基于服务协作中间件的动态流程模型*

刘绍华+, 魏 峻, 黄 涛

(中国科学院 软件研究所 软件工程技术中心,北京 100080)

A Dynamic Process Model Based on Service Cooperation Middleware

LIU Shao-Hua+, WEI Jun, HUANG Tao

(Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62630989 ext 213, E-mail: ham_liu@otcaix.iscas.ac.cn, http://www.iscas.ac.cn

Received 2004-02-10; Accepted 2004-05-08

Liu SH, Wei J, Huang T. A dynamic process model based on service cooperation middleware. *Journal of Software*, 2004,15(10):1431~1440.

http://www.jos.org.cn/1000-9825/15/1431.htm

Abstract: Emerging service oriented architecture is increasing the evolution and variation of software resources. Applications that cannot adapt to dynamic environments will decrease their usefulness, particularly to business process management systems that face requirements changed frequently. In response to the realistic requirements, this paper, based on Web services and business process management techniques, propose a model of service cooperation middleware (SCM) for enterprise computing. First, it discusses the conceptual architecture and operation mechanisms of SCM, then formalizes the meta-model of cooperative processes deployed on SCM. By introducing a model transformation function into the formalization, a static process model can be extended to a dynamic one. In virtue of the reflection capability, structural reconstruction and behavioral adaptation of the dynamic processes can be achieved in SCM by introspection and effectuation. In terms of the conceptual model of SCM, a process virtual machine (PVM) is designed as a running container for cooperative processes. Borrowed from the power of MDA mechanism, the model of a business process can be transformed successively from design to run time. The business processes established on SCM can be more flexibly and extensively applied to various open environments, and leverage modern enterprise computing.

Key words: service cooperation; dynamic process; variable structure; reflection; process virtual machine

摘 要: 新兴的面向服务体系结构正在加速软件的发展和变化,无法适应动态环境的应用将逐渐失去作用,尤其对那些面临着需求频繁变更的业务流程管理系统而言更是如此.为了响应这种现实需求,基于 Web 服务和业

* Supported by the National Natural Science Foundation of China under Grant Nos.60203029, 60173023 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2001AA113010, 2002AA413610, 2003AA413010, 2003AA115440 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312005 (国家重点基础研究发展规划(973))

作者简介: **刘绍华**(1976－),男,江西丰城人,博士生,助理研究员,主要研究领域为网络分布计算,工作流技术,服务协作理论,中间件;**魏峻**(1970－),男,博士,副研究员,主要研究领域为软件工程,软件理论,网络分布计算;**黄涛**(1965－),男,博士,研究员,博士生导师,主要研究领域为软件工程和方法学,分布式对象技术.

务流程管理技术,提出了服务协作中间件(service cooperation middleware,简称 SCM)模型,探讨了其内部机理与协作流程元模型的形式化.通过引入模型转换,扩充流程状态空间,定义了动态流程模型,动态流程的结构重构和行为自适应可以借助 SCM 的反射能力得以实现.针对 SCM 模型,设计了流程虚拟机(process virtual machine,简称 PVM),流程在运行时由 PVM 控制执行.借助于模型驱动机制,业务流程可从设计到运行不断地进行模型变换.由此建立的业务流程能够更灵活、更广泛地适用于各种开放环境,提升企业分布计算.

关键词: 服务协作;动态流程;可变结构;反射;流程虚拟机

中图法分类号: TP311 **文献标识码**: A

伴随着商务活动的全球化,企业价值的提升将更多地源于服务.企业的经营活动正在迅速渗透到 Web 服务中,借助于 Internet 提供全天候、个性化、零距离的优质服务.与面向服务的潮流相呼应,业界正在进行着一场面向服务计算模式的变革.Web 服务技术目前引起了广泛的关注,它一改传统的面向过程、面向对象和面向构件的软件开发方式,成为分布计算、服务互操作和协作的新范型.Microsoft,IBM,Oracle,SUN 等业界巨头都积极投身其中,极大地促进了它的发展[1].

同时,工业界也正在将更多的企业流程扩展到 Web 上,以使这些流程能够在企业内或企业间相互协作,这就是业务流程管理(BPM)[2]的兴起.BPM 又被称为新的复合计算、下一代工作流系统、应用中间件的新平台.它是近几年来自企业内的资源整合技术如 ERP,CRM,EAI 和企业间的交易协作技术如 EDI,e-Business,B2B,Web Service 的碰撞、融合与创新.

面对快速变化的市场需求,企业计算必须以动态和灵活作为应对策略,相应的 BPM 软件必须支持流程实例根据运行环境和变化的需求,通过流程模型和支撑系统的通用机制,动态地改变模型结构和行为,适应各种变化.本文基于 Web 服务和业务流程管理技术,研究协作流程适应环境和需求变化,以及动态调整结构和行为的模型和机制.本文首先探讨基于 Web Service 技术的服务协作模型和服务协作中间件(service cooperation middleware,简称 SCM)概念体系结构;然后建立面向服务的协作流程元模型,由此引入动态流程模型的概念,并由 SCM 的反射能力支持动态流程的结构重构,最后设计适于动态流程执行的流程虚拟机(process virtual machine,简称 PVM)平台.

