基于反省技术的灵活工作流管理系统

姚绍文[1,2],王敏毅[2],周明天[2]

(1. 云南大学信息学院,云南昆明 650091;2. 电子科技大学计算机学院,四川成都 610054)

摘 要: 面对复杂多样的应用环境,灵活性成为工作流技术研究的重点之一.反省作为提高系统灵活性的重要技术手段也可用于工作流管理系统中.本文分别从结构和行为方面对工作流过程及其管理进行了具体化,进而建立其反省模型.为了方便对元级的抽象和编程操纵,我们提出了元对象协议模型,其中包括协议声明、元对象关联、协议选择和元对象管理等机制或方法.进一步,本文通过典型的应用说明了反省方法对提高工作流系统灵活性和适应性是有效的.最后,文章介绍了工作流管理系统中反省技术的实现方法.

关键词: 工作流; 过程; 工作流管理系统; 反省; 灵活性

中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12A-2040-05

Flexible Workflow Management System Based on Reflection Technology

YAO Shao-wen[1,2], WANG Min-yi[2], ZHOU Ming-tian[2]

(1. *School of Information and Electronic Science, Yunnan University, Kunming, Yunnan* 650091, *China*;
2. *School of Computer Sci. and Eng., University Of Elec. Sci. and Tech., Chengdu, Sichuan* 610054, *China*)

Abstract: In complex and diversified application context, flexibility is one of the key issues of workflow technology. As an important approach to improve the flexibility of system, reflection is also available to Workflow Management System (WfMS). This paper reifies the workflow process and its management respectively from structural and behavioral aspects, and as a result, a reflective model of workflow is designed. A model of Meta-Object Protocol (MOP) is proposed to abstract and program on meta-level, including protocol declaration, meta-object association, protocol selection and meta-object management. The typical instance evolution example shows that reflection is effective to improve the flexibility and adaptability of WfMS. The implementation of reflection is presented in the paper.

Key words: workflow; process; workflow management system; reflection; flexibility

1 引言

作为企业商务过程电子化的关键要素,工作流/过程管理技术被普遍关注.由于工作流管理系统(WfMS)在应用上面临纷繁复杂的局面,不同的 WfMS 的异构互联、标准化、向 Internet 或 Web 计算环境迁移、灵活性/适应性、协作性、智能化等方向成为近年来的研究热点[1,4].针对各种应用环境可能的变化[7,12],有的研究着重于过程模型的灵活性,如抽象过程[12]、动态模型[2]、元模型等;另外一些借助灵活的框架、Agent、分布对象、组件化等技术从整体上保证系统灵活性[4];还有一些研究者探索解决运行实例重构/进化的问题,如提出操作原语集、一致性规则、变换函数、实例版本化等[7,9].

经过 OO 语言、操作系统和分布系统等领域的研究实践,反省(Reflection)计算技术已被公认为是构造灵活系统的有效方法[3,8].在 WfMS 中引入基于过程的知识构造反省 Agent,通过改变 Agent 的知识来改变过程的执行,从而实现灵活性[11];在文[10]中,元级过程模型可在元级被检查和修改;一种通用的反省对象知识模型 ROK 可用于表达过程和实例的结构和行为,以及引擎的设计实现[6].这些研究着重揭示工作流过程的元结构,对元级设施还缺乏进一步的抽象,在从元级操纵系统来增加系统灵活性、定制能力方面,仍有待深入.

本文在对 WfMS 从结构和行为方面进行具体化(Reification)、进而得到反省模型的基础上,提出了元对象协议(MOP)模型,该模型使 WfMS 中元级与基础级的分离和关联关系更加清晰,提供了在系统元级编程和操纵的手段,以及解决元对象的管理问题.通过基于反省思想和技术的实例进化示例,本文验证了反省方法能有效地提高 WfMS 灵活性和适应性.

