RBRS: 一种基于角色的移动代理协作模型

李 钢, 李增智, 王 宇, 王 立
(西安交通大学系统结构与网络研究所, 陕西西安 710049)

摘 要: 在基于移动代理的应用中, 相关的协作机制扮演着一个非常重要的作用, 但当前的各种协作模型并不能很好的满足移动代理对协作的要求. 本文提出了一种基于角色的协作思想, 其核心就是将协作功能从移动代理的实现中分离出去, 然后详细介绍了利用这种思想所实现的一个协作模型: RBRS, 最后通过一个应用实例表明, 这种功能的分离不但简化了代理的设计和实现, 而且提高了整个系统可扩充性.

关键词: 移动代理; 代理协作
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2002) 05-0636-04

RBRS: A Role-Based Mobile Agent Coordination Model

LI Gang, LI Zeng-zhi, WANG Yu, WANG Li
(*Sysnet Institute, Xi' an Jiao Tong University, Xi' an, Shaanxi 710049, China*)

Abstract: Coordination plays a very important role in mobile agent-based applications. Unfortunately, current coordination models can't finely meet the coordination need of mobile agents. In this paper, role-based coordination thinking is introduced, in which the key thinking is the separation of coordination function from the implementation of mobile agent. Then a coordination model using this thinking-RBRS-is given in detail, which indicates that this kind of separation not only simplifies the design and implementation of mobile agent, but also improves the scalability of whole mobile agent system.

Key words: mobile agent; agent coordination; coordination model

1 引言

所谓移动代理(Mobile Agent, 以下简称 MA), 是指一种能够在网络上移动代表用户执行给定任务的应用程序, 它的出现是动态网络和智能代理技术发展的结果[1], 是现在业界的研究热点之一.

为了使 MA 技术得到广泛的应用, 相关的协作机制是必须要解决的一个关键问题. 所谓协作, 就是指 MA 与 MA 以及 MA 与本地服务之间的交互过程, 这是 MA 应用程序运行期间的一个基本活动. 虽然业界就传统分布式系统的协作模型已经进行了广泛的研究[2, 3], 但 MA 移动的特性和执行环境的开放性和异质性已经对相关的协作模型提出了新的目标:

® 对代理而言, 要在开放和动态的网络环境中(如 Internet) 获得其它代理和执行环境的完全知识几乎是不可能的, 因此代理的协作应该是一种松耦合关系, 即可以不采用统一的命名空间, 而且也支持异步通信;

® 每个代理执行环境应该有自己的特性和安全策略, 并可以制定符合本地要求的协作规则;

® 不管每个执行环境的特性和协作规则, 代理的协作可以根据特定的应用需求来制定;

® 可以实现复杂的交互协议以满足更为高级的应用.

针对这些目标, 本文首先介绍了当前主流的 MA 协作模型, 分析了各自的优缺点, 并提出了一个基于角色协作的思想及其相应的实现: 基于角色的交互空间(RBRS: Role Based Reactivity Space), 最终通过一个应用实例表明 RBRS 是一种能够符合上述要求的 MA 协作模型.

2 当前主流的 MA 协作模型

目前, 有四种常用的 MA 协作模型:

2.1 客户-服务器模型

大多数 Java MA 系统如 Aglets, D' Agents[4, 5] 等都是使用客户-服务器协作模型. 在该模型中, MA 必须明确的命名与之通信的实体(包括其它 MA 和本地资源), 而且协作各方之间的活动必须严格同步. 该模型具有以下缺点:

® 由于代理的物理位置是不断变化的, 为了使代理之间可以通信协作必须要引入复杂的通信中间件, 这必将显著增加系统设计的复杂性.

® 代理之间的大量交互需要稳定的网络连接.

® 在具体的应用中, MA 可能会动态创建新的代理, 这将使得对通信各方的标识变的困难. 一般来说, 这种模型比较适合 MA 与本地服务之间的协作.