1 服务协作中间件

1.1 中间件发展回顾

企业在核心基础业务平台之上集成各种软件应用,对业务流程实现监控、管理与分析,在各个业务系统间和跨企业合作伙伴间,安全、有效地实施交互和协作,全方位满足各种业务功能的要求,这需要各种中间件的有力支持.

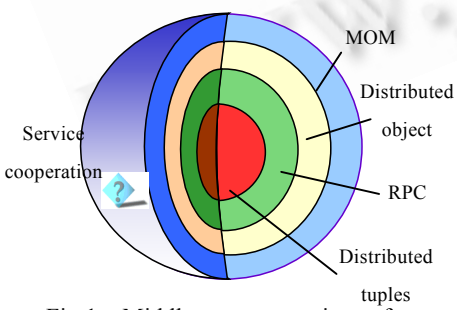

Fig.1 Middleware comparison of abstraction and granularity

图 1 中间件粒度和抽象层次对比

企业计算领域几乎引入了所有核心中间件技术,如事务处理(TPM)、远程过程调用(RPC)、CORBA 总线、可靠消息队列(MQ)等.一般认为,中间件系统主要归为如下 4 种类型:分布式元组(distributed tuples)、远程过程调用、分布式对象(distributed object)和面向消息中间件(MOM).它们的差别在于所提供的编程抽象性及跨越网络和硬件平台的异构性[3].

根据计算单元的粒度,我们将这 4 种中间件分别归于不同的层次,如图 1 所示.分布式元组提供了非常细粒度的数据元组以及元组空间的抽象,并成为较广泛分布的中间件形式之一.远程过程调用扩展出更粗粒度的过程接口,以提供能够跨网络调用某个过程的抽象性.分布式对象提供了粒度仍然很小,但抽象程度毕竟更高的对象模型,尽管对象本身是远程

分布的,但是对其方法的调用可以就像是调用本地空间的对象一样.面向消息的中间件以消息的形式提供了更宏观的抽象性,消息队列可以跨网络进行访问,非常灵活,这种灵活性使得它可以自由地组织和配置那些存放和提取消息的程序之间的拓扑关系[4].

1.2 服务协作模型与服务协作中间件概念体系结构

为了适应现代企业计算的需求,我们引入了一种基于服务协作的新的中间件形式,称为服务协作中间件SCM,旨在将分布的服务单元组织在一起,通过服务粒度上的交互和协作来满足业务需求.与传统中间件相比,SCM 是一种具有更高的抽象程度和计算粒度的系统中间件.

服务,就是由提供者提供给接收者的功能系统,并以服务描述的形式构成两者之间的一种契约关系,其中提供者承诺提供,而接收者同意接收.可以认为,任何 Web 服务都应成为计算机实现的业务流程的一部分,换句话说,都是其他企业计算行为的上游或者下游,只有如此,它才真正成为服务.因此,那些为业务功能而发布的 Web 服务需要按照一定的流程组织在一起.服务协作中间件 SCM 采用的协作模型内部由 3 层抽象构成:协作框架(cooperative framework)、协作流程(cooperative process)和协作活动(cooperative activity).

- 协作活动指的是协作流程中对应于服务调用的最小单元,它可能只涉及一个参与者,也可能需要两个或者更多参与者相互协同完成,在协作活动中由参与者执行的动作可能包括:特定参与者之间的数据交换,在共享的存储空间中将数据分享给活动中所有的参与者以及其他面向服务的活动等.
- 协作流程是用于协调或者指挥某些业务合作伙伴之间特定服务协作执行的序列,流程中的每一步都对应着一个协作活动.
- 一系列相关的协作流程组织在一起便构成了一个协作框架.
- 协作模型的另一个重要方面是投影(projection).一个协作框架到某个特定参与者的投影,就是该框架中仅限于该参与者所能接触的视图.投影能将与具体参与者相关的活动及与其他参与者的协同分离出来,便于灵活构造协作模式.

SCM 的概念体系结构如图 2 所示.