2 WfMS 的反省模型

反省的目的是捕获应用的某些隐藏层面,并用合适的数据结构揭示出来.在对象语言和分布式系统的反省中,比较重要的是将系统关键的概念和行为进行具体化,即表达为程序可操纵的对象或结构[3,5].由于基于 OO 语言的程序结构和工作流在很多方面相似,因此,我们通过类比得到 WfMS 的反省

收稿日期:2002-06-30;修回日期:2002-09-30
基金项目:四川省重点科技攻关项目(No.SG95.17.1);云南省计算机技术应用重点实验室开放基金项目(No.99.3004)

模型.

2.1 工作流描述

工作流模型,也称为过程定义,其描述方法有多种[1].我们采用的方法和文[6]类似,其中特别将同步和条件选择抽象为具有特定功能的实体元素,其图形化表达也比较方便和直观(图 1).任务(task)是通过某些工作达到某个确定目标的基本工作单元;条件选择体(decision)的作用是根据某种条件选择一个后继分支执行;同步体(synchroniser)则处理多条并行任务路径的同步.任务、条件选择体和同步体都是过程定义中的实体(Entity),某种意义上后面两者都是特殊的任务.而实体间是由触发边(Trigger)连接的,即实体由触发边顺序关联.

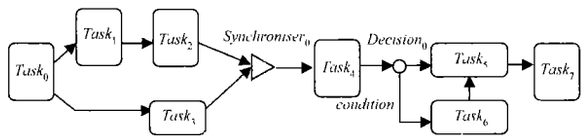

图 1　工作流模型示意图

在此基础上可建立面向工作流的反省模型.为了突出反省机制,我们对过程模型做如下简化和假定:

(1)过程中任务均为具体原子任务,即定义中不包括复合过程,且在实例化前已经确定;

(2)过程中包含唯一的初始任务和结束任务;

(3)执行任务的人或代理均抽象为特殊的资源,和数据对象类似看待;

(4)除选择体的条件外,定义中不指定其他的条件类型;

(5)任务执行不出现异常,即定义中无异常处理机制.

2.2 过程的结构反省

结构反省指元对象提供的机制,检查和访问基础级应用的结构信息、状态空间等,例如:OO 语言结构反省的元级信息包括:对象所属的类、派生结构、当前运行方法等.一些反省语言对这些概念进行了不同形式的具体化[5,8].类似地,我们可以对前面工作流描述中介绍的概念进行具体化,相应表达为如下定义:

定义 1　工作流结构元素:

Task: : = {id: *EntityId*, title: *String*, url: *String*, arl: *ResourceLink*, pred: *EntityId*, succ: set < *EntityId* > }

Decision: : = {id: *EntityId*, pred: *EntityId*, succ: set < {*Condition*, *EntityId*} > }

Synchroniser: : = {id: *EntityId*, pred: set < *EntityId* > , succ: *EntityId*}

Trigger: : = {from: *EntityId*, to: *EntityId*}

Resource: : = {id: *DataId*, name: *String*, type: *DataObject*}

ResourceLink: : = {tid: *EntityId*, reads: set < *DataId* > , writes: set < *DataId* > }

除 2.1 节中介绍的几种基本元素外,Resource 和 ResourceLink 分别代表过程定义中的数据资源和任务与资源的逻辑连接.这些元素可以在元级被反省为简单元对象.而过程定义的元对象则是由这些简单元对象复合而成,其定义如下:

定义 2　工作流过程

Process: : = {title: *String*, tasks: set < *EntityId* > , synchronisers: set < *EntityId* > , decisions: set < *EntityId* > , triggers: set < *Trigger* > , resources: set < *Resource* > , begin: *EntityId*, end: *EntityId*}

类似于类实例化为对象,过程定义在执行前也需要被引擎实例化,表现为活动实例.但与对象不同的是,过程实例在运行中的结构和静态时并不一样,包含更多的维护调度与执行中的信息,这些信息在执行中不断变化.因此,我们也需要反映过程实例结构的元对象:

定义 3　工作流过程实例

Instance: : = {id: *InstanceId*, body: *Process*, finished: sequence < *EntityId* > , ready: sequence < *EntityId* > , scheduled: set < *EntityId* > , signalled: set < {*EntityId*, set < *EntityId* > } > }

其中,body 代表实例对应的过程定义,finished、ready、scheduled 和 signalled 是实例执行中处于不同状态的实体集合或序列,分别对应:已完成任务、就绪任务、已调度执行中任务和处于同步中的同步体.