2.2 面向会议的模型

面向会议的协作模型是由 Telescript[6] 提出并在 Ara[7] 系统中得到实现, 该模型通过让代理在一个抽象的会议点进行

收稿日期: 2001-08-02; 修回日期: 2001-12-03
基金项目: 国家 863 重点项目(No. 863-511-946-008)

交互从而避免了对参与各方的明确命名. 为了协作, 一个代理可以打开一个会议, 而其它代理则可以明确地或不明确地加入, 并在该会议中进行通信以同步各自的活动. 会议一般都被限制在本地以避免诸如不可预测的延迟和不可靠性等远程通信问题. 面向会议的协作模型的主要缺点是它仍然要求交互各方必须严格同步, 即所有参与各方必须同时在同样的地方, 而这将破坏代理动态、自主的本质.

2.3　基于黑板的模型

用于移动计算的 Ambit 模型[8] 和 ffMAIN MA 系统[9] 都采用了基于黑板的协作模型, 此时任何协作都发生在每个执行环境的本地共享数据空间(即黑板) 中, 黑板既是保存消息的数据库, 又是访问本地发布的数据和服务的接口. 为了通信, 代理只需要将消息发送到黑板上而不用知道接收者在那里以及何时读取消息, 这种特性显然适合移动计算, 而且与前两种模型相比, 执行环境可以监控所有发生在本地的基于黑板的协作, 因此提供了更多的安全保证. 然而, 由于代理必须就通信和数据交换中所用的消息标识符达成一致, 即必须保证统一的命名空间, 该模型依然缺乏灵活性.

2.4　元组空间模型

包括 PageSpace[10]、JavaSpace[11]、MARS[12] 在内的交互式 Internet 应用都使用了元组空间模型(又被称为 Linda 类型的协作模型[13]), 其主要设计思想就是将关联机制引入到共享数据空间, 因此可以看成是基于黑板的模型的扩展. 在该模型中, 所有的信息都被组织成元组, 为了得到相应的元组, 需要通过模式匹配机制使用关联方式获得. 由于元组的获得是通过内容而不是通过标识符, 代理之间的协作几乎不需要相互之间的知识, 因此非常适合 MA 与 MA 之间的协作, 而且该模型还同时具备基于黑板的模型的所有优点. 但是, 当前的元组空间模型还存在一定的不足, 具体表现在以下两个方面:

• 代理与代理以及代理与本地执行环境的交互都依赖于元组空间内建的面向数据的模式匹配机制, 而这是很难实现复杂的交互协议的.

• 为了访问本地执行环境中的各种服务, 光靠元组空间的面向数据的交互模型是很难完成的.

可以看出, 在当前主流的协作模型中, 或者模型的协作能力很强, 但代理需要对执行环境具有很全面的知识, 这显然增加了实现的难度(如客户-服务器模型); 或者代理不需要太多关于环境的知识, 但模型协作能力差(如后面三种协作模型), 因此需要开发一种新的协作模型以解决当前存在的问题.

3　基于角色协作的思想

3.1　基本概念

通过对 MA 的深入研究, 不难发现一个 MA 可以看成是由两大部分构成: 即完成代理基本特性的基本功能(如安全、标识、移动等) 和完成 MA 与 MA 以及 MA 与本地服务之间交互的协作功能. 我们所提出的基于角色协作的基本思想就是将 MA 的协作功能与基本功能进行分离, 此时整个协作模型包含四个基本元素: 即代理、对象、角色和环境. 其中, 对象负责描述代理的基本功能, 角色用于描述代理的协作功能, 而环境则可以看成是保存角色的容器, 它将被安装到代理运行的每个代理服务器上. 当一个 MA 被生成时, 它仅含有 MA 的基本功能, 因此按照上述概念, 它应该是一个对象, 以后每当该对象移动到一个新的代理服务器时, 它将获得本地的环境引用, 并根据应用需求通过该环境引用动态绑定相应的角色从而完成规定的协作任务, 因此在基于角色的协作概念中, 代理是由对象通过动态绑定角色来构成的. 这四个基本元素的具体定义可被抽象地描述如下:

环境: : = [环境属性, 环境方法, 角色]

角色: : = [角色属性, 角色方法]

对象: : = [属性, 方法]

代理: : = [对象, 角色]

其中, 符号“: : = ”表示左边的部分是用右边的部分定义的.