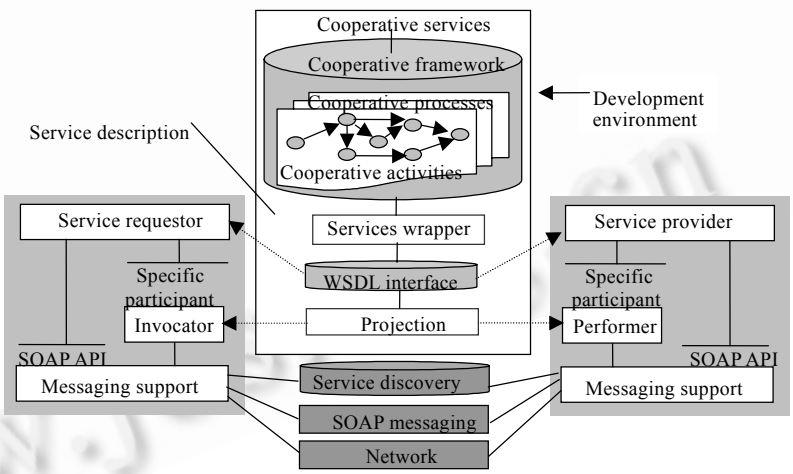

Fig.2 Conceptual architecture of service cooperation middleware

图 2 服务协作中间件概念体系结构

所有相互协作的服务(cooperative services)按照协作模型组织在一起;通过服务包装(service wrapper)将协作模型各层次以 WSDL 接口的形式统一对外呈现;可以通过投影机制将视图分割给各个参与者.所有服务提供方和请求方都是协作流程中的参与者,它们基于 Web 服务的注册、发现、绑定机制,遵从服务描述,以高度可互操作的通信协议进行通信.

SCM 内在的 Web 服务是自描述的,所以 SCM 中参与协作的服务、框架、流程、活动等各种实体事实上都包装了自描述的 WSDL 格式的统一接口,这也是一种良好的适于反射计算的接口形式,运行时系统能从中发现反射实体中的服务描述、消息格式、端口操作与参数类型等重要信息,并通过手工或者自动绑定到服务端口来完成具体的操作.由服务描述的 WSDL 接口扩展而来的反射机制将在第 4 节深入阐述.

2 协作流程元模型

2.1 协作流程的基本组成

协作流程由状态、转换以及状态转换过程中传递的消息等组成[5,6].

定义 2.1. ($a\in$)Activity(活动集合).活动可以看成是服务提供者参与的抽象操作,既可以是单个参与者执行的原子动作,也可以是多个参与者之间的一次协作.

定义 2.2. ($m\in$)Message(消息集合).消息是流程中的数据域,在一个流程内部或者多个流程间共享.消息能够在运行时获取和传达某些含义.对消息 m 的调用会访问到 m 的存储,并可能修改它的值.一个消息由几个部分(part)组成,某个部分又可以是另一个消息,因而消息可以递归定义成多维结构,以表达任何真实的复杂数据类型.

定义 2.3. ($s\in$)State(状态集合).对于运行中的流程来说,其状态主要是指活动的执行态.又令 B 代表流程的诞生,E 代表流程的终结,则有 State=Activity$\cup\{B,E\}$.即一个流程要么刚诞生,要么正在执行活动,要么已经终结了.

2.2 状态和消息的复合语法

定义 2.4. 状态转换Γ是一个三元组(State,Condition,$\rightarrow$),其中:

($s\in$)State 是一个状态集合,

($c\in$)Condition 是一系列条件,

"$\rightarrow$"是 State×Condition×State 的一个子集,即$\rightarrow\subseteq$State×Condition×State.

为便于理解,我们使用 $s\rightarrow s'(c)$代换$(s,c,s')\in\rightarrow$,表示在某个条件 c 下流程能从状态 s 转移到另一个状态 s'.

定义 2.5. 消息加工Λ是一个三元组(Message,State,$\uparrow$),其中:

($m\in$)Message 是消息集合,

($s\in$)State 是状态集合,

"$\uparrow$"是 Message×State×Message 的一个子集,即$\uparrow\subseteq$Message×State×Message.

同样,将$(m,s,m')\in\uparrow$简写为 $m\uparrow m'(s)$,表示消息加工,在经历了状态 s 之后,将输入消息 m 加工为输出消息 m'.

2.3 流程与控制流语义

定义 2.6. 流程 P 是一个五元组(State,Message,Condition,Γ,Λ),其中各项定义如前.

为了阐述流程中更具体的行为,我们部分地借鉴了控制流语义[7]的思想,本文的"控制流"特指运行时流程用于说明后继执行步骤的概念,即流程状态变化的轨迹.它可以是流程的诞生,或者是某个状态,或者是状态的顺序、并行、排他、选择行为,直至流程最终结束.

定义 2.7. 令 $s\in$State,B 代表流程的诞生,则$f\in$Control-flow(控制流集合)由$f::=B|s|(s;s)|(s||s)|(s\hat{}s)|(s|s)$式给定,其中,由顺序(sequential)算子";"复合在一起的两个状态被依次通过,而由并行(parallel)算子"||"复合的两个状态被同步执行,由排他(exclusive)算子"^"连接的状态只能有一个被通过,选择(alternative)算子"|"用于表达控制流中的某些非确定选择,即各状态被独立通过,只取决于其自身的条件是否满足.

基于该表示法,就可以描述协作活动如何复合成协作流程.

2.4 分散的P2P模式与投影计算

在分散式的、面向服务的流程执行环境中,任何服务提供者都是业务流程中的一个参与者,也就是其他提供者行为的上游或者下游.如果每个参与者只与相邻的上下游业务伙伴协作,而不需要中央控制引擎,那么它提供了一个更加自治和分散的解决方案,即对等(peer-to-peer,简称 P2P)计算模式.采用 P2P 模式,所有参与者便都是对等端,并能通过投影操作得到相邻端的对应接口,如图 3 所示.

