中国科学 E辑：信息科学 2008 年 第 38 卷 第 4 期：504 ~ 518
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE IN CHINA PRESS

软件服务多模式交互中间件模型及其支撑技术

陶先平[①②*]，马晓星[①②]，吕建[①②]，余萍[①②]，周宇[①②]

① 南京大学计算机软件新技术国家重点实验室，南京 210093；
② 南京大学软件研究所，南京 210093
* E-mail: txp@ics.nju.edu.cn

收稿日期: 2007-09-24；接受日期: 2007-11-11
国家重点基础研究发展规划(批准号: 2002CB312002)、国家自然科学基金(批准号: 60403014)和江苏省自然科学基金(批准号: BK2006712)资助项目

摘要 软件服务在开放环境下具有独立性、多样性、裁剪性等特点，支持软件服务的多模式交互中间件的研究尤为重要. 提出了一种基于 agent 的软件服务多模式交互中间件模型. 该模型包括一个支持交互模式编程的层次式交互特征分解/综合模型、基于 agent 的中间件结构模型和基于自省技术的可编程协同媒体. 其中，交互特征分解/综合模型可通过交互特征的分解和综合支持程序人员进行多种已有交互模式或新模式编程；基于 agent 的中间件结构模型为多模式交互运行支撑提供了良好的基础框架；可编程协同媒体可有效支持基于多模式交互的软件服务协同. 为验证上述方法的可行性和效率，给出了自行设计和实现的软件服务多模式协同中间件 Artemis-M3C 环境概况，并进行了实例测试及性能分析. 结果表明上述方法是可行的，系统在多模式交互方面具有一定的实用性和效率.

关键词
开放环境
多模式交互
移动 agent
交互模式编程
可编程协同媒体

随着Internet的快速发展与普及，如何在开放、动态、难控的网络环境下实现各类资源的共享和集成已经成为计算机软件技术面临的重要挑战之一. “软件服务”作为一种新的软件实体，承担着封装各种资源、完成计算并对外提供服务的任务，将逐渐成为网络应用软件的基本组成成分. 具体而言，这些软件服务封装了开放环境下的各类数据、信息、软件资源，以自主、开放的方式存在于开放的Internet环境中，通过某种方式加以发布. 基于软件服务的网络应用软件开发和运行将体现为这些软件服务之间跨网络的功能互通、组织联盟、多模式互连、自主协作的过程，从而最终完成网络资源的集成和共享 [1]. 然而，开放环境下的软件服务具有独

立性、多样性、裁剪性等诸多特点, 也给面向软件服务计算带来了困难: 1)独立性: 软件服务的开发、运营、集成都是由不同机构独立完成的. 这种独立性使得“只能使用、不能随意修改”成为软件服务的一大特点. 因此, 如何在运行时满足最终用户的功能和性能需求成为问题. 2)多样性: 由于独立性的存在, 软件服务呈现出复杂的多样性, 比如软件服务的建模方式、交互模式、通信语言、程序语言、开发/运行平台等. 软件服务的多样性使得软件服务之间的互操作以及多模式交互异常重要. 3)裁剪性: 开放网络环境下, 软件服务的开发较难基于一个完备的信息和资源集合, 因此, 相对于特定应用需求, 软件服务可能同时会出现“有求于人”(功能不完整)以及“能力过剩”(功能过多)等现象. 前者使得软件服务必须和其他软件服务开展有效合作; 后者使得软件服务可能需要进行功能的消减.

上述分析说明, 软件服务集成在面向客观应用需求时, 多模式交互、功能动态裁剪是必须面对的困难. 而要解决上述困难, 其焦点之一在于支持服务计算的中间件技术及产品.

在网络环境下, 软件实体之间的交互通常是由中间件完成的. 然而, 传统的中间件系统大多产生于静态、封闭、可控的环境, 面向开放环境中的软件服务交互, 在支持上述多模式交互和动态功能裁剪等方面均存在不足, 因此经典的软件中间件技术难以适应开放环境对软件协同的要求.

分析目前较为流行的中间件模型, 如RPC(remote procedure call)[2], CORBA(common object request broker architecture)[3], Jini/JavaSpaces[4]等, 可以发现, 一个中间件包括两个侧面: 其一是面向程序员的编程模式, 如远程过程/方法调用等交互表达手段; 其二是用于实现实体交互运行支撑机制, 如 Stub, Proxy, Skeleton等一组代理. 从开放环境的角度看, 常用的中间件模型具有两方面的不足: 其一是交互模式的单一固定性, 即一个中间件通常只能支持一种交互编程模式, 要么是紧耦合的远程过程/方法调用, 要么是基于共享空间的松耦合交互等; 其二是代理功能的固定僵硬性: 与交互模式的单一固定性相对应, 中间件中各个代理的形式和功能通常是确定的, 不会随应用系统需求的变化而做相应的调整. 而在开放网络环境下, 我们通常希望能够做到以下两点: ①能够根据环境的具体情形和特定应用系统的需求, 对交互模式进行一定的设计和定制, 从而不仅能够支持交互模式的多样性, 而且可支持多样性的交互模式在同一应用系统中的共存. ②能够将中间件支撑系统中的代理的形式和功能一般化, 不仅能够根据所设计与定制的交互模式对其加以具体化, 而且可根据交互模式的要求对相应的软件实体进行一定的剪裁和增强.

基于上述思路, 本文提出了一种基于 agent 的多模式协同中间件模型并给出了中间件系统 Artemis-M3C 的设计和实现. 在面向程序员的编程模型方面, 基于异质软件服务的交互模式特征分析的基础, 构建了一个交互模式配置模型, 为开放网络环境下软件服务所展现的各种协同模式的定义提供一个统一的技术框架; 在交互运行支撑方面, 采用 agent 的概念和技术来统一和拓广各种既有软件协同媒体的本地代理接口, 又可利用自省技术有选择地开放协同媒体的内部实现, 从而使得用户的协同 agent 可以有序地定制协同媒体的行为. 基于该方法的中间件系统 Artemis-M3C 的试验和运行分析结果也表明该方法的可行性和系统在支持异质软件服务的多模式协同方面具有的良好实用性和效率.