3.2　环境和角色

在构成 MA 运行环境的每个代理服务器中都含有一个相应的环境, 一个环境就是一个容器, 它含有环境属性、环境方法以及一些角色, 其中环境属性和环境方法提供了获取角色的手段. 另一方面, 每个角色都是由角色属性和角色方法构成, 它们共同描述了相应的协作任务和规则.

3.3　对象和代理

所谓对象, 在本文就是指不包含协作功能的 MA, 它所包含的属性和方法描述了 MA 的基本功能, 为了能够与其它的代理或者本地服务进行协作以完成某个任务, 对象需要动态绑定相应的角色. 而对象一旦绑定了某个角色, 也就成为了真正意义上的 MA.

需要注意的是, 在基于角色协作的思想下, 代理和代理之间的协作规则以及代理和本地环境之间的协作规则被屏蔽到具体的角色中, 至于采用这么样的协作规则将取决于具体的应用, 如可以采用客户-服务器模型等, 这样一方面保证了具体协作协议的多样性和灵活性, 同时又能保证代理的“轻量级”特性, 这是前面所述各个主流代理协作模型所不具备的.

4　RBRS 协作模型

所谓 RBRS, 即基于角色的交互空间, 是我们所开发的一个采用了基于角色协作的思想并应用于 java MA 系统的协作模型.

4.1　RBRS 接口模型

在 RBRS 中, 环境的实现参考了 JavaSpaces 规范[11] 的思想, 此时一个环境就是一个实现了 RBRSSpace 接口的 Java 对象. 环境中包含两类对象, 一类就是承担协作任务的各类角色, 另外一类被称为角色容器 RoleContainer, 是 RBRS 系统中特有的对象, 主要作用就是作为某一类角色对象的包容器, 这两类对象都实现了 Role 接口.

为了从环境中动态绑定角色(实际上是角色容器, 下一节将详细讨论这个概念), RBRSSpace 接口主要提供了如下操作:

• write: 该操作用于向环境中写入一个角色, 它有四个参数, 即执行该操作的对象的标识、要写入环境的角色、该操作所属的事务对象、租用时间.

• read: 从环境中得到一个角色容器, 该对象中含有所有

与模板对象匹配的角色,它也有四个参数:即执行操作的对象的标识、匹配模板对象、该操作所属的事务对象、等待时间.

• take:与 read 操作很相似,只是被读出的对象将从环境中被删除,该操作的参数与 read 操作完全一样.

其中,操作接口中所含的事务对象用于保证操作的原子性,租用时间表示该角色可以在环境中保留多长时间,而等待时间表示在 read 或 take 操作返回结果之间,操作调用者可以等待的时间.需要注意的是,在 read 和 take 操作中,都将用到某种形式的模式匹配机制,该机制的实现遵循如下原则:

• 对象中所有的公共变量(即 public 变量)必须不是原始类型(如整数等),如果要用到原始类型,可以采用 Java 的包装器类.

• 如果说对象 A 和对象 B 匹配,则对象 A 中所有的公共变量和对象 B 中所有的公共变量都必须完全匹配,因此这是一个递归过程.

• 对象中公共变量的值可以为空,这时它具有通配符的作用,即一个为空的值和该变量的任何值都是匹配的.

4.2 RBRS 协作模型

首先看一下 Role 接口和 RoleContainer 的基本定义:

```
public interface Role{
    public Vector execute (Vector inputparameters);
}
public final class RoleContainer implements Role{
    private Vector roles;        //用于保存与模板匹配的所有角色
    private Vector result;       //用于保存协作通信的最终结果

    ……                          将所有与模板匹配的角色都保存到 role 中
    public Vector execute (Vector inputparameters){
        Vector input= in putparameters;
        for (int i= 0; i< roles, size(); i+ + ){
            result= ((Role)(roles. elementAt[i])). execute(input);
            input= result;
        }
        return result;
    }
}
```

