软件协同中基于中介的协同模型应用研究

许 婷,俞 春,陶先平,吕 建
(南京大学计算机软件新技术国家重点实验室,江苏南京 210093;南京大学计算机科学与技术系,江苏南京 210093)

摘 要: 在开放、动态的 Internet 平台上实现软件资源的协同是网络技术研究热点之一.传统的软件协同存在协同模式单一的不足,不能很好适应网络资源的多模式协同.本文在我们开展的多模式软件协同研究背景下,基于共享空间协同技术,针对多模式协同特点,在基于 agent 的多模式协同中间件中,设计并实现了一个基于中介的协同环境,为多模式软件协同提供了良好的中介协同支撑.

关键词: 协同; 中介; 元组; 匹配
中图分类号: TN393 **文献标识码**: A **文章编号**: 0372-2112 (2004) 12A-226-05

Design and Realization of a Space Based Software Coordination Model

XU Ting, YU Chun, TAO Xian-ping, LV Jian
(*State Key laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu* 210093, *China*;
Department of Computer Science, Nanjing University, Nanjing, Jiangsu 210093, *China*)

Abstract: How to realize software coordination in the Internet environment becomes a hotspot nowadays. Traditional Single-pattern coordination model can not adapt to the Internet environment. Multi-pattern coordination model will help a lot. This article proposes a space-based coordination architecture, in which space is a shared data store. This kind of architecture has outstanding flexibility and adaptability, which gives a good supporting to Multi-pattern coordination.

Key words: coordination; space; tuple; match

1 引言

Internet 技术的快速发展和普及使得 Internet 平台之上需要高效的资源协同,为此,以软件服务个性化和服务协同模式多元化为特征的中间件技术的研究成为一个核心问题.而此种中间件技术的重要方面就是软件的协同.“协同”一词是我们对“Coordinate”及其名词形式“Coordination”的汉译.从技术的角度来看,一种软件协同技术包括两个层次,一是其协同模型,二是该协同模型的软件实现.协同模型,作为“绑定一组分离的活动为一整体的粘合剂”[1],为主动独立的被协同实体之间的交互的表达提供一个框架.它通常涉及被协同实体的创建与撤销、实体间的通信、实体的空间分布等内容.

从系统实体之间的时间/空间耦合这个角度来看,不同的协同模型时间和空间的耦合程度不同.时间耦合是指交互双方在时间上的同步关系是同步的还是异步.空间耦合主要指交互的双方在空间上的依赖关系.传统的以 RPC 为代表的交互方式,如 RMI,COABA 都是一种空间紧耦合的模型.由于这些模型依赖于 Client/Server 结构,因而要保证 Client 端对 Server 端的单向引用.而基于共享空间的模式是一种时间松耦合、空间松耦合的协同模型.在这种模型中存在着一块公共区域:Space,协同的双方并不直接进行交互,而是通过 Space 作为媒介来进行交互.相对而言,这种模式的优点是:协同双方不需要直接知道对方的地址、访问方式等信息,只需要提供自己的需求给共享空间,让共享空间——Space 来做匹配服务的工作.而且交互的双方也不需要事先约定交互内容的格式,只要共同遵守 Space 的读写规则就可以了.双方具体交互的内容由 Space 来解析.目前,基于共享数据空间的系统有 Javaspace[6]和 Mars[2].这两个系统虽然实现了 Space 的基本思想,但是在协同实体双方的匹配和协同过程中提供的灵活性和适应性是不够的.

本文提出了一种基于中介的软件协同模型.这种模型的基础是协同中的共享数据空间,因而是完全时/空松耦合的.此外,我们结合了其他一些交互要素,期望提供更高的灵活性:我们支持多种可能的协同实体的类型;支持实体间通信多种可能的信道;在协同规则方面,采取了灵活多变的匹配/定位算法,支持请求方和服务方的个体需求.不仅能够帮助请求方定位服务方,还能够帮助其完成相应的方法调用,并且在调用过程中支持一定的容错处理.下文中将进行具体描述.

