,u_1,t_1\rangle$和 $a_2=\langle id_2,n,u2,t_2\rangle$,$a_1,a_2$ 有共同的活动定义 $n=\langle ID,U,T,V\rangle$,且满足条件:$id_1\neq id_2$.

图 1 过程的各个活动多实例的执行情况,根据不同的语义,可以有如下几种:

(1) 若过程定义中没有活动节点是多实例的,则实际执行的轨迹类似图 1.

(2) 若活动 B 是一个多实例(2 个)的节点,则实际执行的过程实例轨迹如图 2 所示.

(3) 若活动 B 和 C 都是多实例,则一种可能的执行轨迹如图 3 所示.其中符号 O 表示之前的多实例先汇聚,后继活动又是一个多实例,符号 O 被称为虚拟汇聚点.

(4) 若活动 B 和 C 形成一个子流程多实例,则实际执行轨迹如图 4 所示.

(5) 若活动 B,C,D 和其后面的活动(E)点都是多实例,B 活动的实例衍生多个子流程情况,则实际执行轨迹如图 5 所示.

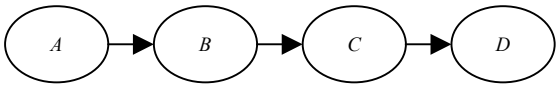

Fig.1 A process sefinition
图 1 一个过程定义

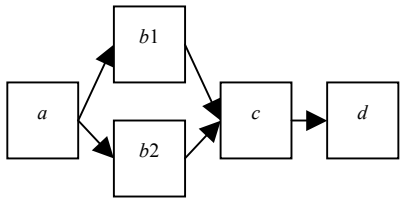

Fig.2 Process trace with 2 instances of activity *B*
图 2 活动 *B* 具有 2 个实例的执行轨迹

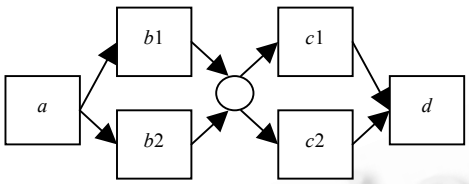

Fig.3 Activity *B* and *C* are multi-instances and they join respectively
图 3 *B*,*C* 是多实例活动且各自要汇聚

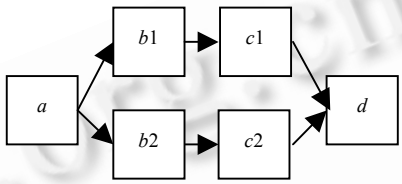

Fig.4 Two sub-processes include instances of activity *B* and *C*
图 4 活动 *B*,*C* 的实例形成 2 个子流程

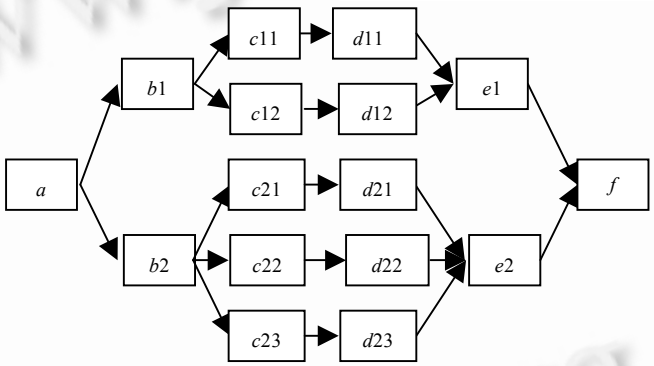

Fig.5 The instances of activity *B* derive some sub-processes
图 5 *B* 活动实例衍生多个子流程

(6) 还有其他的执行轨迹,不再一一列出.

在以上的例子中,仅表示了前一个活动的全部实例都完成提交后,才触发下一个活动的实例,实际上还存在一种情况,就是前序活动的部分实例完成就可触发下一活动.前后活动的触发关系,可以归纳为以下 3 种.

定义 4. 活动 n 的多个实例同步汇聚被称为是 OR-JOIN 的,如果至少有 1 个活动 n 的实例 $a=\langle id,n,u,t\rangle$完成后,$n$ 的直接后继活动被实例化.

定义 5. 活动 n 的多个实例同步汇聚被称为是 ALL-JOIN 的,如果 n 的全部 m 个活动实例 $a_k=\langle id_k,n,u_k,t_k\rangle(k=1,m)$都完成后,才使得直接后继活动被实例化.

定义 6. 活动 n 的多个实例之间被称为是 AND-JOIN 的,如果 n 的全部 m 个活动实例 $a_k=\langle id_k,n,u_k,t_k\rangle(k=1,m)$有 j 个实例$(1<j<m)$,完成后就使直接后继被实例化.