定义 2.8. 投影 Projection,是从流程(State,Message,Condition,Γ,Λ)中分离出来的一部分,是一个七元组:

Projection=(s,$P(s)$,$S(s)$,Ms,Cs,Γs,Λs),

其中 s 是状态集合 State 的一个子集,赋予投影所面向的某个参与者,该参与者端负责内部的操作实现,并能够自动由某个状态转换为另一个状态,某些状态可能又同时属于其他参与者端,由此形成一个调用-被调用链.$P(s)$是 s 的前驱集合,$S(s)$是 s 的后继集合,Ms,Cs,Γs,Λs 分别表示 s 中所涉及的相应的消息、条件、状态转换和消息加工.

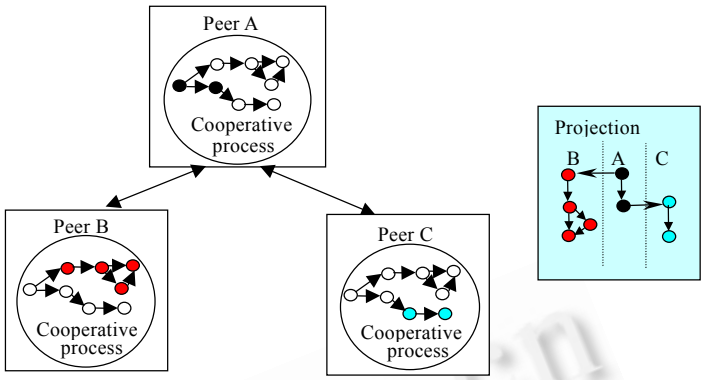

Fig.3 Projecting a process onto upstream or downstream participants

图 3 根据上下游关系将流程投影到参与者

在协作流程模型中,一个对等端作为活动中的一个参与者,其上游对等端是指那些执行其前驱状态的参与者,下游对等端则指那些执行后继状态的参与者.

定义 2.9. 假设给定了状态转换Γ=(State,Condition,→),则对于每个 s∈State,有

- 前驱集合 $P(s)$由下式给定 $P(s)=\{(c,s')|s'\rightarrow s(c)\}$;
- 后继集合 $S(s)$由下式给定 $S(s)=\{(c,s')|s\rightarrow s'(c)\}$,其中 c∈Condition[6].

3 动态流程模型与可变结构

3.1 动态流程模型与模型转换函数

在设计系统结构的过程中,经常会面临许多可变的需求,系统必须在运行时适应改变的需求,动态进行调整.为此,我们跟踪了可变结构系统方面的研究[8],尤其是动态 DEVS(离散事件系统规约)之后[9],提出如下动态流程模型:

定义 3.1. 动态流程抽象定义为一个六元组.

DyProcess=(State,Message,Condition,Γ,Λ,Ω).其中前 5 项的定义如前.Ω代表模型转换,如下定义.

定义 3.2. 模型转换Ω是一个五元组(Message,State,Condition,DyProcess,↓),其中

(m∈)Message 是转换前后两个动态流程中相同消息组成的集合;

(s∈)State 是发生动态流程模型转换的状态;

(c∈)Condition 是发生模型转换的条件.当满足条件时,动态流程从当前状态转移到另一个动态流程中类似配置的状态,即转换为具有该相同转换接口的另一个模型;

(d∈)DyProcess 是当前动态流程模型可以转换过去的其他动态流程集合;

↓⊆State×Condition×DyProcess.

通过动态流程和模型转换的迭代定义,我们可以构建不同动态流程模型之间的转换关系,以使它们可以通过模型转换函数进行转换.

3.2 动态流程模型的可变结构特性

实际上,动态流程模型是动态 DEVS(离散事件系统规约)的一种特殊情况.

定理 3.1. 动态流程是一种特殊形式的动态 DEVS,它也可以表示成动态 DEVS 的形式:

$$\langle X,Y,m_{init},M(m_{init})=(S,s_{init},\delta_{ext},\delta_{int},\rho_{\alpha},\lambda,ta)\rangle.$$

文献[5]已经给出了它的证明,这里不再赘述.下面说明由此定理引出的新含义.

动态流程可以看成是一系列输入事件 X、状态 S、输出事件 Y、内部转换函数δ_{int}和外部转换函数δ_{ext}、模型间转换函数ρ、输出函数λ和时间函数 ta.对于一般事件,状态只是在模型内部发生转换,并可能产生输出.模型

之间的转换函数则由某些约束情况下特殊的输入事件触发,例如服务已经停用、调用失败、连接中断、超时等.在模型转换函数的作用下,原来的模型就能正确转换为一个新模型,如图 4 所示.