图 1 角色和角色容器的基本定义

任何一个实现 Role 接口的角色都含有 execute 方法,这样,当对象动态捆绑某个角色时,它只需要调用角色的该方法就能完成相应的协作任务,因为代理只需要知道完成协作任务的角色类型及其对外接口而不用关心其具体实现,因此代理的设计得到了极大的简化,同时也提高了整个系统的灵活性.此外,在 RBRS 中,还引入了角色容器的概念,它可以看做是某一类角色的容器,之所以这样做是基于这样一个思想:在同一个环境中可能(有时也应该)有多个满足同一个模板对象的角色,这些角色可能分别负责协作的各个方面(例如有的是执行本地协作策略,有的是执行应用协作策略),如果分别绑定这些角色并依此执行,显然增加了代理的设计负担,而通过使用角色容器,可以使代理在一次操作中(即调用角色容器的 execute 方法,在该方法中,每个角色执行后的结果将是下一个角色执行时的输入)完成非常复杂的协作任务,因此对环境的每次 read 或 take 操作,代理绑定的实际是角色容器对象.

下面,简单描述一下 RBRS 的协作过程(如图 2 所示):

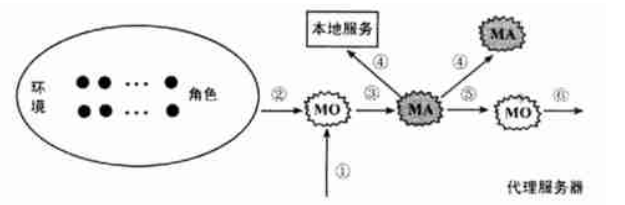

图 2 RBRS 的基本协作过程

(1)对象(图 1 中的 MO)移动到某个代理服务器;

(2)代理运行环境将本地的环境引用传递给 MO;

(3)根据具体的应用需要,MO 通过本地环境引用获得相应的角色容器,此时由于绑定了角色,对象成为了代理(图 1 中的 MA);

(4)通过角色中预先定义好的协作规则,MA 与本地服务或其它的 MA 进行交互以完成相应的任务;

(5)卸载相应的角色;

(6)对象前进到其他的代理服务器继续执行.

4.3 RBRS 安全模型

RBRS 的安全是基于这样一个前提:在允许任何代理在某个代理服务器运行之前,代理的身份验证是由移动代理系统的其它部分来完成的,在此基础上,RBRS 主要提供了基于访问控制列表的安全机制,为此,RBRS 的使用者被分为三类,RBRS 管理员、角色发布者,角色使用者.

• RBRS 管理员:每个代理服务器的管理员,他们主要负责维护相关的环境,既可以向环境写入(write 操作)特定的角色,也可以抽取(take 操作)和读取(read 操作)角色.

• 应用角色发布者:只能向特定的环境写入特定的角色,实现特定应用的协作策略.

• 应用角色使用者:即移动对象,它通过动态绑定本地环境中相应的角色来实现与其它代理或本地服务的协作,因此这类使用者只有读角色的权限.

5 主要优点

综上所述,可以看出 RBRS 协作模型具有如下优点:

(1)简化的代理设计:正如本文所反复强调的那样,RBRS 带来的最大优点就是使代理的设计变得非常简单,因为最复杂,也是最容易变化的协作功能已经从代理的实现中分离出去,保证了代理的"轻量级"特性.

(2)系统框架的可扩展性:因为协作功能与代理实现的分离,任何与代理协作有关的异质性问题都被屏蔽到协作角色中,这样当应用引入一个使用不同技术的执行环境时,不用对代理进行修改,仅需要在新执行环境中安装相应的角色即可,从而提高了系统的可扩展性.

(3)灵活的安全策略:由于角色容器的引入,可以在一个环境中同时存在多个符合某个模板的角色,这样就可以由 RBRS 管理员发布一个针对该应用的角色,用于执行特定的安全策略.从而防止未被授权的代理对本地特定资源的访问,同

时还可以使代理不需要任何异常处理.