这几个定义说明了同一活动的多实例的同步汇聚情况.除了同一活动的多实例汇聚情况以外,还存在多实例不需要汇聚的情况以及前序活动的一个实例触发后继活动的多个实例(split)等情况.因此,前后活动实例的 Split 方式可以分为 1-1,1-n,n-1,n-n 等 4 种情况.

2.2 多实例节点的描述

对于一个多实例活动,由于在工作流执行过程中会产生各种不同的执行轨迹,就需要工作流引擎在对该活

动进行实例化和控制多实例任务提交的同步方面有一定的控制机制,使得工作流引擎在实例化一个多实例活动时,能够根据活动语义产生按人们意愿执行的结果.就必须在过程定义时能够明确无误地将某些信息传递给工作流管理系统.下面我们来分析,需要传递的信息都应该包括什么.

首先,工作流引擎必须知道如何结束一个多实例的活动,即多实例的同步问题.如前所述,多实例的同步方式有 OR-JOIN,AND-JOIN 和 ALL-JOIN 这 3 种,可以用变量 Max_*num* 和 *Threshold* 来表示,Max_*num* 表示达到全部提交的这个最大数,*Threshold* 表示达到一定阈值要求的数字,它小于最大值.

第 2,多个活动实例在同步汇聚(join)时,汇聚的依据是什么.例如:图 5 中,*d*11 和 *d*21 是不能汇聚的,但 *d*11 和 *d*12 要汇聚,它们有共同的前序点 *b*1,这个汇聚特征用变量 *Join_ref* 来表示.

第 3,后继活动的实例化条件.最简单情况下,前序一个活动实例,触发所有的后继活动实例,即 Split 方式的 1-1 和 1-*n* 关系,但在 *n*-1 和 *n*-*n* 方式中,后继节点的实例需要前序的多个实例都完成后才触发.而仅从前序活动的 Join 方式还决定不了是否可以触发后继活动,如图 5 中活动 *f* 的触发,前序活动 *E* 的两个实例是分别从不同路径被触发的.为了记录后继活动节点被触发时前序活动实例应提交完成的数目,设置计数变量 *Trigger*,*Trigger*=1,表示 1-1 和 1-*n* 关系,*Trigger*=*n*,表示 *n*-1 和 *n*-*n* 关系.

综上所述,有如下定义.

定义 7. 一个多实例活动的上下文 *C* 可以用一个 4 元组〈*Trigger*,*Join_ref*,Max_*num*,*Threshold*〉来描述.

则图 5 中的活动上下文可以描述如下(这里将 *Threshold* 的取值等同于 Max_*num* 的值,即按 ALL-JOIN 方式汇聚).

B:〈1,NULL,1,1〉;前序活动 *A* 的一个实例完成就触发 *B*,Null 表示各实例无须汇聚,*B* 虽有两个活动实例,但各自完成后,就触发后继活动.

C:〈1,NULL,1,1〉;*C* 虽然是多实例的,但从前序活动实例触发它的方式看来,只需 *B* 的一个实例触发.因此,*Trigger*=1,*C* 的实例也不需要汇聚.

D:〈1,NULL,*num*(*g*),*num*(*g*)〉;*D* 要汇聚,汇聚数目是 *num*(*g*),*num*(*g*)表示一个计算函数,如,表示一个部门的全体成员数,需要从一个组织部门 *g* 中计算.

E:〈*num*(*g*),“*B*”,1,1〉;*E* 的两个实例要分别等前序 *D* 的两组实例,每组不同的 *num*(*g*)个实例都完成 *E* 才被实例化,每组实例汇聚的依据是 *B* 活动的实例.

F:〈2,“*A*”,1,1〉;*T* 要等 *E* 的两个实例都完成,它们汇聚的依据是 *A* 活动的实例.

3 多实例同步调度

3.1 控制机制描述

工作流引擎在解释执行一个活动时,可以分成 3 个阶段,即创建活动实例、分发工作项给用户、接收用户提交的任务并激活下一个活动实例.在这个过程中分别有 3 种不同的状态:被初始化、激活和完成[8].一个多实例活动,也有这 3 个阶段,但何时激活后继活动,由于多实例间存在同步问题,需要一定的算法来保证流程的正确执行.