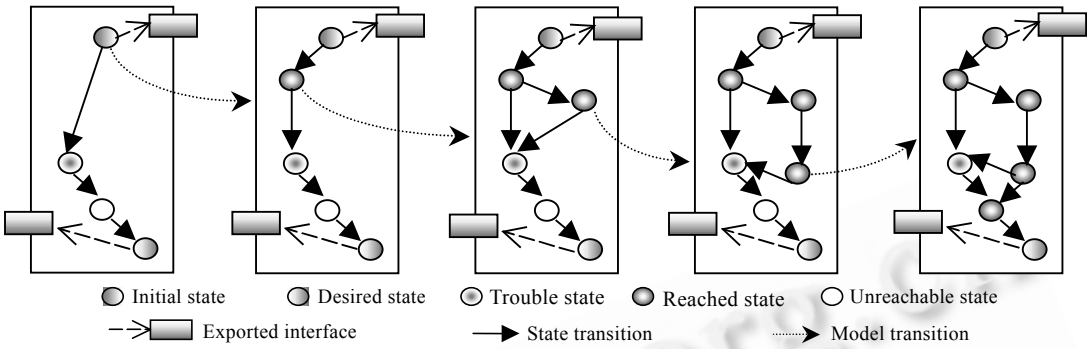

Fig.4 An example of model transitions in dynamic processes

图 4 动态流程间模型转换的例子

对于动态流程来说,它的状态空间、内部和外部转换函数、输出和时间函数、模型转换函数等都是可以变化的.动态流程可以理解为添加了模型转换函数的一系列具有相同转换接口的一般流程模型,由模型转换函数决定哪个模型作为原来模型在某些状态下的后继.

借助于上述严格定义的转换函数和元数据共享,例如共享 WSDL 接口、通用 XML 消息格式以及能够一致理解的转换约束条件等,静态流程也可以通过扩展其原有的状态空间,使之包含可能发生的改变,并利用模型转换函数来产生动态可变结构的流程,以实现动态可变的需求.因而,该动态流程模型与可变结构系统是一脉相承的.

4 基于 SCM 的反射机制支持动态流程

4.1 SCM中的反射机制

自适应动态流程可以借助于 SCM 中的反射机制来实现.反射(reflection)是系统能够在运行时发现和调整自身的一种能力[9,10].运行时改变流程结构和行为的方法之一就是将其定义为 SCM 中的反射流程.Web 服务天然就是自描述的,所以 SCM 中的协作流程可以包装成自描述的 WSDL 服务形式,并设计出相应的反射实体.

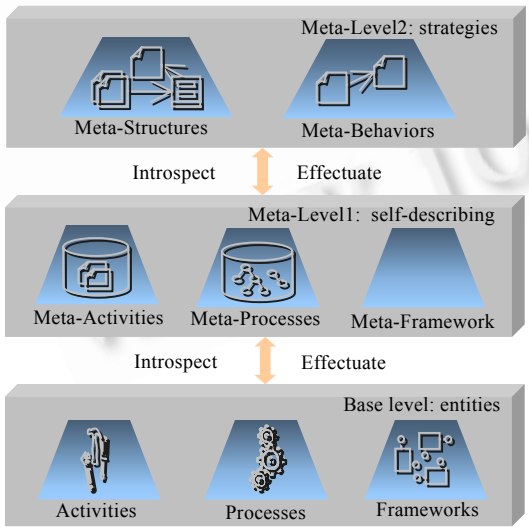

Fig.5 Reflective architecture for dynamic processes

图 5 支持动态流程的反射体系结构

SCM 中的反射体系结构如图 5 所示,那些处理自描述的实体和面向应用的实体一般分别处于两个不同的层次,即元层(meta-level)和基层(base level).元层又可以进一步细分为元一层(meta-level1)和元二层(meta-level2),前者主要专注于所有实体的自表达(self-representation),后者则主要内省(introspect)更抽象的结构和行为策略.另外,在每层中又可以根据实体的种类,利用活动、流程、框架等兴趣平面(concern plane)作进一步的划分.

基层中主要包含目标系统为完成某任务而需要的协作框架、协作流程、协作活动,以及服务提供者、消息格式、约束条件、状态转换、模型转换等各种

应用实体.元层则主要是对这些实体的抽象和自描述,包括解释活动的元活动、说明流程的元流程、表达框架的元框架等.所有反射接口都以 WSDL 的形式统一表达,运行时系统能从中发现反射实体中的消息格式、端口类型等重要信息,并通过手工或者自动绑定到服务端口来完成实际的操作.

4.2 反射接口和一级实体

反射接口根据其能力的不同至少可以分为两大类.内省接口具有观察和推理自身状态的能力,如在 SOAP 动态绑定前查找服务提供者、端口类型、活动实现可用的操作及参数类型、消息格式等信息.有效化(effectuate)接口则具有修改自身解释的能力,这使得不中断系统执行和修改应用代码而直接动态重配置系统结构和调整行为成为可能.

一般来说,内省和有效化主要发生在基层中的部分实体身上,这些实体常称为一级实体(first-class entities),SCM 中主要用到的一级实体如图 6 所示,它们之间的关系恰好应用了 Type-Object,Property,Composite,Proxy,Strategy 等经典设计模式[5].