通过对过程定义和实例结构的揭示,我们可以进行结构反省的具体化过程,所得到的数据结构是一系列的元类.外部程序可以在元级读取和操纵揭示过程信息的元对象.同时,对结构的反省也是以下对引擎中过程调度行为反省的基础.

2.3 过程管理的行为反省

除了揭示系统深层信息外,反省的另一个主要目的是定制系统的行为,因此将那些对基础级应用本来透明的行为通过某种形式具体化是很有必要的.从应用角度,程序行为是连续发生的;但从系统角度,其过程可能包含若干系统内部状态的变换,其间经历了若干系统级的行为.如果将行为开放出来,允许以特定的方式定制,这就称为对系统行为的具体化,或行为反省.在 OO 语言和分布系统中,常见的行为反省有:对象方法的分派、方法的执行、接收消息、消息编解码等[3,5],这些行为定制增加了系统的灵活性和适应性.类似的 WfMS 行为定制主要在两个阶段:过程建模和执行,各自的行为有明确的区分.

过程建模一般称为模型进化,主要进行建模阶段的模型修改.另外,为了存储和加载过程定义,将过程定义保存为一定格式并把这种格式调入执行核心,对模型的典型操作还包括序列化.修改过程总是可以分解为一系列原子性的操作,简单列举如表 1.

表 1　修改过程定义的原子操作

Insert	Task, Synchroniser, Decision, Trigger, Resource, ResourceLink
Remove	
Set	title, begin, end

过程的执行是引擎内若干过程实例经历其生命周期的各个阶段,执行中的行为在引擎调度执行过程实例中总是伴随发生,导致引擎的系统或实例的内部状态发生变化,在非反省系统中,这些行为对过程设计者和参与使用者是透明的.本文通过合适的元类反映为可操纵的编程对象,将这些行为进行具体化,从而实现对这些关键行为的定制.这些可能的行为归纳如表 2 所示:

在实现上,为了提高在元级对结构和行为的各个方面操

表 2 过程执行中的关键行为具体化

名称	涵义	标识符
InitializeInstance	将过程定义实例化	Initialization
DestructInstance	过程实例的销毁	Destruction
ScheduleTask	调度实例,选择就绪任务	Scheduling
ActivizeTask	分配资源,激活任务	Activation
ExecuteTask	根据任务定义的执行过程	Execution
CompleteTask	完成任务,释放资源	Completion
Synchronize	并行任务同步	Synchronization
SelectTask	根据条件选择后继任务	Selection
SelectInstance	选择过程实例进行调度	Focusing
Receive/SendMessage	接收和发送消息	Receiving/Sending
DispatchMessage	分派消息到过程实例	Dispatching

纵的能力以及反省范围控制的灵活性,我们使用反省种类(category)指明哪些方面需要被反省,对应每个种类定义了元类.例如 Process 表示了对过程定义反省的种类,其元类是 MProcess,可提供过程定义的元信息,以及操纵过程定义的所有元接口.对种类有两种使用方式,一是仅具体化,而不定制实现,一般用于揭示元级信息;二是通过派生的元类实现具体化功能,用于定制系统的元级行为.需要具体化及派生元类的种类可通过下节介绍的元对象协议声明显式指定.

3 元对象协议模型

前面介绍了反省的 WfMS 哪些方面可以被具体化,具体化的元对象相当于提供了系统级的访问入口.但如何组织元对象,并与基础级系统关联,进而影响系统的行为则是元对象协议模型需要解决的问题.在编程语言中,元对象协议 MOP 反映了元对象提供服务的接口,反省通过 MOP 提供语言的辅助接口,允许设计者可以增量地修改其行为和实现[3,8].同样,如果把过程描述看作一种语言,MOP 也可说明如何为过程描述增加反省的能力.MOP 模型涉及元对象协议声明、元对象关联、协议选择,及元对象管理.