本文余下部分组织为: 第 1 节介绍中间件交互编程的交互特征分解及模式配置模型; 第 2 节介绍基于 agent 的自省可编程协同媒体设计思想; 第 3 节介绍 Artemis-M3C 的设计、实现及其性能测试分析; 第 4 节讨论相关工作; 最后是全文总结和工作展望.

1 基于交互特征分解/综合的交互编程模型

软件实体之间一次交互不论采取什么方式, 一般都是表现为交互请求方和交互服务方之间通过某些交互媒体的信息交换, 只是在不同的应用场景和运行环境下, 其交互规程、交互模式等有所不同, 但其交互基本模型可以抽象表达为图 1.

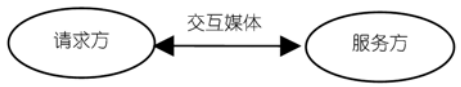

图 1 交互基本模型

比较常见的交互模式有RPC模式[2]、Tuple Space模式[5]、Meeting模式[6]等. 这些交互模式通常情形下总是隐含有软件实体开发时刻的编程方式和交互时刻的交互规程及其基础设施. 在现行中间件技术支撑下, 客户端或者服务端软件实体的开发一般总是基于其中某个交互模式并只能和同模式的其他实体进行交互, 因而每个服务只能遵循单一模式交互, 而不支持同一系统中有不同交互模式服务的存在. 然而在开放、动态的网络环境下, 网络软件的开发越来越多地倾向于基于“先服务、后实体”理念的服务集成. 这种思想提倡“关注分离”原则, 要求服务设计和开发独立于交互模式, 强调服务互连时进行交互模式的配置, 通过交互模式的编程体现不同的环境需求, 从而支持一个服务的多种模式交互.

实现上述需求有若干种途径, 其中一种是对可见交互模式进行“枚举匹配”, 为每两种交互模式的转换设计一个转换器. 可以看出, 这种方式比较笨拙, 并不科学: 一来转换器数量相当可观, 二来一旦出现新的交互模式, 处理非常繁琐; 另一种选择是运用“关注分离”原则[7,8], 对交互本质进行结构化研究, 进行各种模式的基本成分分解, 在一致性和完整性等原则的指导下给出一个通用的交互编程模型, 研究其和各类已有交互模式的转换, 从而可以在软件服务集成时进行交互模式编程. 从静态观察角度, 交互模型实质上即为该模式的各特征描述及其固化组合; 从动态运行角度, 交互实现则体现为运行时刻上述各特征在实现载体中的处理. 进而, 一个交互编程模型可以是由一个交互特征模型、一个配置方案以及一套运行支撑所构成, 不同的交互模式区别在于上述三者的不同. 显然, 上述基于“关注分离”的“结构化”途径是一种可行方法, 能够有效支持软件服务之间的多模式交互.

1.1 交互模式定义

从交互的基本模型(图 1)可以看出, 请求方、服务方和交互媒体构成了交互的 3 个侧面, 而从更细节的交互特征分析, 上述 3 个侧面是正交的, 且可以再分解为若干子特征的组合, 比如请求方侧面中包含服务方规约、非功能性约束、服务裁剪等, 而这些子特征又具有正交特性并可以再分解直至原子特征. 基于上述交互特征分解/综合思想, 本文给出了基于 XML Schema 的交互模式定义框架, 用户在进行服务集成时, 可以根据应用需求、环境状况等特性, 对交互中的请求方、服务方和媒体等侧面进行分解, 在该框架下直接进行交互编程.

交互模式定义框架.

```
〈?xml version="1. 0" encoding="UTF-8"?〉
〈xs:schema xmlns: xs=http://www. w3. org/2001/XMLSchema
    xmlns="http://ics. nju. edu. cn/interaction mode/schema"
    targetNamespace="http://ics. nju. edu. cn/interaction mode/schema"
    elementFormDefault="qualified" attributeFormDefault="unqualified"〉
   〈xs:element name=" Interaction Mode "〉 〈xs:complexType〉 〈xs:sequence〉
      〈xs:element ref="Aspect" minOccurs="1" maxOccurs="unbounded"/〉
   〈/xs:sequence〉 〈/xs:complexType〉 〈/xs:element〉
   〈xs:element name="Aspect"〉 〈xs:complexType〉 〈xs:sequence〉
      〈xs:element ref="Compound Feature" minOccurs="1" maxOccurs="unbounded"/〉
   〈/xs:sequence〉 〈/xs:complexType〉 〈/xs:element〉
      〈xs:element name=" Compound Feature"〉 〈xs:complexType〉 〈xs:sequence〉
      〈xs:element ref="Feature" minOccurs="1" maxOccurs="unbounded"/〉
   〈xs:element ref="Compound Feature" minOccurs="1" maxOccurs="unbounded"/〉
   〈/xs:sequence〉 〈/xs:complexType〉 〈/xs:element〉
      〈xs:element name="Feature"〉 〈xs:complexType〉
      〈xs:attribute name="Name" type="xs:string"/〉
      〈xs:attribute name="Type" type="xs:string"/〉
      〈xs:attribute name="Parameter" type="xs:string"/〉
      〈xs:attribute name="Feature Procedure" type="xs:string"/〉
   〈/xs:complexType〉 〈/xs:element〉
〈/xs:schema〉
```

其中, 一个交互模式是由一个或者多个交互“侧面”构成, 每个“侧面”由一个或者多个复合“特征”构成, 每个复合“特征”由一个或者多个复合/原子特征构成, 原子特征是指无需继续分解, 可以直接表达其名字、类型、具体参数以及相应处理方法的交互因素, 如限定 1 s 内返回结果的限时同步要素可以表达为〈TimeLimit, int, 1, TimingProc〉. 原子特性定义中的 TimingProc 指一个计时处理程序, 该程序将由交互分解人员提供.

