面向资源的工作流事务模型*)

Resource-Oriented Transaction Model for Workflow Management System

赖耀东[1] 朱建新[2] 高 济[1]

(浙江大学人工智能研究所 杭州310027)[1]

(浙江师范大学计算机科学与工程学院 浙江金华321004)[2]

Abstract It's the reason for current Workflow Management System having not provided transaction model and the support of data recovery and exception handling that there is no direct transforming between activity-oriented workflow model and data-oriented transaction model. This paper proposes an idea of implementing resource-centric transaction model for WFMS: Integrate the plug-in container of resource in former architecture, separate resource scheduling as independent functional module from skill module, and imply transaction planning in agent collaboration, without losing flexibility and interaction ability of former architecture. Thus provide the support of concurrency and recovery for shared data resource in Workflow, which is the basis of data consistency control and exception handling.

Keywords Software architecture, Software agents, Workflow management system, Advanced transaction model, Exception handling

1 概述

计算机辅助的工作流(workflow)可定义为发生于分布、异质系统环境下的一系列程序的调用和数据的交换。近年来出现的大批工作流管理系统(WFMS),包括我们开发的基于多 Agent 虚拟组织框架的工作流管理系统 SaFlow[4],为工作流程的协商、调度、组织、控制、用户接口、监测、分布式、异质等方面提供了强大的支持;从软件工程的角度,作为工作流程的开发和运行的平台,WFMS 必须为工作流程提供异常处理的机制,将失败语义从正常的程序运行逻辑中分离出来[2]。然而,这些系统都没有提供事务概念上的流程管理,缺乏在异常和失败发生的情况下保证数据正确性和依赖性的能力。

高级事务模型(Advanced Transaction Model, ATM)[1]基于数据库的事务概念和理论,为数据一致性控制和数据恢复提供了一套定义严密的失败语义。在 WFMS 中应用 ATM,其基本思想是将传统的数据库事务作为搭建工作流的基本单位,并为成功提交的事务提供补偿事务,以块(block)的形式实现活动的原子性和可撤消性;然而过分以数据库为中心,限制了基于 ATM 的 WFMS 在实际环境下的适应能力,也大大降低了 WFMS 的灵活性。

本文提出了以广义的数据资源为中心实现工作流程事务模型的构想:在不影响 SaFlow 原有框架的灵活性和交互能力的前提下,将广义资源包容器集成到框架,将事务模型和资源控制隐含于复合活动的调度之中。为工作流程中活动的共享资源提供并发和恢复的支持,给系统的数据一致性控制和异常机制打下良好的基础。

2 广义的数据资源

当前的 WFMS 集中解决工作流活动之间的协调和合作;而 ATM 是从数据库的事务概念和原理发展而来。面向活动的工作流模型和面向数据的事务模型无法进行直接的互换,这也是当前的 WFMS 无法提供对数据恢复和失败处理的有力支持的原因。

为实现工作流模型和事务模型之间的互换,一些折中的做法是:传统意义上的事务流程作为组成活动的子单元,并为每个事务流程提供对应的补偿事务,以实现工作流活动的补偿恢复(linear saga)[1];或者,为传统事务模型增加环境上下文的交互能力(flexible transaction)[1]。然而,这些互换只能在有限的条件下进行。而且,互换以互相牺牲性能为代价。由此可见,要在 WFMS 中实现事务模型,必须在 WFMS 的框架中加入面向数据资源的特性,并在建模语言中提供显式的、独立的数据资源描述。

分析工作流的处理的数据可以发现,有些数据被若干活动共享,并作为运行结果永久保存;有些只作为中间量不做保存,随工作流运行结束而失效。WFMS 只需要对前者保证其一致性和正确性,后者不会影响工作流的运行环境。工作流通过领域层操作,将数据处理隐含于操作过程中,数据不局限于数据库数据,领域操作所涉及到的一切文档、数据库记录、数据场、文件、甚至可执行代码,都可以成为领域操作的对象。为此,我们提出引入广义资源概念,即领域操作中一切可影响工作流运行环境的领域操作对象。广义资源的访问隐含于领域操作中,不仅导致了语义上的模糊,使资源访问缺乏统一的调用接口,而且系统也无法灵活高效地控制资源的并发和调度,从而无法实现真正事务意义上的工作流模型。我们构想:在原有框架基础上,增加共享资源的包容和合成,以及资源并发和事务模型的清晰描述,将资源访问和领域操作分离,为数据恢复和失败处理提供框架上的支持。