3.1 协议声明

实现对系统设施的定制或增加功能往往需要具体化多个方面的元对象组合在一起.我们通过 MOP 声明包含若干反省种类的元对象的这种关系,进而明确元对象接口.MOP 声明可用 XML 形式描述,其 DTD 语法如图 2 所示.在协议中,是一些对需要具体化的反省种类的声明,每个声明由类型、种类、定制元类和元对象名组成,而后两项不是必须的.类型包括:ATTR,BEF,AFT,SELE,REPL 等几种,分别表示元对象:作为协议对象属性;作用在行为前、行为后;作为可选项接受选择;替换实现定制的行为.除 ATTR 外,其它类型分别对应 3.3 节中介绍的不同元对象管理机制.category 说明了需要被具体化的种类.定制元类的名称由 custom 属性指定.标记的内容如果存在,是元对象的名字.

```
<? xml version = "1.0" encoding = "ISO = 8859 - 1"? >
<! DOCTYPE protocol[
<! ELEMENT protocol(reify + ) >
  <! ATTLIST protocol name CDATA # REQUIRED >
<! ELEMENT reify( # PCDATA) >
  <! ATTLIST reify
    type(ATTR | BEF | AFT | SELE | REPL) # REQUIRED
    category CDATA # REQUIRED
    custom CDATA # IMPLIED >
]>
```

图 2 MOP 声明的 DTD 语法

MOP 声明可用于:在过程定义中通过协议选择与过程对象进行关联;指示引擎核心加载及登记元对象,建立绑定;从逻辑上说明若干种类元对象的组合关系;描述元对象在元级中接受管理的方式;帮助生成元组件的框架代码.

3.2 元对象关联和协议选择

通过反省可以将系统分为基础级和元级,前者即通常意义的应用,而后者代表对应用透明的应用执行环境或设施,可以通过元对象检查甚至调整系统设施保证开放性.基础对象功能必须借助元级设施才能完成,因此元级和基础级的分离并非完全隔离,元对象总是和一个或多个基础对象发生关联,而基础对象则几乎透明地和一个或多个元对象关联.

实现上,关联关系的建立可以是静态或动态的,前者可通过协议选择在过程描述中声明,而后者可通过管理工具为运行中的对象绑定元对象.按关联的范围区分,协议选择可针对系统(默认)、过程、任务/实体、资源等.以下是 XML 格式的过程定义选择协议的示例.

```
< ! - - process PCInstall is associated with MOP TimeStat &
    AuditLog - - >
< process title = "PCInstall" MOP = "TimeStat, AuditLog" >
  < ! - - the task ReqDevice is started by sending E-mail
      form - - >
  < task title = "ReqDevice" MOP = "EMailForm" > ……
    < /task >
  ……
< /process >
```

该示例有三个 MOP 关联,前两个与过程体本身有关,作用范围是整个过程定义;而 EMailForm 则只与任务 ReqDevice 关联,即仅在该任务执行时起作用.具有关联的元对象在系统执行中起作用,对元对象的修改(元级)将最终影响到应用对象(基础级)的状态或行为.

3.3 元对象管理

对系统行为的定制可能需要多个元对象的参与,而对系统的多重定制也容易出现系统某个方面的多重具体化,因此,在元级如何管理多个元对象是很重要的.管理从两个角度进行:如何使具体化不同方面的元对象共同协作;如何协调多重具体化的元对象.我们用设计模式解决不同类别的元对象管理问题,几种常见的元对象管理模式如图 3 所示.