1.2 交互特征分解/综合模型

以上给出的是一个交互模式的 XML Schema 定义框架, 并未涉及一个复杂的交互模式的特征分解过程以及从各原子特征进行综合并形成一个交互模式的过程. 为给出一个指导性的交互模式分解/综合方案, 基于上述定义框架, 本文以实体配置为中心, 采用多维空间模型, 给出了一个层次式多维交互特征分解/综合模型. 服务集成过程中, 可以参照该模型, 在可视化工具(如 Artemis-M3C)支持下, 进行交互特征的分解、组织、配置、综合和描述.

需要提出的是, 该模型并没有对可能的侧面以及特征穷尽枚举. 原因一是分解相对而言是没有确定性的, 侧面以及特征的划分并非唯一; 原因二是这个集合并非是有限集合, 随着人们对交互的理解的深入以及应用需求的不断变化, 需要关注的侧面和特征可能会不断增加.

由此看来, 穷尽枚举是很困难的甚至是不可能的. 因此, 从另外一个方面, 本文试图给出的是一个交互特征分解思想以及一个基于多维空间的开放、可扩充的分解模型. 这样, 之后需要添加的特征可以通过添加维或者维度上值的方式加入到模型中.

同样基于上述分析, 可以给出分解/综合模型的基本建模准则:

1) 开放性: 模型的结构必须是开放的, 能够有效支持各类特征的表述及其组合, 能够支持新特征的加入;

2) 统一性: 模型必须是统一的, 能够表达各种交互模式特征, 能够通过配置、组合, 定义各类已有和新型交互模式;

3) 一致性: 特征的分类必须满足正交要求, 符合“关注分离”的基本原则; 各特征的参数确定必须满足一致性, 不能破坏相应交互模式的语义要求.

4) 完整性: 在通过特征组合及参数配置进行某次交互的模式编程时, 必须满足特征和参数的完整性, 符合该交互模式的语义要求.

限于篇幅, 图2仅给出本文提出的交互特征分解/综合模型的第2层及其简单语义说明, 详细信息请参见技术报告[1].

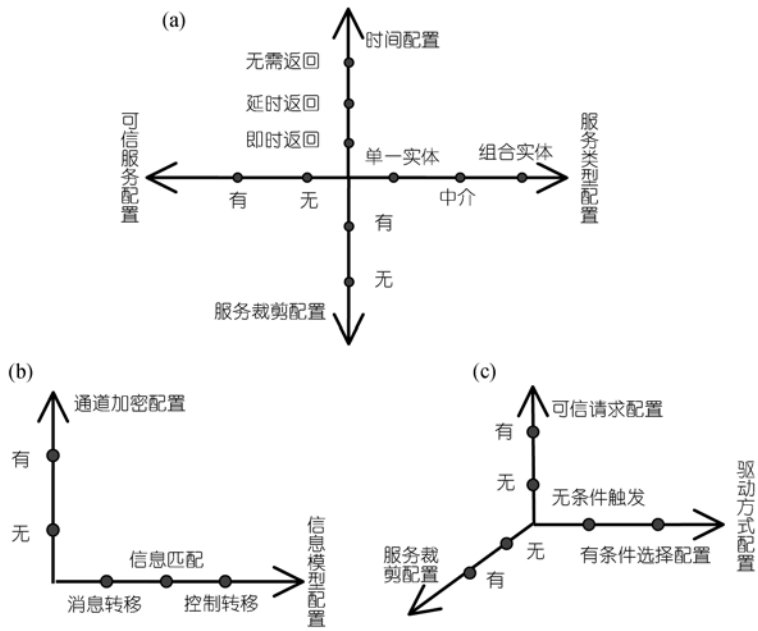

图 2　交互特征分解模型

(a) 请求方配置模型; (b) 媒体配置模型; (c) 服务方配置模型

在请求方配置模型中, 服务类型配置指明了服务的类型和具体的配置建议, 时间配置指明了交互的同步/异步特性, 可信服务配置指明了请求方在该交互中是否要求服务方提供可信计算保障(如果需要, 则集成人员必须给出可信计算评估策略), 服务裁剪配置指明了交互中是

1) 基于 agent 的多模式协同中间件技术报告. 南京大学计算机软件新技术国家重点实验室, 2004

否需要对服务方进行服务的裁剪(如果需要, 集成人员必须给出裁剪策略).

在媒体配置模型中, 信息模型配置指明了媒体进行的是以MP(message passing)[9]为代表的信息转移、以Space为代表的信息匹配(在此情形下, 集成人员必须给出匹配规则), 还是以agent为代表的控制转移(在此情形下, 运行环境在运行时刻将创建一个执行agent), 通道加密配置指明了交互信息是否加密通信.

在服务方配置模型中, 驱动方式配置指明服务方在收到请求方请求后是否有条件开始服务(在此情形下, 集成人员必须给出驱动策略), 可信请求配置指明服务方是否对请求者有可信计算要求(在此情形下, 集成人员必须给出可信保障策略), 服务裁剪配置指明是否需要对服务功能进行裁剪(在此情形下, 集成人员必须给出裁剪策略).

从软件服务的交互编程角度, 集成人员可以在上述分解/综合模型指导下, 将交互模式进行分解, 以配置的方式针对每个特征进行编程, 给出其特征内容以及相应处理规程, 完成交互编程. 上述所有配置信息及策略信息将被运行支撑中间件所收集并直到运行时刻被运行环境所处理, 以实现多模式交互.

2 基于 agent 的多模式交互中间件

上节讨论支持多模式交互编程的交互特征分解/综合模型. 本节介绍多模式交互中间件模型及其运行支撑机制.

2.1 静态结构

现有中间件产品从结构上均呈现一种“请求—代理—服务”架构, 其中的“代理”承担了软件实体之间的关联工作, 如 Corba 的 Stub/Skeleton, Tuple Space 模式的 Space, 消息中间件的消息队列, Jini 的 Lookup 服务等. 然而, 目前的“代理”仅能提供针对特定交互编程模型的运行支撑能力, 较难支撑不同模式的软件服务之间多模式并存的交互. 然而, 上述不足并非中间件结构所致, 究其原因, “代理”能力的不足是根本原因.

为此, 我们借助软件agent的移动性、自主性和私有协议封装能力 [10,11], 一般化协同中间件的“代理”概念, 统一和拓广各种既有软件协同媒体的代理(如Stub/Skeleton)的能力, 使用移动agent技术为运行时刻提供支撑. 在此概念下, 提出一种基于agent的软件服务多模式交互中间件概念框架如图 3.