收稿日期:2004-09-02;修回日期:2004-11-18
基金项目:国家自然科学基金(No. 60233010,No. 60273034);国家 863 计划(No. 2002AA116010);国家 973 计划(No. 2002CB312002)

2　技术背景

本文的研究背景是南京大学软件与新技术国家重点实验室研究的"基于 Agent 的多模式协同中间件 ARTeMIS. M³C"系统.该中间件以 agent 技术为基础技术,支持软件服务集成和多模式协同,主要功能包括软件服务发现、软件服务集成、多模式协同配置、软件服务协同、系统安全及监控等功能.该系统的多模式协同配置中,支持以 RPC 和以中介为代表的几种协同模型.基于中介的协同模型以共享数据空间为基础,体现了协同中的更高的灵活性.

如图 2 所示,在有中介的协同模型中,中介充当了请求方和服务方之间的桥梁.请求方和服务方之间事先不需知道对方的信息(名字标识、访问地址、服务的调用方式等).他们所需的仅仅是中介的信息(名字标识,地址和调用方式).请求方将自己的需求提供给中介,服务方把自己可以提供的服务注册到中介上,这是两个可以不同步的操作.中介为每个请求匹配到相应的服务,并组织调用,最后将结果返回给请求方.在这种协同模型中,表现出如下三个特征:

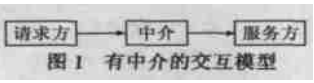

图 1　有中介的交互模型

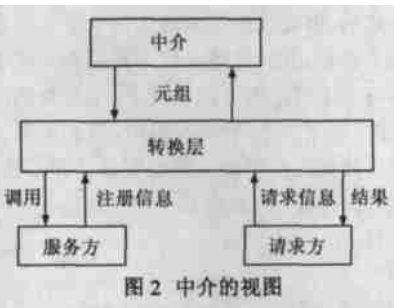

图 2　中介的视图

特征 1:由共享数据空间所带来的时间和空间上的松耦合.空间松耦合表现在协同双方在空间上的不相关性.而时间上的松耦合表现在请求方和服务方不需要任何时间上的同步,服务方的注册和请求方的请求是完全分离的操作.虽然请求方也可能会被阻塞,但是这种阻塞等待所产生的原因是中介找不到任何一个匹配的服务进行调用,而并非某个特定的服务是否被注册.

特征 2:灵活的匹配算法.如何将请求方的请求和服务方的服务进行匹配,中介提供默认的匹配方式.在默认的匹配方式中,考虑了不同匹配算法的优点,体现了通用性.但是更重要的是其所提供的可编程功能,这种可编程功能体现了针对不同交互实体个体的灵活性.请求方和服务方可以将自己特殊的匹配要求放入中介中,中介可以动态的实现请求方和服务方之间的匹配,这种动态的匹配是针对某个具体的请求方和服务方而言的,并且在某次涉及该请求方或服务方的匹配过程中被触发.

特征 3:灵活的调用方式和调用过程.在传统的以 RPC 为代表、依赖于 stub-skeleton 的模式中,请求方必须明确的知道服务方服务的调用方式,并且根据服务方的调用方式来组织参数,进行调用.而在基于中介的交互模型中,由于中介的存在,将以前的一条完整、统一的信道分割成了两个部分:中介和请求方之间的信道及中介和服务方之间的信道.中介可以适应请求方和服务方之间相异的实体模型.对于请求方来说,只需要知道中介的访问方式,将所需的服务名和参数传递给中介,而具体找到服务方和进行相应调用的工作都是由中介来完成.从而具体服务的调用方式就被中介屏蔽了.同时,服务的调用过程也被中介所屏蔽,如果调用不成功,中介可为请求方做出一些相应的处理.