3 广义资源的包容和控制结构

3.1 资源的包容

工作流程中共享的、对环境产生物理影响的文档、数据库记录、数据场、文件、甚至可执行代码,都可以作为资源,在工

*)国家自然科学基金资助项目(69773019)。赖耀东 硕士生,主要研究方向为人工智能和软件工程。朱建新 讲师,主要研究方向为计算机网络。高 济 教授,博士生导师,主要研究方向为人工智能、软件工程和 CIMS。

作流建模中显式地描述。资源 Resource 可定义为8元组(Data-Description, Data-Buffer, Input-Arg-set, Output-Arg-Set, Write-Process, Read-Process, Cancel-Process, Update-Process),其中:

Data-Description:描述资源对象所抽象的数据的可用信息。

Data-Buffer:用于资源存取操作的缓冲区域。

Input-Arg-Set:资源写操作的输入参数集。

Output-Arg-Set:资源读操作的输出参数集。

Write-Process:资源写操作,将输入参数集对资源数据的影响反应到 Data-Buffer 中。

Read-Process:资源读操作,将 Data-Buffer 保存的资源数据反应到输出参数集中。

Cancel-Process:撤消操作,撤消资源初始化以后对 Data-Buffer 的一切修改。

Update-Process:更新操作,将 Data-Buffer 中保存的修改提交到资源数据中。

原有框架为包容领域操作,用微构件模型定义构件接口和接口提供的领域操作,由系统协作合成语言编译器编译为 COM 或 CORBA 规范中的构件调用。对资源的读、写、撤消和更新过程作为领域操作,得到原框架基于微构件领域操作调用模式的良好支持。Data-Buffer 作为资源操作的缓冲区域,拥有可同资源数据存储结构等效互换的物理存储结构。资源的初始化过程中,Data-Buffer 将得到资源数据的副本,资源提交前的一切修改都将在 Data-Buffer 中进行,并保存一个以活动为索引的操作日志。

资源模型的描述如下:

```
Resource〈名称〉
  Date-Description:〈概念模型〉
  Input:{〈type〉〈参数名〉,}*;
  Output:{〈type〉〈参数名〉,}*;
  Operator-Set:{(〈领域操作名〉〈构件名〉.〈接口名〉.
                〈函数名〉)}*
  Write-Process:{〈函数表达式〉|〈规则〉|〈规则组〉}*;
  Read-Process:{〈函数表达式〉|〈规则〉|〈规则组〉}*;
  Cancel-Process:{〈函数表达式〉|〈规则〉|〈规则组〉}*;
  Update-Process:{〈函数表达式〉|〈规则〉|〈规则组〉}*;
End [〈名称〉]
```

3.2 资源访问的合成

资源访问层作为 Agent[3] 行为控制器(Behavior-Controller)的层次之一,用以启动面向基本活动的资源访问。资源访问的合成关心的是如何协调地聚合相关的资源存取操作,实现基本活动的资源访问的目标,其定义构成资源访问规则的透明描述。为提供强大的交互能力,我们将资源的访问作为 Agent 技能层领域操作集合的一部分,由技能合成提供资源访问操作的配置和组合。原框架技能合成提供两种方式的规划:状态-反应规则组和任务规约模型,实现了资源访问的弹性和上下文交互能力。

技能合成的结构如图1。

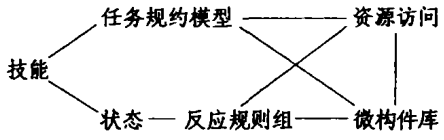

图1

资源访问模型定义了资源访问的所操作的资源名称和操作类型,其 BNF 表示如下:

```
ResourceOperator〈资源访问名〉
{ ResourceName:〈资源名〉
  OperateType:〈访问方式〉
  Parameter:{〈type〉〈参数名〉,}*
  [Description :〈Text 〉]
  [Pre-Activity:〈活动名〉][Post-Activity:〈活动名〉] }+
```