该中间件框架中, 协同守护 agent 和多模式协同媒体构成了中间件的核心.

2.1.1 协同守护 agent(SCA)

从交互特征分解及综合产生交互模式过程看, 特征配置项选择、参数的确定及其处理程序决定了交互方式, 因为配置项、参数和处理程序均独立于服务提供商和运营商, 软件服务运行前, 其驻留节点的协同媒体并不能得知具体参数及其处理方法, 必须在运行前为其提供. 为此, 针对每个服务, 将该服务的交互模式编程信息及相关处理程序封装到 SCA 中, 在运行前被派驻到各个被集成的软件服务节点, SCA 和软件服务一一对应. 运行时刻, 软件服务并不直接与其他软件服务进行交互, 所有交互和协同行为均由封装了交互模式信息的 SCA 控制, 由它根据协同具体上下文信息和其他 SCA 合作完成. 具体的软件服务将只接受它的 SCA 指令,

完成自己的功能. 在此概念下, 现有的各类交互模式中起重要作用的“代理”(如 Corba 的 Stub/Skeleton、Tuple Space 模式的 Space、消息中间件的消息队列、Jini 的 Lookup 服务)均可视为 SCA 的某种特例. 如此, 多个 SCA 之间构成了一个良好的交互环境, 可以很好地屏蔽软件服务的多重异构特性, 为软件服务的多模式交互提供了一个统一的中间件结构模型.

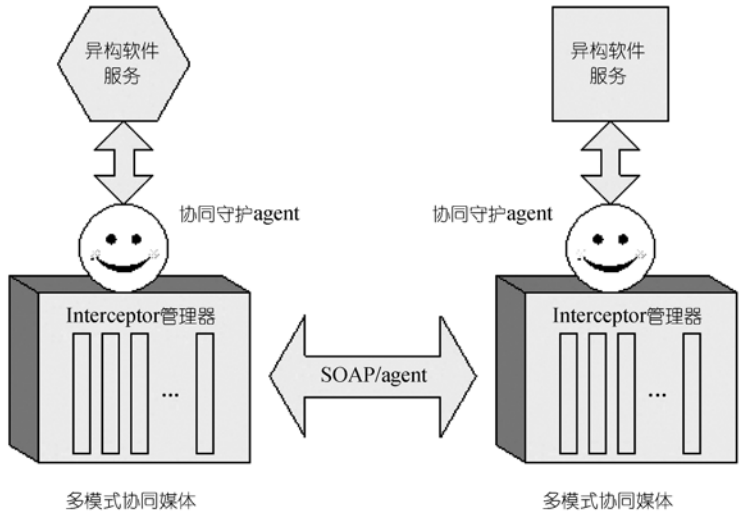

图 3　基于 agent 的软件服务多模式交互中间件概念框架

2.1.2　多模式协同媒体

上述概念模型需要得到协同媒体的支撑. 从支撑角度讲, 因为配置项、配置参数独立于服务提供商和服务运营商, 软件服务运行前, 其驻留节点的协同媒体并不能得知具体参数及其处理方法, 只有在运行时刻才能获得这些信息, 因此, 为有效支撑多模式交互, 参与协同支撑的所有协同媒体必须开放, 支持可编程, 使得能够通过媒体的可编程来动态改变媒体行为, 从而达到动态调整交互行为的目的.

支持交互多模式的可编程协同媒体的实现可以借助自省中间件技术 [12,13], 增加协同媒体的元接口, 允许通过元接口(meta-interface)来动态地改变协同媒体的行为. 具体而言其设计思想如下:

● 将交互特征配置及其参数具体化(reify)为一(元)数据对象, 将配置的参数处理规则具体化为元程序. 元程序通过元接口“上载”进入协同媒体中, 成为协同媒体中关于元数据对象的解释器.

● 协同媒体通过元接口截获上述表示交互动作具体配置的数据对象, 通过解释器执行, 对其进行操作, 并可能生成一组新的交互动作数据对象, 提交下一个解释器执行.

● 基于截获者(Interceptor)设计模式 [14], 组织各个配置项参数解释器, Interceptor的串联执行实现一次交互的多配置项组合效果, 并通过Interceptor模式的结构性限制动态上载的解释器对服务器资源的访问, 缓解因上载带来的安全问题.

基于上述分析, 借助自省中间件技术和 Interceptor 设计模式, 本文设计并实现了如上图所

示的基于 agent 的可编程协同媒体 Artemis-PCM(programmable coordination media).

如上所述, SCA 封装了该服务参与的所有交互的特征配置数据及其处理方法, 每个特征处理方法组织为一个 Interceptor. 当 SCA 迁移到服务所在节点后, PCM 在部署时刻将 SCA 中配置信息及 Interceptor 写入 Interceptor Manager 中, 运行时刻将会根据交互标识对各配置信息依次进行处理, 从而准确、完整地支持通过交互特征分解/综合形成的各类交互模式.

2.2 运行机制

PCM 作为协同应用的核心主体, 作为单例类存在于应用服务器中, 生命周期完全依赖于应用服务器的生命周期. PCM 中的 Interceptors 依赖于动态到达的 SCA.

运行时刻, 服务代理对客户端应用和服务提供了简单一致的基于消息的接口. 客户端软件实体发出的请求行为由服务代理封装, 被 PCM 代理截取, PCM 代理在识别请求后将请求交给 Interceptor 管理器, 按照该请求的配置情形各 Interceptor 依次对请求对象进行处理, 产生一个基于 MP 的 soap 消息或者一个基于控制转移的执行 agent, 向服务端 PCM 发送(如图 4 所示). 服务器端执行机制类似于客户端. 如此, 一次特定配置的交互将如实被客户端/服务端 PCM 解释执行.