3　基于中介的协同环境

在中介中,请求方的请求信息和服务方的服务信息都被抽象为元组的形式存放,中介为每一个请求元组匹配相应的服务元组.从静态的角度看,我们可以把中介看作若干元组的集合.从动态的角度看,中介频繁的进行着元组的写入、写出和匹配工作.同时,中介不仅要进行请求方和服务方信息的匹配工作,还要根据匹配到的信息进行相应的调用,最后把结果返回和请求方.我们可以把对中介的视图表示如图 2.

3.1　请求方和服务方的匹配机制

请求方和服务方信息的匹配,是中介的核心工作之一.中介将请求方和服务方的用以匹配的内容表示为一个元组.一个元组由若干分量所构成,可以具体表示为:

(方法名,{备选方法名集合},{参数类型序列},返回类型)

在元组的内容中,第一个分量代表着方法名,在计算机中应该表示为一个字符串.方法名是服务的名字.对于请求方来说,它所提供的方法名代表了它所期望的服务的语义信息,比如提供方法名为"add",表示其希望得到加法的服务,"sub"表示其希望得到减法服务.不同的是,对于服务方来说,它所提供的方法名表示它所提供的服务的具体的方法名,而不仅仅是一种语义,而是实际可以调用的方法名.

备选方法名集合:在计算机中应该表示为一个字符串的集合,仅仅是对于请求方而言的,而且对于请求方来说也是可选的.代表了请求方在服务名称上的可能的多种近似的选择.

参数类型序列:代表了服务所需要参数的相应类型的序列.对于大多数服务来说,参数应该是有序的,因此对于大多数服务来说,请求方和服务方提供的参数类型序列中参数应该是一一对应的.

返回类型:代表了服务返回的类型.请求方和服务方的返回类型应该是完全相同的.

对于服务方而言,用以匹配的元组是其可以提供的一个具体的服务.而对于请求方而言,用以匹配的元组是一个模板,代表了其所希望得到的服务.所以元组虽然具有统一的表现形式,又可具体分为请求元组和服务元组.请求元组是请求方提供信息的抽象,服务元组是服务方提供信息的抽象.

中介将综合多种匹配算法决定两个元组是否匹配,匹配算法可能有多种,但是他们的匹配程度和应用场景是不同的.下文中将描述几种不同的匹配算法,以及系统是如何采用这些匹配算法来实现匹配的灵活性.

3.1.1　元组之间的匹配算法

3.1.1.1　基本匹配算法　基本匹配算法是决定两个元组是否匹配的最一般的算法,也是最为严格的算法.在参数类型上,要求参数类型的顺序是一定的,并且不考虑模板中的备选方法名序列.

对于模板 A 和服务元组 B 来说,A 和 B 相匹配必须满足下列条件:

(1) 参数的个数相等,即参数类型序列的长度相等;

(2) 每个模板中参数类型都要与服务元组中对应位置上的参数类型相容.

(3) 请求方所提供的请求方法名和服务方的服务方法名要基本吻合.基本吻合的含义是完全相同或存在词汇上的互相包含关系和大小写兼容的关系.例如,"add"和"Add"是基本吻合的,"add"和"my_add"也是基本吻合的.

(4) A 和 B 中的返回类型完全相同.

3.1.1.2 考虑备选方法名的基本匹配算法 考虑备选方法名的基本匹配算法在基本匹配算法的基础上,考虑了请求方提供的备选名字.如果模板 A 和服务元组 B 满足上面基本匹配算法中的第一条,第二条和第四条,并且 A 中的方法名或备选方法名中的任意一项与 B 中的方法名基本吻合,则我们认为 A 和 B 是匹配的.从上述规则可以看出这种算法比基本匹配算法要宽松,考虑了请求方的多种选择,具有更高的灵活性,但是根据这种算法找到的元组不一定是匹配程度最高的.