如图 3(*a*)的中介者(Mediator)模式解决元对象的聚合问题,从而实现多种不同种类的元对象参与系统行为的定制,中介者的生成通过 MOP 的声明实现,参与的元对象可共享中介者提供的数据空间,在此基础上彼此协作交互.如图 3(*b*)的组合(Composite)模式提供了一种多重具体化的若干元对象的

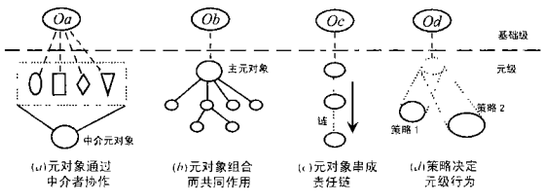

图 3　几种元对象管理的模式

灵活协调机制,组合的元对象管理各个子元对象,根充当主元对象和基础对象关联,不同的遍历方式导致不同的方式完成复杂的元级行为.如图 3(*c*)的责任链(Chain of Responsibility)模式可以实现元对象的选择关联,在处理元级请求时,链上的元对象限于处理能力,每次只有一个可以处理.图 3(*d*)是应用策略模式,这种模式往往在需要替换性修改系统行为时采用,实现不同策略的元对象在不同场合下和基础对象发生关联.但需要注意的是,不能同时有若干策略元对象在元级并存,因为它们往往在行为上是冲突的.

需要注意的是,尽管元对象管理机制是灵活的,在通过元对象定制系统行为时还是应当注意防止冲突的发生,毕竟元级编程在提供系统灵活性的同时总会造成一定的安全隐患.

4　应用举例

本节以实例进化为例,说明通过 WfMS 的反省和 MOP 的使用,元级对系统功能的增加或修改和基础级应用(过程的功能定义和任务实现部分)的耦合关系非常少,且使用方式灵活统一.实例进化(Instance Evolution)是自适应工作流中重要的研究课题之一[4,7,10],其基本要求是当过程定义由于某种原因发生变化(升级或临时性变化),处于运行中的实例在保证正确性的前提下取得与变化后的定义一致.相对一些已提出的方法,本文基于反省技术采取比较保守的进化思路,即只要变化没有波及已调度执行的区域,就可进化,过程实例在动态运行中与负责进化的 MOP 发生关联,绑定合适的元对象负责进化.实例进化的 MOP 声明如下:

```
< protocol name = "LazyInstanceEvolution" >
  < reify type = "ATTR" category = "Process" > new_proc < /
    reify >
  < reify type = "ATTR" category = "Instance" > curr_in-
    stance < /reify >
  < reify type = "BEF" category = "Scheduling"
      custom = "MLIEScheduling" > lie_scheduler < /reify >
< /protocol >
```

基于声明的 MOP,用 Java 语言定义实现定制的元类 MLIEScheduling 如下:

```
class MLIEScheduling extends MScheduling{
  ……
  int schedule(…){
    复制 curr_instance.body 为 old_proc;
    比较 old_proc 和 new_proc,得到对过程定义的刷新原
    子操作列表 op_list;
      for(op_list 中每个操作 op){
        检查 curr_instance.finished;
        if(op 所针对的元素没有调度过)
          对 old_proc 执行操作 op;
        else
          实例不能进化,解除元级关联,返回;
      }
      将重构后的 old_proc 赋予 curr_instance.body;
      将 curr_instance.scheduled 中的任务清除;
      解除元级关联,返回;
  }
};
```

5　技术实现方法

从实现角度,工作流引擎的框架提供元对象/组件的管理设施,对系统追加或定制的功能可以通过向引擎中登记元对象灵活地实现,如过程模拟.

描述为 XML 格式的元对象协议,可以引导核心以适当的方式加载元组件.根据元对象协议可以生成元组件的框架代码,结合元类的实现可以编译为元组件.在核心完成对基本元组件的加载和资源初始化后,引擎即启动,从而可以执行具体的应用过程.

为方便维护、加载及交换,过程定义同样基于 XML 形式描述,其中可以通过引用声明元对象协议的 XML 文件及显式的协议选择建立过程对象和元对象的关联.管理者可以将过程定义加载到引擎核心,进而手动或自动实例化.引擎中可同时维护同一个或不同过程定义的若干实例,它们被并发调度和执行,在此期间,与核心或过程对象绑定的元对象在不同方面具体化过程的结构或调度执行的行为方面.

在管理方面,系统可以通过绑定合适的元对象到过程上,实现过程执行的日志记录、过程实例动态信息的跟踪.同样道理,也可以对过程执行的性能和资源使用情况进行统计,以帮助设计人员对过程进行优化.