假设 Agent 模型中定义了一资源模型 Re_Test,在技能模型 S_Test 中合成了对此活动的一次写访问,此访问必须在活动 A_Pre 执行成功之后才可以进行,技能 S_Test 由基本活动 A_Test 我们有如下的定义:

```
Resource Re_Test
  Date-Description: Concept_Test;
    //一个用于描述资源的可用信息的概念模型
  Input: 参数1,参数2,……参数 N;
  Output: 参数1,参数2,……参数 N;
  Write-Process: … Read-Process: …Cancel-Process: … Update-
Process: …
    //以上四个过程由微构件调用实现……
  End Re_Test
  Skill S_Test
    Description: ;
    Activity: A_Test;
    Performance;
      OperationSeries
        ……
        (ResourceName: Re_Test
        OperateType: W
        Parameter: 参数1,参数2,……参数 N
        Description: A_Test 合成的对 Re_Test 的写操作
        Pre-Activity: A_Rre)
        ……
  End S_Test
```

3.3 资源调度

资源调度 Resource scheduler 是由基本活动发起的对资源的存取操作,它可表示为一个6元组(Status, Task-Locker, Activity-Locker, Processing-Activity, Pre-Activity-Set, Post-Activity-Set),其中:

Status:当前活动对资源的调度状态,有等待态、访问态和完成态。

Task-Locker:顶层活动用以锁定资源,实现不同工作流程之间对资源的互斥操作。

Activity-Locker:基本活动对资源的读写锁,用以同 Agent 上不同基本活动对资源的互斥操作。根据活动对资源不同的读写要求,应将 Activity-Locker 设置为 W(写)、R(读)。W 排斥一切读、写要求,R 不排斥新的读要求,但排斥写要求。

Processing-Activity:资源当前的调度活动。

Pre-Activity-Set:资源当前调度的先期活动集。

Post-Activity-Set:资源当前调度的后期活动集。

资源当前调度的先期和后期访问集可以在资源访问操作中显式地描述,也可以在调度过程中动态地生成,用以指定资源访问的顺序依赖关系。

软件 Agent 作为包容体系结构上层的基本构件,其不仅需要描述一组由该 Agent 调度和监控的资源,管理资源调度的共有信息,而且需要显式地描述该 Agent 用以满足基本活动资源访问要求的访问合成。基本活动发起对资源的存取操作以后,需要管理一个资源调度的列表,监控活动当前对资源的调度状态。基本活动上的资源调度状态引用 Agent 上的资源调度对所有活动的共有状态。

软件 Agent 初始化的过程中,将根据 Agent 自身模型中定义的资源模型初始化一个可用资源实例的列表。任何它所调度的基本活动提出对资源列表中资源的访问要求,Agent 将检查资源列表,集中调度各活动资源访问的要求,具体操作如下:

	Agent 调度状态	A 活动调度状态	B 活动调度状态	C 活动调度状态
初始态	Task-Locker: No Activity-Locker: No Processing-Activity: None			
A 活动读请求	Task-Locker: 1 Activity-Locker: R Processing-Activity: A	Pre-Activity-Set: None Post-Activity-Set: None Status: Running		
C 活动读请求，先期活动为 B	Task-Locker: 1 Activity-Locker: R Processing-Activity: A	Pre-Activity-Set: None Post-Activity-Set: None Status: Running		Pre-Activity-Set: B Post-Activity-Set: None Status: Waiting
B 活动写请求	Task-Locker: 1 Activity-Locker: R Processing-Activity: A	Pre-Activity-Set: None Post-Activity-Set: B Status: Running	Pre-Activity-Set: None Post-Activity-Set: C Status: Waiting	Pre-Activity-Set: B Post-Activity-Set: None Status: Waiting
\A 活动完成，资源重新调度	Task-Locker: 1 Activity-Locker: W Processing-Activity: B	Pre-Activity-Set: None Post-Activity-Set: B Status: Complete	Pre-Activity-Set: A Post-Activity-Set: C Status: Running	Pre-Activity-Set: B Post-Activity-Set: None Status: Waiting
B 活动完成，资源重新调度	Task-Locker: 1 Activity-Locker: R Processing-Activity: C	Pre-Activity-Set: None Post-Activity-Set: B Status: Complete	Pre-Activity-Set: A Post-Activity-Set: C Status: Complete	Pre-Activity-Set: B Post-Activity-Set: None Status: Running
C 活动完成	Task-Locker: 1 Activity-Locker: No Processing-Activity: None	Pre-Activity-Set: None Post-Activity-Set: B Status: Complete	Pre-Activity-Set: A Post-Activity-Set: C Status: Complete	Pre-Activity-Set: B Post-Activity-Set: None Status: Complete
工作流程1提交	Task-Locker: No Activity-Locker: No Processing-Activity: None			