3.1.1.3 非严格匹配算法 这种算法主要用于服务型构中参数的匹配.在有些参数顺序无关的服务中,为了能够更大范围的查找匹配的服务,可采取非严格匹配机制.该机制先将服务型构中的参数按字典序重新排列,将重排过的参数列表与服务进行比较,若匹配则原服务请求与该服务匹配.

3.1.2 元组之间的匹配机制

3.1.2.1 匿名匹配机制 在上述的算法中,我们都支持匿名匹配机制.所谓匿名匹配是指在上述的元组中,请求方可以把任意一项或多项表示为匿名分量进行匹配.中介对匿名分量不做处理,默认为匹配.这种机制考虑到匹配中可能出现的语义不确定性,主要有两种作用:(1) 请求方可以通过这种方式扩大可能匹配到的服务的范围.这样,当中介发现一个服务调用不成功的时候,可以找到更多可能替代的服务;(2) 允许请求方保留一些自己不确定的因素,把一些处理的工作都交给中介来做.

3.1.2.2 动态匹配机制 动态匹配体现了中介的可编程特性.具体来说,请求方在提供请求信息的同时,可以把自己对服务方的具体要求写成一段程序,然后把这段程序提供给中介.中介把这段程序存放在元空间中.中介在涉及这个请求的相关匹配工作时,会在元空间中检索与匹配元组相关的程序,并触发执行这段程序.对于服务方来说,也是如此.在中介中,这样一个可触发的动态匹配程序可表示为一个三元组:

(Id, T, $reaction$)

Id 代表这段动态匹配程序的书写者. T 代表一个元组,这个元组既可以是请求元组也可以是服务元组. $reaction$ 代表 T 在和别的元组匹配时所要求触发的匹配操作.由于考虑了交互双方的个体的需求,动态匹配会给中介的匹配操作带来一些灵活性,但是过分的灵活性也会导致一些问题.所以我们需要对动态匹配机制进行规范:

(1) 首先,对于特定的请求方或服务方来说,他们只能对自己写入的请求或服务元组写动态匹配程序,无权对别人写入的元组操作.也就是说任何一个有权写入(Id, T, $reaction$)的 Id,都必须是 T 的写入者.

(2) 其次,触发的程序中不能对中介中已有的其他元组进行修改.

这些规则的执行,需要中介采用一定的安全策略.一个可行的策略是:使得每个访问中介的请求者或服务者都需要持有一个 Id.同时每个元组都有一个所有者,这个所有者就是写入它的 Id.只有元组的所有者才能对其进行修改.对于(Id, T, $reaction$),也只有在此 Id 是 T 的所有者的情况下,才能被写入.在这种策略之下,可以保证元组只被其写入者修改.

3.1.3 中介匹配过程

在中介中,如何为一个请求找到相应的服务呢?首先,请求方和服务方提供的原始信息都不是上文中所提到的元组形式的,因此要把请求转化为请求元组,把服务的注册信息转化为服务元组,这些工作都是匹配的必要的准备.然后,在考虑有多个可能服务的情况下,尽量找到匹配程度最高的服务.其次,如果找不到匹配程度较高的服务,我们可以在允许的范围内找到匹配度低一些的服务.同时,还要考虑用户个体的特殊的匹配要求.根据以上想法,给出下面的匹配算法流程(图3).