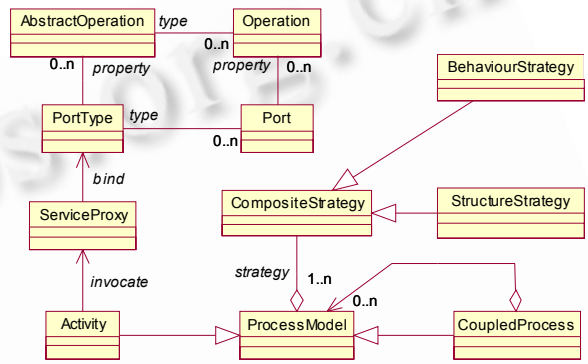

Fig.6 Reflection oriented first-class entities

图 6 面向反射的一级实体

绑定到服务提供者的几个活动的复合可以演进为更强有力的动态流程模型,它可以在运行时得到解释,以表现变更的行为.原子模型,即一个活动,就是到服务提供者某个端口的绑定.复合模型,即一个流程,则支持原子模型的顺序、并行、排他、选择等复合构造.复合模型包含到多个服务提供者的各种不同端口类型的绑定,并构成这些参与者之间的协作关系.

4.3 流程结构的动态重构

协作框架、流程和活动等实体,无论是原子模型还是复合模型,都具有统一的反射接口,因而,通过动态重构这些部件,流程就可以很容易地修改其内部结构和对外行为.

当流程 P^0=(State,Message,Condition,Γ,Λ,Ω)的结构需要变更的时候,即在某种条件 $c\in$Condition 下,例如服务已过期却没有更新,或者提供者不再租借其服务,那么流程中绑定到这些服务的活动 a 的运行状态 $s\in$State 就会失效.此时,动态流程 P^0 可以手工或者自动地转换到另一个不消费该服务的流程模型 P'=(State′,Message′,Condition′,Γ',Λ',Ω'),即发生Ω:$s\downarrow P'(c)$,并且从新模型 P'中发生模型转换的相同状态 s'(即 $s'=s$ 且 $s'\in$State′)开始执行.模型转换通过协作流程结构重构来响应外部或内部事件,并隐含某些状态的改变.

为保证转换前后 P^0 与 P'流程实例间的连续性,在此之前需要做的就是保存当前执行状态 s 和转换前后流程模型共用的消息 m 的值等信息.然后,通过元流程内省接口查找到满足具有相同转换接口等必要约束的其他合适的流程模型 P'后,就可以完成最终的Ω:$s\downarrow P'(c)$模型转换.如果不存在这样的流程 P',运行时系统则可以通过调用元活动内省接口去发现绑定到具有相同端口类型的不同服务提供者的其他活动 a'',用它替换当前失效的活动 a,构造出能够正常运转的新流程 P'',并将它保存到基层,以便进行模型转换Ω:$s\downarrow P''(c)$,并生成相应的元流程及反射接口,存放在元层中供以后重用.如果连这种可替换的活动 a''都找不到,系统还可以通过查找元层中的参与者、消息格式、端口类型、操作及参数类型等信息,主动构建绑定到可用服务提供者的新活动 a''',再将它复合成有效的新流程 P''',通过转换到新流程 P''',即Ω:$s\downarrow P'''(c)$, 完成预定的任务,同时在元层中增加相应的反射实体和接口.

概括来说,基层中的实体首先通过内省接口调用元层中的元实体,请求调整结构或行为;然后,元层实体处理这些请求,调整结构或行为策略;在完成之后,元层将修改的结构等返回给基层,并通过有效化接口应用这些变

更;于是,基层的结构或行为就随之变化了.

5 流程虚拟机与模型驱动的动态流程

5.1 流程虚拟机(PVM)

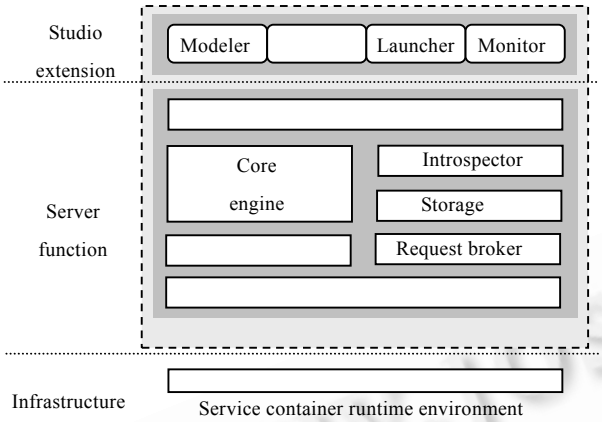

Fig.7 Architecture of process virtual machine

图 7 流程虚拟机(PVM)系统结构

基于服务协作的流程必须运行在一个业务流程管理的支撑环境中,该环境不仅能够设计协作流程和活动,而且要能以 WSDL 的形式包装成 Web 服务,并能为企业协作框架中的所有参与者自动创建相应的视图投影.

因而,我们基于 SCM 概念体系结构设计了一个流程虚拟机 PVM(如图 7 所示)系统,作为 SCM 的参考实现.流程模型直接由 PVM 控制执行,元数据及自适应的流程模型会在系统运行时进行更新,系统行为和结构会随着系统需求的改变而调整.PVM 建立在 SOAP 通信的基础之上,主要由服务器端功能和客户端扩展两大部分组成.