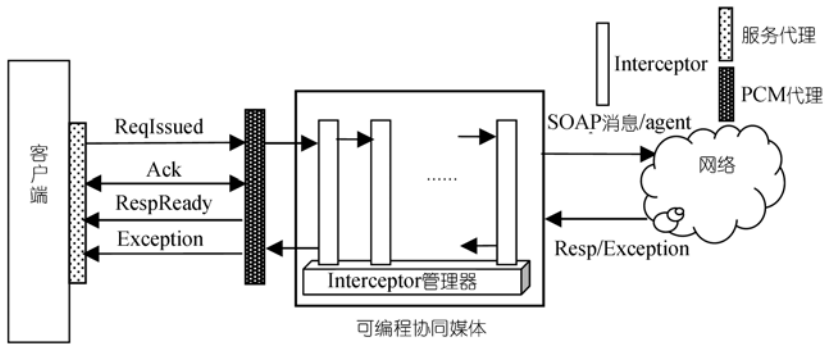

图 4 基于 agent 的可编程协同媒体运行示意图

从以上基于 agent 的 PCM 静态结构及运行机制可以看出, 该协同媒体不仅能有效支持可配置自定义交互模式的运行, 并且具有足够的能力为交互模式的演化提供运行支持.

3 中间件 Artemis-M3C 实现及性能分析

中间件系统包含运行支撑环境和集成开发环境两个部分. 其中, 运行支撑环境包括可编程协同媒体以及其间的连接器、agent 生产、管理、运行环境以及系统监控部分; 集成开发环境通过图形界面为软件服务集成开发人员提供了查找/集成软件服务、配置服务间协同行为规则、管理/运行集成系统以及运行时刻监控等功能.

3.1 集成开发环境

集成开发环境(M3C-IDE)为用户提供了一套完整、易用的操作界面以完成服务集成系统开发、运行及维护等活动, 如图 5 所示.

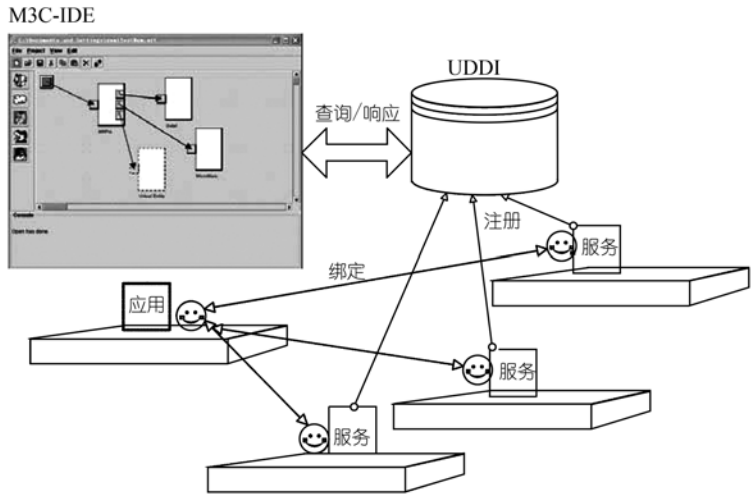

图 5　M3C-IDE 示意图

M3C-IDE 主要功能有:

● 基于 agent 的匹配服务查找. M3C-ID 可连接服务注册处按用户输入的查询需求搜索相关的已有服务, 并在界面上显示以供用户选择.

● 应用系统集成. 提供用户通过可视化手段完成服务互连以及交互的特征配置. 图中的点表示软件服务实体, 边表示服务协同关系. 通过编辑边的属性, 配置服务间交互特征.

● 协同部署. 配置结束后, M3C-IDE 将为每个软件服务创建一个 SCA, 并部署到相应节点的 PCM 中.

● 运行及监控. M3C-IDE 激活应用系统执行, 并可在监控界面显示各个服务实体所处运行平台发回的监控信息.

● 维护及撤销. 用户可对已有系统进行修改、并重新部署. 应用系统完成后, M3C-IDE 可撤销所有使用的服务实体上的属于本应用的 SCA, 并从本地协同环境中撤销应用程序.

3.2　M3C 运行支撑环境

出于对现有web service技术 [15]和应用服务器技术的兼容及标准的支持, 在运行环境中实现了可编程协同媒体与SOAP实现系统Axis的有机结合, 以支持软件服务间可配置的多模式协同行为.

基于 Axis 环境发布的软件服务的服务形态基本局限在基于 RPC 的服务调用方式, 不能完全满足我们对多模式协同以及底层通信环境的要求, 在 Axis 与请求实体以及 Axis 与服务实体之间分别插入一个可编程协同媒体(PCM). 请求客户端将请求送入 PCM 进行处理, 然后再转交 Axis 引擎按原有协议进行服务调用(如图 6 所示). 对于服务端, 为实施 SCA 中所定义服务行为, 必须在服务被调用前将请求送入 PCM 进行处理. Axis 中通过 Provider 完成对软件服务的调用, 例如 RPCProvider 实现了 RPC 调用模式. 我们扩充了 Axis 的现有实现, 加入了一个与 Axis 中原有的 RPCProvider 相应的 MMCProvider, 实现了 PCM 在服务器端的插入(如图 6

所示).

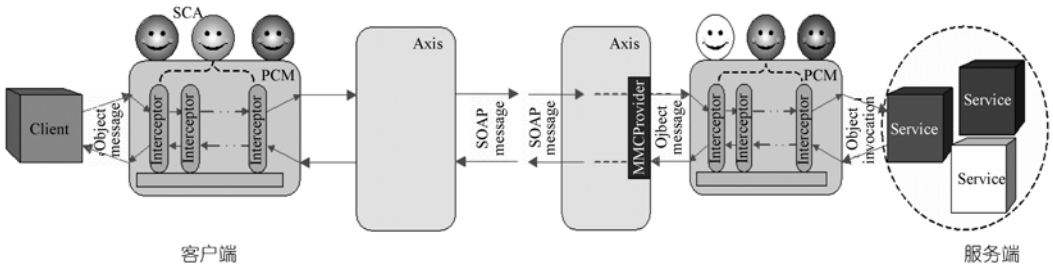

图 6 M3C 运行环境示意图

针对基于传统 RPC 模式的软件服务, 其部署在 Axis 中时, 显式表明其调用模式为“RPC”, 例如以下的名为 Calculator 的服务.