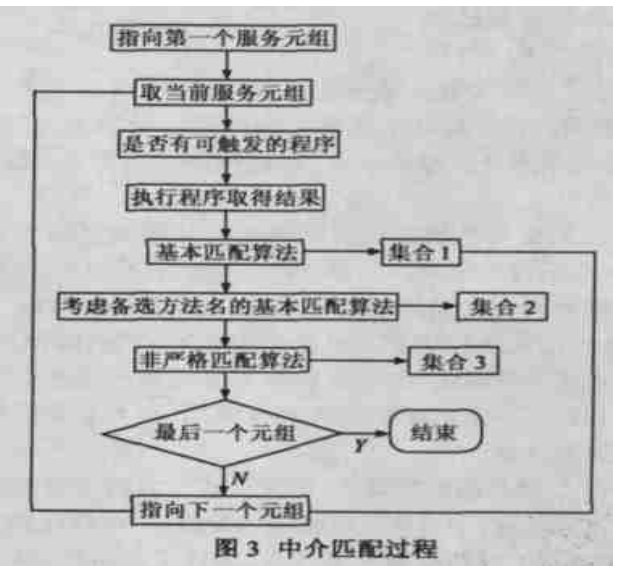

图3 中介匹配过程

每写入一个请求元组时,就代表了一次匹配过程的开始.中介通过轮询所有的服务元组,找到相应的三个集合.这三个集合代表了与请求元组匹配程度不同的服务元组.这三个集合之间是没有交集的.集合1代表了基本匹配算法认为匹配的服务元组.集合2代表了考虑备选方法名的基本匹配算法认为匹配的服务元组.集合3代表了通过打乱参数次序的非严格匹配算法认为匹配的服务元组.

3.2 中介对协同过程的适应性

3.2.1 中介对协同实体类型和信道的适应

由于中介的存在,造成了发送方和接受方之间信道的分割.在客观上为发送方和接受方在信道和访问方式上的适应性提供了可能.在传统的 RPC 方式和中发送方和接受方的信道要求是统一的.虽然 CORBA[7] 中通过 ORB 的互操作对这一

缺陷做出了改变,但是迄今为止,能够支持的也是为数不多的互操作协议.而有了中介,支持发送方和接受方之间不同的信道和交互方式就变得相对容易.中介支持以下几种变化:

(1)为请求方请求中介提供了多种可能的信道.我们用包装的方法,为中介提供多种形式的接口.例如 RMI 形式的接口,Web Service 形式的接口.特别是,以 Web Service 方式提供的接口可以和其他服务一样的发布和调用.这就提供了一种 Web Service 的可能得应用场景:客户可以到网上去查找自己所需要的服务,如果没有发现注册的合适的服务,也可以直接调用中介所提供的服务,间接的去寻找和调用自己所需要的服务.同时,我们为服务方的注册也提供了多种形式的接口.从这个角度来看,请求方或服务方与中介之间的通信是一种基于 C/S 模式的通信.作为服务器端的中介提供了多种供客户端调用的方式.

(2)支持服务方不同的调用方式:在调用的过程中,中介可以根据匹配到的服务方的调用方式动态的组织参数,进行调用.

3.2.2　中介对调用中错误的处理

中介处理调用过程中可能的失败:中介如果发现一个服务的调用发生异常,采用如下策略解决:

(1)如果中介已经为请求方匹配了多个服务,还有别的服务没有被调用过,中介可以尝试调用别的服务;

(2)如果中介暂时寻找不到可以调用的服务,可以让请求方在中介中等待一段时间,等待新的可用服务的注册.

4　系统实现及实例分析

基于以上关于中介设计的讨论,我们使用 Java 语言,并在 RMI、Web Service 等技术的支持下实现了中介的功能.下面,将具体描述系统的实现.系统实现分为以下几个部分:(1)提供元组的存储和匹配机制:系统提供了一个公共的空间来存放元组形式的信息.并且提供了对这种空间的管理方法;(2)提供元组与注册信息和请求信息之间的转换:

用户所提供的信息和真实的元组之间存在差异,系统实现了这些信息和元组之间的正反向转化工作;(3)提供请求方和服务方使用的接口:系统提供了服务方注册的接口及请求方请求中介的接口.同时,双方还可以通过一组接口提供自定义的匹配程序;(4)其他功能:考虑到中介中的元组和其他信息需要持久的存储,系统利用 java 语言中提供的持久化存储机制,把中介中的元组和其他信息同时存入文件中.同时利用 java 语言中提供的身份认证和权限机制,实现了简单的安全管理.