另外,一个多实例活动,在被初始化后,除了要维护几种不同的状态变化以外,还必须控制多实例的同步信息以及何时触发后继活动.一个可多实例化的活动被它的前序活动实例触发而进入执行状态后,在该活动节点上需要做到:(1) 每个符合条件的执行者登录到系统时都能够得到这个活动产生的一个实例(即工作项).(2) 用户提交的工作项达到一定数目时,将活动状态置为“完成”.为做到这两点,就需要一个控制体来维护和控制这些信息,我们将这个控制体称为 Shell.

定义 8. 若 *A*,*B* 是过程定义中具有前后直接触发关系的多实例活动,记为 $A\rightarrow B$.过程运行中分别产生的活动实例 *a*,*b* 之间的触发关系记为 $a\rightarrow b$.

定义 9. 在一个过程实例 *I* 的执行过程中,若存在 $A\rightarrow B$,对于前序活动 *A* 的所有活动实例 $a=\{(a_{11},a_{12},\ldots,a_{1n}),(a_{21},\ldots,a_{2n}),\ldots,(a_{n1},\ldots,a_{nn})\}$,若它们不汇聚,则 $a_{ij}\rightarrow\{b_{ij1},b_{ij2},\ldots,b_{ijn}\}$;若它们汇聚,则$(a_{i1},a_{i2},\ldots,a_{in})\rightarrow\{b_{i1},b_{i2},\ldots,b_{in}\}$或

者 $a\rightarrow\{b_1,b_2,\ldots,b_n\}$.我们将用于产生并控制一组活动实例$\{b_{ij1},b_{ij2},\ldots,b_{ijn}\}$或$\{b_{i1},b_{i2},\ldots,b_{in}\}$或$\{b_1,b_2,\ldots,b_n\}$的一个控制体称之为活动的多实例 Shell.

定义 10. 由同一个 Shell 产生的多个活动实例称为兄弟实例.

为了控制多实例产生与同步,Shell 必须维护必要的数据信息,可以用一个 8 元组⟨*ID*,*Path*,*Trigger*,*Join_ref*,Max_*num*,*Threshold*,*Submit*,*Status*⟩来表示,其中:

ID 标志一个 Shell 控制体.

Path 是过程实例执行的轨迹,记录从起始活动到该活动,中间有哪些活动实例被执行.

Trigger,*Join_ref*,Max_*num* 和 *Threshold* 含义同定义 7.

Submit:记录已提交的实例数.

Status:记录 Shell 的状态,Shell 有 4 种状态:waiting,valid,finished 和 full.waiting 表示还不能工作,是一个控制体产生时的初始状态.valid 表示能够产生活动实例.finished 表示完成任务,不再处理提交的工作项.full 表示已分配到最大数目的工作项,full 状态可以限制工作项不能再被分配.

3.2 控制算法

为描述方便,从一个用户提交一个活动实例的工作项开始讨论控制算法,即当前的活动的 Shell 已经产生,处在有效状态,分配了工作项给用户.用户提交时,可以找到所提交的活动实例对应的 Shell.控制算法所做的工作就是如何修改当前的 Shell 状态,判断多个实例的汇聚,并产生后继活动的 Shell.

设活动 *a* 的 Shell G_a 生成的一个实例被提交处理,活动 *b* 是过程定义中 *a* 的后继活动,我们用 $G_a.id$,$G_a.path$ 等形式分别表示 G_a 维护的 8 元组 *ID*,*Path*,*Trigger*,*Join_ref*,Max_*num*,*Threshold*,*Submit*,*Status* 的取值,用 *b.trigger* 等形式表示活动 *b* 的上下文属性 *Trigger*,*Join_ref*,Max_*num* 和 *Threshold*.

算法 1. 处理提交的活动实例,并产生后继活动的 Shell.

(1) IF $G_a.status$="finished" RETURN(FALSE); //活动已完成则忽略提交的工作项并返回.

(2) $G_a.submit\leftarrow G_a.submit+1$;

(3) IF $G_a.submit=G_a.threshold$

THEN $G_a.status\leftarrow$"finished"; //提交数到达 *Threshold* 时,Shell 完成使命.

(4) IF (*b.join_ref*=NULL) OR NOT *Exist_shell*(*b*)

THEN

$G_b\leftarrow$*Create_shell*(*b*); //不需要在后继活动 *b* 汇聚,或后继活动 *b* 的 Shell 不存在,创建活动 *b* 的 Shell G_b

$G_b.path\leftarrow G_a.path+a.id+b.id$; //将 *a* 活动实例 *ID* 和 *b* 的活动 *ID* 加到流程执行路径上,传给 G_b.

$G_b.trigger\leftarrow G_b.trigger-1$;

IF $G_b.trigger$=0

THEN $G_b.status\leftarrow$"valid" //置活动 *b* 的 Shell 有效.