〈service name="Calculator" provider="java:RPC"〉
〈parameter name="allowedMethods" value="*"/〉
〈parameter name="className" value="samples. rpc. Calculator"/〉
〈/service〉

对此类服务的调用可以直接使用 Axis 中的 RPCProvider 接口, 针对通过配置自定义交互模式的服务, 其调用请求需要经由 PCM 处理, 部署到 Axis 环境中时, 需指明其调用模式为“MMC”, 例如以下的名为 Calculator2 的服务.

〈service name="Calculator2" provider="java:MMC"〉
〈parameter name="allowedMethods" value="*"/〉
〈parameter name="className" value="samples. mmc. Calculator2"/〉
〈/service〉

对此类服务的调用借助系统插入 Axis 中的 MMCProvider 接口, 将交互信息交付 PCM 处理.

3.3 性能分析

为了支持多模式交互, 本文给出了一种交互特征配置方法, 利用 agent 技术, 在 SOAP 中间件 Axis 中插入一个可编程协同媒体 PCM, 通过 PCM 对软件服务交互的运行时刻处理实现多模式交互. 无疑, 泛化的交互 agent 以及交互媒体 PCM 的介入将对软件服务的交互性能产生影响, 为评估这种影响, 本文进行了相应的性能测试和分析.

测试平台是两台 Dell OptiPlex GX400 PC, 内存 512 MBytes SDRAM, CPU 主频 1.4 GHz, 操作系统分别为 Windows XP 以及 Windows 2000 Server, 它们同处一个局域网内. 测试用例是一个旅游公司机票预定系统. 该系统由旅游公司机票预订服务 A 和航空公司航班查询服务 B(该服务平均耗时 1000 ms)构成. 运行时, A 为游客提供旅行订票服务, 在提供服务过程中需要请求 B 提供特定航班信息.

Agent 和 PCM 的介入必定会对 A 和 B 之间的交互产生性能影响, 我们选取“请求响应时

间”为代表进行上述交互性能影响分析, 考察在不同交互模式下、不同的 agent 和 PCM 介入程度下该值的变化及其规律. 本文进行了以下 4 种试验:

● 经典 RPC 模式: 将服务 A 和 B 均部署为 Axis RPC 服务. 该模式中, agent 和 PCM 均不介入 A 和 B 之间的交互, 由 Axis 支撑完成. 测试结果表明, 连续 100 次交互中, 支撑开销趋于稳定, 接近于服务时间.

● 自配置 RPC 模式: 将服务 A 和 B 均部署为 Axis MMC 服务. 交互模式配置时, 只进行简单服务绑定, 完全遵循经典 RPC 语义. 此时 agent 将被派发至 A 和 B 所在节点, PCM 将负责它们的交互. 但此模式下, PCM 内交互特征处理 Interceptor 序列为空. 连续 100 次交互的测试结果表明, 请求响应时间总体上有上升, 但很快趋向于一个能接受的稳定状态.

● 定时 RPC 模式: 将服务 A 和 B 均部署为 Axis MMC 服务. 在第 2 个试验基础上, 在客户端进行限时配置. 运行时刻, agent 将携带限时处理 Interceptor 进入 A 和 B 所在节点, PCM 将动态加载该 Interceptor 并负责两者交互. 连续 100 次交互的测试结果表明, 请求响应时间总体有上升, 但很快趋向于一个能接受的稳定状态.

● 可裁剪定时 RPC 模式: 在上例基础上, 将 A 的请求接口设为 Query(string start, string destination, long number), B 的服务接口设为 QueryPrice(int number, string start, string destination). 此时, 必须为服务端配置裁剪项, 实现两者接口、参数的转换. 运行时刻, agent 将携带裁剪 Interceptor 进入 A 和 B 所在节点, PCM 将动态加载该 Interceptor 并负责两者交互. 连续 100 次交互的测试结果表明, 此时 PCM 开销虽有所增加, 但也趋向于一个能接受的稳定状态.

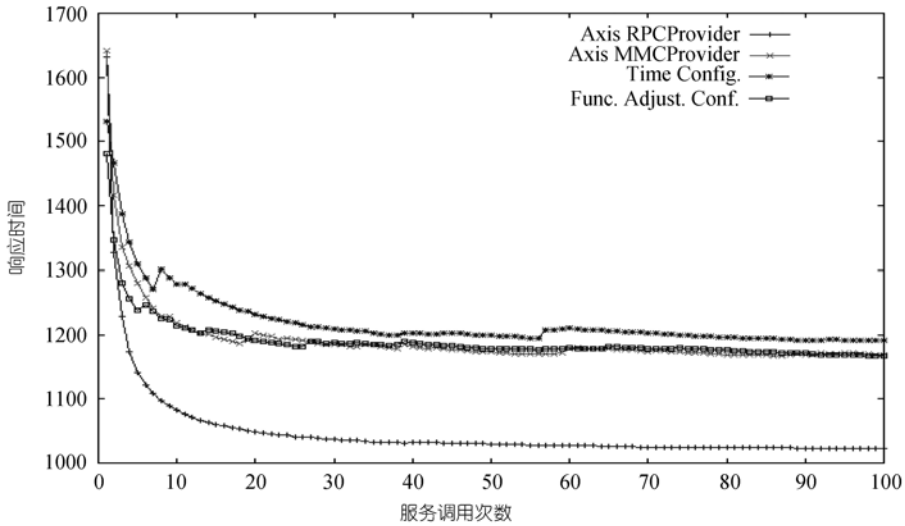

图 7　PCM 性能测试分析图

从上述多个试验中可以看出, 为支持软件服务之间的可配置多模式交互, 携带配置特征处理程序的移动 agent 以及动态加载了各类 Interceptor 的 PCM 确实给交互支撑带来了额外的开销, 但是这种开销的增加是有限的、稳定的, 相比它为多模式交互提供的支撑能力, 开销是可以接受的.

4 相关工作

随着Internet应用的覆盖面和规模的扩大, 软件协同技术和平台的研发越来越受到关注. 软件协同涉及面非常广泛, 如软件体系结构 [16~18]研究、语义网络 [19]研究、基于信任的可信计算 [20, 21]研究等. 本文工作主要侧重从协同模型、交互模式及其支撑角度开展多模式软件服务协同研究.