我们的系统作为"基于 Agent 的多模式协同中间件 ARTeMIS. M^3C"系统的一部分,运行情况良好,能够达到设计时所预想的功能.

下文将通过一个具体的例子来体现中介对协同过程的支持(图 4,图 5):*B* 和 *C* 是存在于网络中的两个服务.他们都可以提供两个整数的加法服务.不同的是,*B* 是通过 Web Service 的方式提供的,而 *C* 是一个 RMI 程序的 Server 端.*B* 和 *C* 都将详细的服务信息:服务名、参数类型序列、返回类型、服务地址、访问方式注册到了中介 *D* 中.

同时服务方 B 对请求方有相应的要求,提供了如下一段程序(用 java 语言表示如下):

```
Class B. Reaction {
  Boolean match(RequstTuple T1,ServiceTuple T2) {
      Int i = T1.getParaCount();
      Boolean b = false;
      for(int j = =0;j + + ;j < i)
if T1.getParamAt[j] < 100 b = false;
      Return b:
}
```

可以看出 *B* 要求请求方的每一个被加数都小于 100;

请求方 *A* 向中介提出一个请求:请求的内容是要进行两个整数 3 和 5 的加法操作,结果返回一个整数.中介将这个请求转换为元组,并进行相应的匹配工作.中介经过轮询匹配到了两个已注册服务 *B*,*C* 分别存放在集合 1、集合 2 中.上文中曾经提到过,集合 1、集合 2 和集合 3 的匹配程度是不同的,所以调用的次序应该是不一样的.中介首先考虑调用的应该是集合 1 中的服务 *B*.中介从服务的注册信息中获取了服务的地址和调用方式,发现服务是采用 Web service 方式访问的.中介把请求方的参数组织成 Web service 的调用方式,并进行调用.中介发现在调用的过程中抛出了异常.这时,中介就可以停止这次调用,接着从集合 2 中选取服务 *C* 进行调用.中介从服务注册信息中发现 *C* 的调用方式是 RMI 方式的.就根据 RMI 的调用方式组织参数,并进行调用.如果这次调用是成功的,就把结果返回给请求方,如果不成功,把错误信息返回给请求方.

图 4　服务在中介中的注册

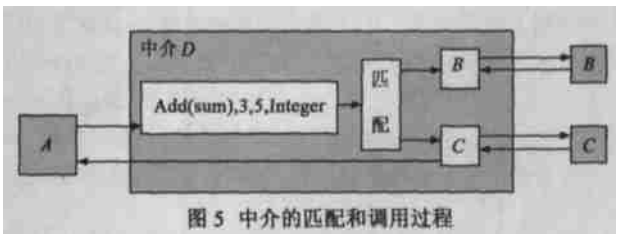

图 5　中介的匹配和调用过程

5　相关工作及比较

最早的 Space 系统是 Jini[8] 系统中的 Javaspace[6].Javaspace 的思想来源于 Linda 模型,但是它的主要用途却不是用于协同.其主要用途是为 Jini 群体中的实体提供存储区.但是由于其基于元组的存储方式和灵活的读写功能,使其已经能够充当"共享黑板"的作用:作为供/需双方的共享数据空间,双方可以借此交换数据.在 Javaspace 中,用户可以通过读/写方法访问 Space.

最近,比较有影响的 Space 系统是 Mars(Mobile Agent Reactive Spacese)[2].其主要用于移动 agent 之间的协同:读/写 Space

的实体都是移动 agent ,Space 作为移动 agent 之间协同工作的媒介. Mars 的存储机制和读写机制与 Javaspace 基本相同,其比较特殊的一点就是提出了 Reactive Spaces 的概念,也就是其所说的可编程特性. 读/写的双方都可以编制程序. 这些程序可以在读/写 Space 的过程中动态的触发. 这种可编程特性主要就是为了在协同的过程中,增加协同个体的需求,提供一定的灵活性. 目前,Mars 已经有多个在不同移动 agent 平台下运行的版本.