如资源处于未上锁状态，则根据本活动所属的工作流程号锁定 Task-Locker，根据活动的访问要求以 W/R 锁定 Activity-Locker，填写 Processing-Activity 为本活动；将资源调度状态的副本链接到本活动的调度列表中并填写先期活动集、后期活动集和状态。

如资源被同工作流程中的其他活动锁定，则根据读、写的互斥要求以及先期活动集以及等待活动集所描述的访问顺序的依赖关系决定其访问要求是否遭到排斥。将资源调度状态的副本链接到本活动的调度列表中，调整相关活动的先期和后期活动集，保持资源访问依赖关系的正确性。

如资源被非本活动所属的工作流程锁定，则拒绝访问要求。

如资源被提交或撤消，则解除 Task-Locker 上的锁定，清除相关活动的资源列表。

一个可能的调度如下：如在 Agent 模型中有关于某资源 Re_Test 的描述，则 Agent 初始化过程中将把此资源加入到自身的资源列表中。接下来顺次有此 Agent 调度的三个基本活动 A，C，B 对此资源提出了访问要求，假定此三个活动都属于任务1，并且它们所调用的技能分别包含如下的资源访问调用：

A：(ResourceName：Re_Test OperateType：R ……)

C：(ResourceName：Re_Test OperateType：R Pre-Activity：B)

B：(ResourceName：Re_Test OperateType：W ……)

用一个调度表说明在这样的情况下系统状态变化过程如上表所示。

3.4 动态规划的事务模型

做了以上的工作，我们已经把资源抽象并从领域操作的语义中完全分离出来。事务模型将不再考虑对工作流运行环境不产生影响的领域操作，事务的撤消、回滚、提交等操作只以资源为对象。工作流运行于多 Agent 的虚拟框架上，由 Agent 集中管理本地资源。工作流程由一个软件 Agent 发起顶层复合活动开始，通过复合活动的调度规则层层分解，由底层的基本活动完成技能层的调用。技能层合成了一组领域操作，以及资源的访问操作。技能层完成对资源访问操作的调用之后，不再控制其操作细节；访问操作将生成一个资源的调度实例，由 Agent 的资源调度模块统一控制(如图2)。调度实例包含了资源访问相关于工作流实例、活动和依赖关系的说明信息，实际上，工作流运行产生的所有调度实例的集合，组成了此工作流的事务模型，调度实例集合的生成隐含于工作流的运行过程之中，即事务模型的动态规划过程。资源的抽象包容，使系统可以根据需要灵活定制并发的粒度；技能层和资源调度层的分离，使事务控制不会影响工作流活动规划的灵活性；事务模型动态规划于工作流运行过程中，更增强了事务模型的交互能力。

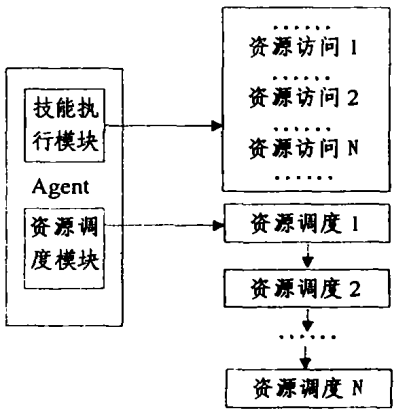

图2

我们增加了活动(包括复合活动和基本活动)的运行状态：未完成态、完成态和提交态，来实现事务模型的控制；软件

(下转第64页)