目前可见的协同模型有多种, Papadopoulos等人 [22]将其分为两类: 数据驱动协同模型以及控制驱动协同模型. 其中, 控制驱动协同模型主要指协同实体基于接口调用完成交互动作. 而数据驱动协同模型主要指协同实体之间共享一个数据空间, 协同实体间通过交换数据进行交互. 与此同时, 随着agent技术的发展, 上述模型在agent协同领域也有了更多的发展, 如BlackBoard模型 [23]、Meeting模型 [6]等, 而Giacomo Cabri, Letizia Leonardi等人还从交互耦合度角度对各种agent协同模型进行了系统分析和总结 [24]. 上述工作在不同的角度针对不同的特定模型进行了研究, 但是并没有涉及到各个模型之间的转换或者互操作, 不支持软件服务之间的多模式交互.

随着"软件作为服务"趋势的日益明朗, SOA/SOC[15]中的服务协同和交互逐渐成为研究热点. 在现有的WS技术如SOAP, WSDL, UDDI, BPEL等工作基础上, 学者就web service交互方式单一, 计算/协同混杂等不足进行研究, 如Lucchi等人 [25]结合数据驱动协同模型, 设计并实现了一个基于共享空间的web service协同环境; Lemniotes等人 [26]针对web service协同中暴露出的计算、协同和通信相互混杂从而灵活性不强的问题, 提出了一种基于通道(channel)的协同语言pεω, 该语言可以帮助建立和管理web service之间的协同connector, 每个connector是若干互连的通道通信子. 这样, 通过将web service协同集中于通道管理中, 分离web service的计算、协同和通信. 然而上述工作虽然针对web service协同的不足提出了一定的改进, 但是这些方案大多限于局部, 并不是一个普遍意义下的web service多模式交互的处理方案.

与此同时, 随着人们对软件协同中行为和交互两者的"关注分离", 交互、交互规约及交互管理的研究也渐渐兴起. 其常见做法是对交互进行分析, 给出形式化交互规约并通过一个合适的交互媒体确保交互规约的执行. 在这些工作中, Minsky等 [27]针对agent交互行为、交互约束提出了一个LGI(law governed interaction)框架. 该框架中, agent交互约束显式独立于agent功能代码部分, 这些交互约束、策略被分布于相应agent所在节点, 运行时刻agent的交互行为被一个称为Controller的机构所管理, 该机构通过检查交互行为和交互约束、交互策略的相容性决定该交互是否可以真正被提交, 同样, 交互结果也必须在通过检查后被返回源agent. LGI以一种分布式控制机制, 实现了交互约束、策略的强制执行, 保证了交互的可靠性. 虽然LGI的策略分离、分布控制思想为软件协同研究提供了一种借鉴, 但是LGI主要针对交互行为的分布控制, 并未真正实现交互约束的可编程、交互模式的可配置, 其管理机制也不是一个通用的协同媒体, 因此实现普遍意义下的软件服务多模式协同尚有不足. 此外, Stefano等人 [28]在软件agent协同研究中提出了一种将agent的行为和交互进行分离, 分别进行设计、实现的方法. 具体到交互而言, 这种方法支持agent交互的规约化, 并自动从中生成含有交互约束管理、检查的代码, 这些代码将在agent交互的运行时刻被执行以检查agent的交互行为是否符合交互设计中指定的约束或协议. 上述工作支持agent协同和计算的分离, 可以帮助实现agent的交互管理, 但

是该工作并没有对交互的内部因素进行系统分析, 采取的方法只是为交互动作“附加”约束, 显然不能支持灵活的多模式交互. 除此以外, 其他类似的工作还有Dimitoglou等 人[29]提出的middleware++, Mamei等人[30]提出的TOTA中间件模型等.

协同媒体是软件协同的重要支撑设施, 自Maes等人[12]提出基于自省的可编程中间件后, 可编程协同媒体的研究已经成为了近几年软件协同支撑的研究热点. 如Blair等人[31]提出的自省的对象请求代理系统OpenORB、Minsky等人[27]的LGI等, 其中将自省工作用于agent协同支撑方面最具代表性的是Cabri等人[24]的Mars系统. 在agent协同环境中, Mars是一个可编程的共享元组数据空间, 为实现该空间的可编程, Mars除含有普通的元组空间外, 还拥有一个元空间, 元空间存放的管理普通空间存取动作的编程指令——Reaction, Reaction可以通过agent载入进行变更. 协同实体对普通元组空间的数据访问将触发相应元空间的某些reaction, reaction的执行可能会影响协同实体的协同行为. Mars中的可编程思想为我们的工作提供了良好的借鉴. 但是, 相比Mars, 本文工作中的基于agent的多模式协同媒体适用面更广、更灵活, 对多模式协同的支撑能力更强.

5 结语

软件服务集成和协同是构建大规模网络应用的基本方法之一. 由于软件服务基于的协同模型以及采用的交互模式各不相同, 软件服务的多模式交互技术及其中间件是软件服务协同研究中的关键技术之一. 针对上述目标, 本文首先提出了一种基于 agent 的软件服务多模式协同框架, 将软件服务协同统一在 agent 协同框架中. 针对交互模式的多样性, 抽取了交互模式基本特征及其处理规程, 构建了一个统一的交互模式配置模型, 一方面为已有模式的互操作提供可能, 另一方面支持各种新模式的创建. 在交互支撑方面, 采用 agent 的概念和技术来统一和拓广各种有软件协同媒体的本地代理接口(如 RPC 和 ORB 的 Stub/Skeleton), 并利用自省的技术设计了一个可编程多模式交互协同媒体. 最后, 将上述工作有机融合, 本文介绍了一个软件服务多模式协同中间件 Artemis-M3C 环境. 实例测试及性能分析结果表明该方法的可行性和系统在支持软件服务的多模式协同方面具有一定的实用性和效率.