ELSE $G_b.status\leftarrow$"waiting";初始化 G_b

(5) ELSE //*b* 的 Shell 已由兄弟实例创建

$G_b.trigger\leftarrow G_b.trigger-1$;

IF $G_b.trigger$=0

THEN $G_b.status\leftarrow$"valid"

在算法 1 中判定后继活动 Shell 是否存在,有:

算法 2. *Exist_shell*(*b*) //判定活动 *b* 的相应 Shell 是否存在.

DO

(1) 搜索活动 *b* 的 *Shell* G_b;

(2) IF NOT FOUND THEN RETURN(FALSE);

(3) $Path\leftarrow G_b.path$;//若找到一个 Shell,就将其 *path* 值赋给变量 *Path*;

(4) *Split_activity*←G_b.*join_ref*; //赋值给变量 *Split_activity*;

(5) *Location_b*←*Split_activity* IN *Path*; //在 G_b 的执行路径上中找到其汇聚依据活动的位置;

(6) *Split_instance_b*←*Path*[*location_b*+1]; //找到汇聚依据活动的实例;

(7) *Location_a*←*Split_activity* IN G_a.*path*;//汇聚依据在前序 Shell 执行路径上的位置;

(8) *Split_instance_a*←G_a.*path*[*Location_a*+1];//找到前序活动实例的汇聚依据;

WHILE *Split_instance_b*≠*Split_instance_a*;//找到的 Shell 记录的汇聚依据与 *a* 实例的汇聚依据不同,继续循环;

(9) RETURN(TRUE).

3.3 同步善后处理

如果一个活动的多实例是 OR-JOIN 或 AND-JOIN 时,在满足提交的实例数目后,后继活动就开始执行了,不再等待另外的活动实例.如何处理这些“迟到”的实例,就是多实例同步的善后处理问题.善后处理策略可以有两种,即工作流引擎可以向其他兄弟实例“通报”或“不通报”.所谓通报,就是通过一种方式告知其他实例不必再继续执行.不通报,则还在执行的活动实例,直到提交时,才得知提交的实例已“过期”.

算法 1 中,采用的是一种“不通报”处理策略,当多个实例达到触发数后,将 Shell 的状态置为 finished,到达的兄弟实例提交时,Shell 若处于 finished 状态,则提交的实例就被忽略.

3.4 多实例工作流举例

工作流程中多实例的情况经常可见,例如,某单位发布一个信息调查表,由办公室秘书下发各部门经理(假设有 3 个部门),部门经理下发部门员工(各部门员工数不一样),各员工返回部门经理,考虑到有员工出差等情况,调查表人数达 80%即可,最后汇总到办公室秘书.其工作流程图如图 1 所示,但有 5 个活动结点.该流程中有 3 个活动是多实例的,即 *B*(部门经理下发)、*C*(部门员工填写)、*D*(部门经理汇集),其执行轨迹类似图 5,*B* 有 3 个实例,但更简单一些,没有子流程出现.根据定义 7,可以进行以下的形式描述(忽略活动的其他基本属性的描述).

A:⟨1,NULL,1,1⟩,*A* 是只有一个活动实例的活动,*Trigger*,Max_*num*,*Threshold* 都为 1.实例无须汇聚,*Join_ref*=Null.

B:⟨1,NULL,1,1⟩,类似图 5 中的 *B* 结点,但实际执行是 3 个活动实例.

C:⟨1,NULL,*num*(*g*),*num*(*g*)×0.8⟩;类似图 5 中的 *C* 结点,每个部门员工(实例)数不同,用 *num*(*g*)变量表示,实例要汇聚,汇聚数目是 0.8×*num*(*g*),表示 80%的人完成就有效.

D:⟨*num*(*g*)×0.8,“*B*”,1,1⟩,类似图 5 中的 *E* 结点,3 个部门经理的 3 个实例要分别等各自的员工完成后汇集,汇聚的依据是其发起的活动 *B* 的实例.

E:⟨3,“*A*”,1,1⟩;类似图 5 中的 *F* 结点,秘书要等 3 个部门都完成,它们汇聚的依据是其发起的活动 *A* 的实例.

整个执行过程中,以结点 *C* 为例,来分析对内存容量的要求.假设,从活动实例产生到每个员工获得工作项是随机的,在 24 小时内,一个部门的 20 名员工陆续看到该项工作,看到过程符合泊松分布,则根据随机理论得出,平均每人看到该工作项的时间为 1.2 小时.若平均每人完成该工作所用时间为 1 小时.再假设,创建一个 Shell 用内存为 200 字节,创建活动实例用 350 字节,则用 Shell 控制方式下,20 名员工全部执行完占用内存的时空(时间×内存量)为 350×1×20+200×1.2×1(1 个 Shell)=7240,但若采用生成 20 个工作项的方法(如 IBM Bundle),20 名员工占用的时间容量为 350×(1.2+1)×20=15400,Shell 控制方式下完成一个多实例活动可节省 8 160 的时空.