在客户端扩展中,Modeler 用于设计和维护面向服务的协作流程模型,Monitor 用于管理和跟踪服务器的状态,Wrapper 用于将流程模型组织成 Web 服务的形式,Launcher 用于自动部署服务和流程.

在服务器端功能中,Core Engine 管理所有协作流程和活动的执行;Request Broker 为便于外来参与者请求内部流程服务提供了必要的功能;Invocation Agent 提供了统一的调用机制,使各种应用都能以类似的方式进行调用;Introspector 基于内省和有效化接口为所有运行时的实体提供前面所述的反射能力;Storage 管理服务器端对象的持久化;Messaging Dock 基于底层的 SOAP 协议处理程序负责 SOAP 消息的编码、解码和通信等.

在客户和服务器之间的 Passport Authenticator 满足服务流程操纵的安全访问控制需求,它确信只有授权的成员才能存取框架中的服务和流程,保证了数据的安全性.

5.2 模型驱动的动态流程

为了真正实现动态流程这一目标,除了反射能力支撑之外,还需要有一套完善的模型驱动体系结构(MDA).MDA 是通过使用形式化模型来支持系统描述和互操作的一种方法,旨在通过把系统描述背后的逻辑与它的具体实现相分离,以保证构造的信息系统能够适应新的硬件和软件平台.该方法最显著的特点在于,系统描述是与具体的实现技术和平台无关的.系统定义独立地存在于任何实现模型之外,同时又具备到许多可能平台基础设施(如 Java,XML,SOAP 等)的形式映射.在 MDA 中,首先要以平台独立建模语言,如 UML 的形式,建立平台独立模型(PIM),然后,通过采用形式规则将 PIM 映射到某些实现语言和平台(如本文的 PVM 系统),把平台独立模型转换成平台相关模型(PSM)[11].

5.2.1 流程元模型建模

元模型建模是系统描述、建模或者元数据管理中的重要活动.异构环境中的互操作最终都通过元数据和元模型的共享和理解来达到,包括模型的自动演化、发布、管理和解释等.

为此,我们结合新推出的 UML2.0 标准,研究适合流程元模型建模的 UML 表示方法,并体现在 PVM 的 Modeler 功能之中.结构的变化可能涉及到模型的宏观规则,这部分以用例(use case)的形式表达;也可能涉及到模型的微观行为,这部分以状态(state)或活动(activity)图的形式表示;还可能涉及流程的参与者以及他们之间的交互,这需要时序(sequence)图和协作(collaboration)图来刻画.

新的结构和部件一旦产生就应该立即呈现于运行环境,因而显式的元模型建模工具 Modeler 与隐式的反射体系结构集成在一起,以方便在设计时构建动态流程模型和在运行时自动调整流程结构,前者如通过内省接口

观察元层可用的服务端口类型,后者例如通过有效化接口更新基层的模型结构和交互行为.

基于服务协作的流程必须运行在一个业务流程管理的支撑环境中,该环境应该不仅能够设计协作流程和活动,而且要能以 WSDL 的形式包装成 Web 服务,并能为企业协作框架中的所有参与者自动创建相应的视图投影.

5.2.2 从 PIM 到 PSM 的映射

在研究将平台独立的流程元模型映射到适合 PVM 解释执行的平台相关的流程模型的时候,我们对比了 XPDL,BPML,WSFL,XLANG,BPEL4WS 等各种流程定义语言.首先试验了 WSFL,发现该语言虽然在集成面向服务的自动型应用方面具有优势,但不利于集成 EAI 和 B2Bi 中普遍存在的人工型活动,尤其在 WorkList 任务处理方面非常不便,所以,最终选择了 WfMC 沿用至今又扩展了 Web 服务支持的 XPDL 语言.

Modeler 可以将类似于 UML 图的可视化模型映射为 XPDL,并由 Wrapper 包装成 WSDL,再由 Launcher 部署到用 Java 编写实现的 PVM 执行环境中去,从而完成从 PIM 到 PSM 的映射,使文本流程(plain process)成为可执行流程(executable process).PVM 技术通过对可执行流程模型运行时的解释,提供动态的系统结构和行为.

5.2.3 运行时的动态流程

在设计时,我们采用元模型建模工具基于控制流语义,捕获和表达了流程的结构及变化,但在运行时则需要依靠 PVM 的动态控制执行能力,借助反射接口,通过 SOAP 通信机制,动态绑定到服务提供者,使由简单语法复合起来的流程部件真正具有运行时的生命力.

PVM 的核心功能就是将抽象行为动态绑定到实际服务提供者.在设计时,行为可以是活动中的一个功能操作,或者是对流程实例的一个生命期调用.一旦处于运行时态之后,那些行为就必须绑定到具体的服务实现.基于 SOAP 调用的绑定可以不是静态的,但应能随着业务流程的调整或者服务提供商的演变动态适应外界的变化,绑定到运行时真正可用的提供者.因而,面向服务的动态流程的本质就是找到流程模型和服务提供者之间的动态绑定.