和传统 RPC 及 CORBA 等中间件相比, 基于 agent 的多模式交互中间件具有“多种交互协同模式共存与互操作、中间件支撑模型可配置、交互协同对象可剪裁”等特征, 为解决经典中间件模型中存在的“交互模式的单一固定性、中间件交互支撑的僵硬性”等不适合开放应用需求的问题提供了新的途径.

需要说明的一点是, 软件服务的语义研究和适配研究等并不在研究范围之内. 鉴于我们的研究基础和研究方向, 后继工作将在交互模式配置模型的形式化方面和交互模式库及其管理方面以及软件服务的可信计算方面进行. 力争使得 Artemis-M3C 环境能够真正适用于 Internet 范围内软件服务的多模式集成和协同.

参考文献

1 吕建, 陶先平, 马晓星, 等. 基于 Agent 的网构软件模型研究. 中国科学, E 辑, 2005, 35(12): 1233—1253

2 Birrel A, Nelson B. Implementing remote procedure calls. ACM Trans Comp Syst, 1984, 2(1): 39—59 [DOI]
3 Balen H, Elenko M, Jones J, et al. Distributed Object Architectures with CORBA. Cambridge: Cambridge University Press, 2000
4 Arnold K, Osullivan B, Scheifler R, et al. The Jini(TM) Specification. Redwood, London: Addison-Wesley, 1999
5 Corradi A, Zambonelli F, Leonardi L. A scalable tuple space model for structured parallel programming. In: Proceedings of Conference on Programming Models for Massively Parallel Computers. Berlin, Germany: IEEE CS Press, 1995. 25—32
6 Papadopoulos G, Arbab F. Modeling activities in information systems using the coordination language MANIFOLD. In: Proceedings of ACM symposium on applied computing. Atlanta, Georgia: ACM Press, 1998. 185—193
7 Parnas D. On the criteria to be used in decomposing systems into modules. Commun ACM, 1972, 15(12): 1053—1058
8 Kiczales G, Lamping J, Mendhekar A, et al. Aspect-oriented programming. In: Mehmet A, Satoshi M, eds. Proceedings of ECOOP. Finland: ACM Press, 1997. 220—242
9 Ward S, Robert J, Halstead H. A syntactic theory of message passing. J ACM, 1980, 27(2): 365—383 [DOI]
10 Lange D. Seven good reasons for mobile agents. Commun ACM, 1999, 42(3): 88—89 [DOI]
11 Cabri G, Leonardi L, Zambonelli F. Engineering mobile agent applications via context-dependent coordination. IEEE Trans Softw Eng, 2002, 28(11): 1039—1055 [DOI]
12 Maes P. Concepts and experiments in computational reflection. In: Proceedings of OOPSLA. Orlando: ACM Press, 1987. 147—155
13 Costa F, Duran H, Parlavantzas N. The role of reflective middleware in supporting the engineering of dynamic applications. In: Proceedings of the 1st OOPSLA Workshop on Reflection and Software Engineering: Reflection and Software Engineering. New York: ACM Press, 1999. 79—98
14 Gamma E, Helm R, Johnson R, et al. Design Patterns: Elements of Reusable Object-Oriented Software. Redwood. London: Addison-Wesley Professional, 1994
15 Erl T. Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. London: Pearson Education, 2004
16 杨芙清, 梅宏, 吕建, 等. 浅论软件技术发展. 电子学报, 2002, 30(12A): 1901—1906
17 梅宏, 陈锋, 冯耀东, 等. ABC: 基于体系结构、面向构件的软件开发方法. 软件学报, 2003, 14(4): 721—733
18 Mary S, DeLine R, Klein D, et al. Abstractions for software architecture and tools to support them. IEEE Trans Softw Eng, 1995, 21(4): 314—335 [DOI]
19 Lee T, Hendler J, Lassila O. The semantic web. Sci American, May, 2001
20 Blaze M, Feigenbaum J, Ioannidis J, et al. The role of trust management in distributed systems security. In: Proceedings of Secure Internet Programming: Security Issues for Mobile and Distributed Objects. New York: Springer, 1999. 185—210
21 Blaze M, Feigenbaum J, Lacy A. Decentralized trust management. In: Proceedings of IEEE Symposium on Security and Privacy. Oakland: IEEE Press, 1996. 164—173
22 Papadopoulos G. Special track on coordination models, languages and applications. In: Proceedings of ACM symposium on Applied computing. San Antonio: ACM Press, 1999. 147—148
23 Domel P, Lingnau A, Drobnik O. Mobile agent interaction in heterogeneous environments. In: Proceedings of the 1st International Workshop on Mobile Agents. Berlin: ACM Press, 1997. 136—148
24 Cabri G, Leonardi L, Zambonelli F. Mobile agent coordination models for Internet Applications. Comput, 2000, 33(2): 82—89 [DOI]
25 Lucchi R, Zavattaro G. WSSecSpaces: a secure data-driven coordination service for web services applications. In: Proceedings of ACM Symposium on Applied Computing. Nicosia: ACM Press, 2004. 487—491
26 Lemniotes T, Papadopoulos G. Coordinating web services using channel based communication. In: Proceedings

of the 28th Annual International Computer Software and Applications Conference. Los Alamitos: IEEE Computer Society Press, 2004. 486—491

27 Minsky N, Ungureanu V. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. ACM Trans Softw Eng Methodol, 2000, 9(3): 273—305 [DOI]

28 Stefano A, Santoro C, Pappalardo G. Enforcing agent communication laws by means of a reflective framework. In: Proceedings of ACM Symposium on Applied Computing. Nicosia: ACM Press, 2004. 462—468

29 Dimitoglou G, Moore P. Middleware for large distributed systems and organizations. In: Proceedings of the 1st International Symposium on Information and Communication Technologies. Dublin: ACM Press, 2003. 536—542

30 Mamei M, Zambonelli F. Self-maintained distributed tuples for field-based coordination in dynamic networks. In: Proceedings of ACM Symposium on Applied Computing. Nicosia: ACM Press, 2004. 479—486

31 Blair G, Coulson G, Blair L, et al. Reflection, self-awareness and self-healing in OpenORB. In: Proceedings of the 1st Workshop on Self-healing Systems. Charleston: ACM Press, 2002. 9—14