6　结论

本文将反省技术用于工作流系统中,通过反省模型揭示 WfMS 的结构和行为方面特性,从而可以在元级访问和修改过程定义与实例的元信息,或者定制系统设施中的行为,提高了 WfMS 的灵活性.为了更好地抽象元级,便于使反省模型通过灵活高效的手段编程操纵,本文提出了元对象协议模型,这样可有效地组织不同协作方面的元对象构成元级的逻辑实体,建立基础级和元级有效的联系,显式地实现元对象协议与基础对象的关联,有效地处理元对象的协作与协调等组织和管理问题.借助元对象协议,文章通过实例进化典型应用示例验证了基于反省机制的方法对工作流系统是有效而灵活的.

随着对过程的进一步深入建模,如增加异常处理、允许复杂结构等,其反省模型将不断发展,以涵盖工作流管理的更多层面.另外,元对象的一致性和冲突管理,以及基于反省机制处理一些 WfMS 中的高级问题等研究还将继续深入.

参考文献:

[1] 史美林,杨光信,等.WfMS:工作流管理系统[J].计算机学报,1999,22(3):325-334.

[2] 史美林,杨光信,等.一个基于 Web 的工作流管理系统[J].软件学报,1999,10(11):1148-1155.

[3] 王敏毅,姚绍文,周明天.反省的对象中间件—理论与方法研究[J].计算机科学,2001,28(4).

[4] Amit P Sheth, et al. Processes driving the networked economy[J]. IEEE Concurrency, 1999, 7(3): 18-31.

[5] Brendan Gowing, Vinny Cahill. Meta-object protocols for C++: The iguana approach[A]. Proc. of Reflection'96[C]. San Francisco: Reflection, 1996.

[6] David Edmond. Applications of Reflection for Cooperative Information Systems[D]. Queensland University of Technology, 2000.

[7] F Casati, S Ceri, et al. Workflow evolution[J]. Data and Knowledge Engineering, 1998, 24(3): 211-238.

[8] G Kicazles. Beyond the balck box: Open implementation[J]. IEEE Software, 1996, 13(1): 8-11.

[9] M Reichert, P Dadam. ADEPTflex: Supporting dynamic changes of workflows without loosing control[J]. Journal of Intelligent Information Systems, 1998, 10(2): 93-129.

[10] T D Meijler, et al. Realising run-time adaptable workflow by means of reflection in the baan workflow engine[A]. Proc. of the CSCW Workshop on Adaptive Workflows Management [C]. Westin Seattle: CSCW, 1998.

[11] U Borghoff, P Bottoni, et al. Reflective agents for adaptive workflows [A]. Proc. of PAAM-97[C]. London, UK: PAAM, 1997.

[12] W M P van der Aalst. Generic workflow model: How to handle dynamic change and capture management information[A]. Proc. of 4th IECIS [C]. Edinburgh, Scotland: IEEE/CS, 1999.

作者简介:

姚绍文 男,1966 年 8 月生于湖南省永顺县,电子科技大学计算机应用技术专业毕业,博士,云南大学信息学院教授、中国计算机学会西南网络与 MIS 专委会副主任委员,1998 年至 1999 年为澳大利亚南澳大利亚大学电信研究所访问学者,主要研究方向为语义 Web 技术、网络协议工程、网络分布式计算、着色 Petri 网(CPN)建模、知识工程技术等,近年来,先后在国内外发表学术论文 30 余篇.

王敏毅 男,1973 年 6 月出生于浙江嵊县,电子科技大学计算机应用技术专业毕业,博士,主要研究方向为计算机网络、分布式对象技术、移动计算、Agent 技术,先后在国内外学术期刊及会议上发表论文 20 余篇.

周明天 男,1939 年 3 月出生于广西容县,电子科技大学计算机学院教授、博士生导师,主要研究方向为网络信息系统、分布式处理系统、对象中间件技术等.在国内外学术期刊及会议上发表学术论文 100 余篇.