5.3 动态协作流程应用的场景

在业务流程通过 PVM 相互协作的应用中,我们发现由流程服务请求和提供者交互形成的业务流程,在支撑系统的动态改变机制的支持下,为了完成业务活动,会动态组织,形成适应不同应用需求的拓扑结构.

我们提炼出几种典型的 PVM 拓扑结构,如图 8 所示,分别适合于 EAI,B2Bi,Pub/Sub,P2P,P2P Cluster 等不同企业分布式计算场景.文献[12]详细介绍了这些流程应用场景各自的优缺点.

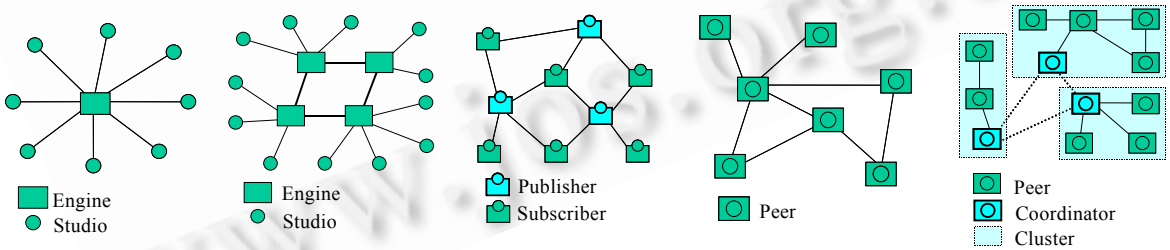

Fig.8 Diversified application scenes

图 8 流程应用的多样性场景

6 结论与未来工作方向

本文基于 Web 服务技术,提出了新一代服务协作中间件(SCM),建立了协作流程的元模型,在此基础上发展出动态流程的概念,并通过 SCM 的反射特性支持运行时动态流程的结构重配置等自适应行为,最终设计实现了流程虚拟机(PVM)系统,能够通过模型驱动方法,动态控制执行流程,从而适应环境的变化而自动调整自身.建立和运行在此系统之上的动态流程可以应用于更开放、更多样化的业务流程管理场景中.

流程与服务结合是目前 BPM 发展的现状.面向服务的流程建模和基于业务流程的服务开发应该同时包含在大规模企业项目的开发中,以提供更加良构的、可重复的解决方案.随着行业应用经验的不断丰富,从中提取出一定的领域模型形成标准和规范,以便更好地向用户提供优质的流程和服务也将成为可能.而动态流程的长远目标是将流程模型设计为一种演化的自适应模型并广泛部署,使它们能够在更大范围内、更深层次上进行通信、交互、协作和自组织,并允许智能化流程在运行时形成动态联盟,从个体简单的自适应规则出发,演化出更加丰富多彩的流程群落,以适合各式各样的场景.以上几个方面是我们未来中长期的工作.

References:

[1] Chappell D, Jewell T. Java Web Services. O'Reilly Press, 2002.

[2] BPMI. Business process modeling language (BPML). 2001. http://www.bpmi.org/bpml-spec.esp

[3] Feng YL, Huang T, Jin BH. Network distributed computing and software engineering. Beijing: Science Press, 2003 (in Chinese).

[4] Liu SH, Wei J, Xu W. Service cooperation middleware and its support for process integration. In: Han YB, Shi ML, eds. Proc. of the Int'l Workshop on Grid and Cooperative Computting (GCC 2002). Beijing: Publishing House of Electronics Industry, 2002. 614~625.

[5] Liu SH, Wei J, Xu W. Towards dynamic process with variable structure by reflection. In: Williams M, ed. Proc. of the 27th Annual Int'l Computer Software and Applications Conf. (COMPSAC 2003). Dallas: IEEE Computer Society, 2003. 120~125.

[6] Liu SH, Wei J, Xu W. Modeling decentralized adaptive process based on the service cooperation middleware. In: Ertas A, ed. Proc. of the 7th World Conf. on Integrated Design and Process Technology (IDPT 2003). Austin: Society for Design and Process Science, 2003. 765~771.

[7] de Bakker J, de Vink E. Control Flow Semantics. Cambridge: MIT Press, 1996.

[8] Barros FJ. Modeling formalisms for dynamic structure systems. ACM Trans. on Modeling and Computer Simulation, 1997,7(4): 501~515.

[9] Uhrmacher AM. Dynamic structures in modeling and simulation: A reflective approach. ACM Trans. on Modeling and Computer Simulation, 2001,11(2):206~232.

[10] Costa FM. Combining meta-information management and reflection in an architecture for configurable and reconfigurable middleware [Ph.D. Thesis]. Lancaster University, 2001.

[11] Object Management Group (OMG). Model-Driven architecture. 2001. http://www.omg.org/mda/

[12] Liu SH, Wei J, Xu W. Diversifying and improving business process management systems by service cooperation middleware. In: Lin HM, Ehrich HD, eds. Proc. of the 3rd Int'l Conf. on Quality Software (QSIC 2003). Dallas: IEEE Computer Society, 2003. 396~399.

附中文参考文献:

[3] 冯玉琳,黄涛,金蓓弘.网络分布计算和软件工程.北京:科学出版社,2003.