4 与其他相关工作的比较及结论

多实例控制体 Shell 解决一个活动的多实例同步控制问题,通过过程中前后活动节点的上下文描述,可以解决子流程的多实例问题和多实例的多层分支问题.文献[3]虽然提出了多实例的不同类别,但也仅仅进行了分类,并没有对多实例的同步控制进行更深入的分析.本文提出的 Shell 概念与 IBM 的 FlowMark 采用 Bundle 方式有些类似,但 Shell 的实现方法并不像 Bundle 方式那样创建多个实例,而只是记录多实例的状态,在用户登录后,根据状态来创建实例,既节省了存储容量,又体现了灵活性.文献[4]和文献[5]的多实例触发机制实质上是一种消

息触发下的循环控制或多线程,严格意义上讲,这种方式不能做到一个活动的多个实例的同步,因为它是先执行了一个活动实例之后,再触发同样的活动循环或并行执行,它们主要用于解决工作流之间的多次被调用,而本文提出的多实例控制体允许多个用户同时申请工作项,主要解决的是多用户的任务同步问题.文献[6]在研究工作流建模机制中,认识到多实例是一个重要考虑的因素,但没有讨论多实例的具体表现形式,文献[7]的着眼点在于多个不同工作流间的相关活动的协调,与本文讨论的多实例的同步有相似之处,但两者解决的问题不同,其协同控制机制也不同.

工作流活动的多实例有各种不同的表现形式,本文通过分析这些表现形式,对多实例的活动的上下文进行了形式描述,提出了多实例控制体和相应的控制算法,解决了工作流执行系统对多实例的控制与分配.

多实例会引发工作流管理中的许多问题,如,多实例活动的事务保证、具有多实例活动的流程的循环执行和多实例由于共享资源而造成死锁等问题,这也是目前许多商用工作流管理系统不支持多实例的主要原因.我们在解决了控制调度的基础上将进一步研究这些问题.

References:

[1] Barthelmess P, Wainer J. Workflow systems: A few definitions and a few suggestions. In: Comstock N, Ellis CA, eds. Proc. of the Conf. on Organizational Computing Systems—COOCS'95. Milpitas: ACM Press, 1995. 138−147.

[2] Casati F, Ceri S, Pernici B, Pozzi G. Conceptual modeling of workflows. In: Papazoglou MP, ed. Proc. of the OOER 14th Int'l Object-Oriented and Entity-Relationship Modelling Conf. Gold Cost: Springer-Verlag, 1995. 341−354. http://citeseer.ist.psu.edu/casati95conceptual.html

[3] van der Aalst WMP, ter Hofstede AHM, Kiepuszewsk KB, Barros AP. Workflow patterns. BETA Working Paper Series, Working Paper 47. Eindhoven: Eindhoven University of Technology, 2000.

[4] Barros AP, ter Hofstede AHM. Modeling extensions for concurrent workflow coordination. In: Proc. of the 4th IFCIS Int'l Conf. on Cooperative Information Systems, CoopIS'99. Edinburgh: IEEE Computer Society, 1999. 336−347. http://citeseer.ist.psu.edu/barros99modelling.html

[5] van der Aalst WMP, Barthelmess P, Ellis CA, Wainer J. Proclets: A framework for lightweight interacting workflow processes. Int'l Journal of Cooperative Information System, 2001,10(4):443−481. http://tmitwww.tm.tue.nl/staff/wvdaalst/Publications/p140.pdf

[6] Fan YS , Wu C. Research on workflow modeling method to improve system felxibility. Journal of Software, 2002,13(4):833−839 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/833.pdf

[7] Liu XQ, Wang XL, Zeng GZ. Multi-Workflow combining method based on coordination mechanisms. Computer Engineering, 2003, 29(2):118−119 (in Chinese with English abstract).

[8] Workflow Management Coalition. Terminology & Glossary. Technical Report, WFMC-TC-1011, 1999.

附中文参考文献:

[6] 范玉顺,吴澄.一种提高系统柔性的工作流建模方法研究.软件学报,2002,13(4):833−839. http://www.jos.org.cn/1000-9825/13/833.pdf

[7] 刘向前,王晓琳,曾广周.基于协调机制的多工作流过程合并方法.计算机工程,2003,29(2):118−119.