符号序列的翻转距离，由 S. Hannenhalli 等给出的计算有向符号序列翻转距离的算法时间复杂度为 $O(n^5)$，因此算法 A_{ST} 的最坏时间复杂度为 $O(m^2n^5)$。

5.2 F3ST 问题的近似算法

F3ST 问题的优化形式为：给定三条长度为 n 的有向符号序列 π^1,π^2,π^3，集合 $S=\{1,2,\cdots,n\}$ 的一个真子集 L，求一条有向符号序列 $\pi^*=\pi_1\pi_2\cdots\pi_n$，使得 $\sum_{i=1}^{3}d(\pi^i,\pi^*)=\min\sum_{i=1}^{3}d(\pi^i,\pi^V)$，而且 $\pi_i=i,i\in L,|\pi_i|=i,i\in S-L$。

近似算法 A：设计序列 $\pi^A=\pi_1\pi_2\cdots\pi_n$，取 $\pi_i=i,i\in L$；对其余的符号从小到大设置其方向，对每一个未确定方向的符号 i，

1）如果 i 与 $i+1$ 在 π^1 中相邻并以 $i,i+1$ 的顺序出现，且 $i+1$ 方向未定，取 $|\pi_i|=i$、$|\pi_{i+1}|=i+1$，π_i、π_{i+1} 的方向分别与 i、$i+1$ 在 π^1 中方向相同；

2）如果 i 与 $i+1$ 在 π^1 中相邻并以 $i+1,i$ 的顺序出现，且 $i+1$ 方向未定，取 $|\pi_i|=i$，、$|\pi_{i+1}|=i+1$，π_i、π_{i+1} 的方向分别与 i、$i+1$ 在 π^1 中方向相反；

3）如果 i 与 $i+1$ 在 π^1 中不相邻，或 i 与 $i+1$ 在 π^1 中相邻但 $i+1$ 方向已定，则任意指定 i 方向；

全部符号方向取定后输出 π^A 及 $d^A=\sum_{i=1}^{3}d(\pi^i,\pi^A)$。

对 F3ST 问题的实例 Π，算法 A 输出的 π^A 及 $A(\Pi)=\sum_{i=1}^{3}d(\pi^i,\pi^A)$ 能够使得 $A(\Pi)/OPT(\Pi)\leqslant 8$。设 π^* 为最优解，π^{1*} 为满足 $\pi_i=i,i\in L,|\pi_i|=i,i\in S-L$ 的与 π^1 最近的序列，则只要说明下式

$$\begin{aligned}A(\Pi)&=\sum_{i=1}^{3}d(\pi^i,\pi^A)\leqslant d(\pi^1,\pi^A)+(d(\pi^2,\pi^1)+d((\pi^1,\pi^A))+(d(\pi^3,\pi^1)+d(\pi^1,\pi^A))\\&\leqslant 3d(\pi^1,\pi^A)+(d(\pi^2,\pi^1)+d(\pi^3,\pi^1)+d(\pi^3))\\&\leqslant 6d(\pi^1,\pi^{1*})+2\sum_{i=1}^{3}d(\pi^i,\pi^*)\leqslant 6d(\pi^1,\pi^*)+2\sum_{i=1}^{3}d(\pi^i,\pi^*)\\&\leqslant 8\sum_{i=1}^{3}d(\pi^i,\pi^*)\\&=8OPT(\Pi)\end{aligned}$$