我们的系统和 Javaspace 和 Mars 有相似之处,在设计思想上都是用共享数据空间来实现协同. 具体来说,例如用元组作为存储和匹配的基本单位,元组的持久化,匹配算法的可编程等. 但是,我们的系统和 Javaspace ,Mars 有很大的不同:

应用的场景不同. Javaspace 主要是为 Jini 提供实体的持久化存储,并通过其读写接口做一些服务的查询工作. 而 Mars 主要是用于移动 Agent 的协同. 而我们设计的中介的主要应用场景是两个软件实体之间的协同. 这种协同主要表现在请求方和服务方之间的交互,由于应用场景不同,因而工作重点也是不同的.

在匹配算法上增加了新的内容. 在 Javaspace 和 Mars 中采用的都是简单的属性读. 这种匹配方法在我们的系统中,只相当于匿名匹配. 而我们的系统中,由于元组的特殊性而增加了多种匹配算法. 例如非严格匹配、动态匹配等.

增加了 Space 的功能. 在 Javaspace 和 Mars 中,Space 只是作为共享的数据空间,其本身是被动的,只能供使用者读写. 而在我们的系统中,中介是主动的. 其不仅帮助请求方找到相应的服务方,还进行相应的调用,并通过调用增加了处理上的灵活性.

6 结论

综上所述,本文提出了一种基于中介的软件协同模型,这种模式具有时间/空间松耦合的特性,因而与传统的 RPC 模式相比具有更大的灵活性和网络适应性. 但是,由于中介在匹配和调用上需要一定的开销,所以会影响效率. 在请求方明确知道服务方的信息、并且能够进行调用时,就不必要通过中介的方式. 理想的应用场景是综合多种协同模型,根据实际情况:如网络情况、用户需求等来决定具体的协同模式. 同时,本文主要介绍了中介的模型和设计,本文中所提到的匹配算法、安全策略等内容都将有进一步的研究.

参考文献:

[1] N Carriero ,D Gelernter. Coordination languages and their significance [J]. Communications of the ACM ,1992 ,35(2) :97 - 107.

[2] G Cabri ,L leonardi ,F Zambonelli. Mars: A programmble coodination achitecture for mobile agents [J]. IEEE Internet Computing ,2000 ,4 (4) :25 - 26.

[3] Giacomo Cabri ,Letizia Leonardi ,Franco Zambonelli. Mobile-agent coordination models for internet application [J]. IEEE Computer ,2000 ,33 (2) :82 - 89.

[4] P Ciancarini. Coordination models and languages as software integrators [J]. ACM Commputing Surveys ,1996 ,28(2) :300 - 302.

[5] D Gelernter ,N Carriero. Coordination languages and their significance [J]. Communications of the ACM ,1992 ,35(2) :96 - 107.

[6] J Waldo ,et al. A note on distributed computing ,mobile object systems [J]. Lecture Notes in Computer Science ,Springer Verlag(D) ,1997 , (1222) :49 - 64.

[7] Sun Microsystems. The JavaSpace Specification [Z/OL]. http ://chatsubo. javasoft. com ,June 1997.

[8] Object Management Group. Common Object Services Specification [Z/OL]. Volume 1 ,March 1994 ,http ://www. omg. com.

[9] Sun Microsystems. Jini Architecture Specification [Z/OL]. http ://www. sun. com/software/jini/ ,June 1997.

作者简介:

许 婷 女,1981 年生,硕士研究生,主要研究领域为移动 agent 技术,软件协同. E-mail :xuting @softlab. nju. edu. cn.

陶先平 男,1970 年生,博士,副教授,主要研究领域为移动 agent 技术,软件中间件技术.

吕 建 男,1960 年生,博士,教授,博士生导师,主要研究领域为面向对象技术,分布计算技术,构件技术.