中的 $d(\pi^1,\pi^A)\leqslant 2d(\pi^1,\pi^{1*})$ 即可。算法 A 使得 $G_R(V(S),E(\pi^1),E(\pi^A))$ 中仅含一条黑边的圈与 $G_R(V(S),E(\pi^1),E(\pi^{1*}))$ 中仅含一条黑边的圈相同，假设其余的黑边为 b 条，则 $d(\pi^1,\pi^A)\leqslant b$，$d(\pi^1,\pi^{1*})\geqslant b-c$，其中 c 为 $G_R(V(S),E(\pi^1),E(\pi^{1*}))$ 中至少含两条黑边的圈的个数，由 $c\leqslant b/2$，即得 $d(\pi^1,\pi^A)\leqslant 2d(\pi^1,\pi^{1*})$。

参 考 文 献

1 Sankoff D, Leduc G, Antoine N, Paquin B. Gen order comparisons for phylogenetic inference: Evolution of the mitochondrial . Pro. Natl. Acad. Sci. U. S. A., 1992, 89: 6575～6579

2 Kececioglu J, Sankoff D. Exact and approximation algorithms for the reversal distance between two permutations. Algorithmica, 1995, 13: 180～210

3 Kececioglu J, Sankoff D. Efficient bounds for oriented chromosome inversion distance. In: Proc. 5th Annual symposium on combinatorial pattern matching, Lecture notes in Computer science-807, 1994. 307～325

4 Bafna V, Pevzner P. Sorting by reversals: genome rearrangements in plants organelles and evolutionary history of X chromosome. Mol. Biol. and Evol., 1995, 12: 239～246

5 Hannenhalli S, Pevzner P A. Transforming cabbage into turnip-Polynomial algorithm for sorting signed permutations by reversals. In: Proc. of STOC'95, Las Vegas, USA, 1995. 178～179

6 朱大铭，马绍汉，等. 翻转距离星树问题的复杂度和近似算法. 软件学报录用

（上接第117页）

Agent 管理了一个可由其在本地通过执行基本活动访问的资源列表。基本活动管理了一个用以说明活动本身资源访问状态的调度实例列表。基本活动对资源的任何访问要求，经 Agent 的允许，将 Agent 资源列表里的资源加入到本活动的资源调度列表中，同时由 Agent 修改资源的调度状态（加任务锁和活动锁）。基本活动技能调用完成，则活动进入完成态，等待上层的提交命令；基本活动调用失败，则基本活动进入未完成态，由 Agent 撤消活动资源调度列表中资源的修改，然后重置此资源为未锁定态，最后将活动状态置为失败。顶层复合活动当其所有活动均为完成态，则一次发出提交通知，由基本活动完成其资源调度列表中所有资源的提交，然后重置资源的调度状态，归还资源。

结论和后续工作 本文提出了面向资源的基于多 Agent 虚拟组织框架的扩展，加强了框架资源包容、调度和事务建模的能力，为基于多 Agent 虚拟组织框架的数据流管理系统 SaFlow 提供了事务模型，即异常和失败发生的情况下保证数据正确性和依赖性的基础。

显式的资源定义抽象了资源存取的细节，为系统提供资源存取了统一的方式和清晰的语义，区分了工作流的共享资源和中间数据，并灵活控制了和并发控制的粒度。资源调度作为单独的模块从技能模块中分离，在活动规划中动态生成调度计划，由 Agent 提供独立的调度控制，保持了原有框架任务规划和活动协调的灵活性和交互能力的。资源访问合成于技能之中，实现了资源访问的灵活性以及同领域操作的协作和交互。资源访问缓冲区的引入，以及显式的资源访问依赖性说明，实现了活动执行的提交概念，并为活动的撤消和恢复奠定了基础。

以此为基础，我们将为 SaFlow 研究更加灵活可靠的异常处理机制。

参 考 文 献

1 Alonso C, et al. Advanced Transaction Models in Workflow Contexts [R]: [Technical Report]. IBM Almanden Research Center, 1996

2 Hagen C, Alonso G. Flexible Exception Handling in Process Support Systems [R]: [Technical Report No. 290 ETH Zurich]. Department of Computer Scince, Feb. 1998

3 高济，王进. 基于 Agents 的软件合成框架 ABFSC [J]. 计算机学报，1999, 22(10): 1050～1058

4 骆海波. 一种基于多软件 Agent 的工作流管理系统[D]: [浙江大学硕士学位论文]. 1999. 2