(4)环境的高度异质性:在代理移动到的每个结点,只要可以安装环境,就可以实现代理与本地服务的交互,对代理而言,只需要知道交互的接口,具体的实现技术对它是透明的,这种特性显然非常符合大型、开放的网络环境(如 Internet).

(5)对复杂交互协议的支持:在传统 MA 协作中,为了避免使 MA 的设计和实现过于复杂,代理的协作机制通常都很简单,这当然就限制了 MA 的应用范围.而在 RBRS 中,代理的交互功能都是由安装在本地的角色来完成的,因此可以在保证代理简单实现的前提下,制定非常复杂的交互协议,并由角色来完成.

6　相关进展

目前,关于 MA 协作方面的研究,最著名的有如下几个协作模型:

(1)在 MOON 研究工程中,由 Modena 大学所开发的 MARS[12]实现了一个用于 Java 移动代理的可移植的 Linda 类型的协作结构,但它并没有从根本上克服传统 Linda 协作模型不支持复杂交互协议的缺陷.

(2)TuCSoN[14]模型采用了与 MARS 非常相似的协作模型和思想(即也支持元组空间的可编程性),而且还支持代理通过 URL 对远程元组空间的引用,但按照我们的观点,这将增加代理设计和实现的复杂性.

(3)LIME 模型[15]中的交互不仅包括移动代理,而且也包括移动用户和设备,此时参与协作的每个实体都拥有一个属于自己的元组空间,该空间是代理与外界进行交互的唯一途径.当两个实体要进行通信时,这两个实体所拥有的元组空间将进行合并,并通过合并后的元组空间进行协作.

7　结束语

在本文,提出了 MA 协作的基本目标,并提出一个可以实现该目标的协作模型: RBRS,实践证明,将代理的基本功能与协作功能进行分离,不但可以简化代理的设计和实现,而且可以提高系统的灵活性和扩充性,最终拓宽了 MA 的应用范围.当然,在本文完成之际, RBRS 中还有两个关键的问题没有解决,即: ①协作模型必须与代理通信语言进行集成从而产生高级、面向知识的代理协作; ②由于面向代理的开发方法学在分析和设计多代理程序时的重要性,因此它也应该被集成到协作模型中.

参考文献:

[1] Vu Anh Pham, Ahmed Karmouch. Mobile software agents: an overview [J]. IEEE Comm, 1998(7): 26- 37.

[2] R M Adler. Distributed coordination models for client-server computing [J]. Computer, 1995, 29(4): 14- 22.

[3] D Gelernter, N Carriero. Coordination languages and their significance [J]. Comm. ACM, 1992, 25(2): 96- 107.

[4] D B Large, M Oshima. Programming and Deploying Java Mobile Agents with Aglets [M]. Addisorr Wesley, Reading, Mass: 1998.

[5] D Kotz, et al. Agent TCL: targeting the needs of mobile computers [J]. IEEE Internet Computing, July-aug. 1997, 1(4): 58- 67.

[6] J White. Mobile agents [J]. in Software Agents, J Bradshaw, et., AAAI Press, Menlo Park, Calif, 1997: 437- 472.

[7] H Peine. Ara Agents for Remote Action [M]. in Mobile Agents: Explanations and Examples, W R Cockayne, M Zyda, eds, Manning/Prentice Hall, Greenwich, Conn: 1997.

[8] L Cardelli, D Gordon. Mobile ambients [A]. Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science, Springer-Verlag [C]. Berlin: 1998. 1378, 140- 155.

[9] P Domel, A Lingnau, O Drobnik. Mobile agent interaction in heterogeneous environment [A]. First Int'l Workshop on Mobile Agents, Lecture Notes in Computer Science, Springer-Verlag [C]. Berlin: 1997. 1219, 136- 148.

[10] P Ciancarini, et al. Coordination multiagent applications on the WWW: A reference architecture [J]. IEEE Trans. Software Eng, 1998, 24(8): 362- 375.

[11] E Freeman, S Hupfer, K Arnold. JavaSpaces Principles, Patterns, and Practice [M]. Addisorr Wesley, Reading, Mass, 1999.

[12] G Cabri, et al. MARS: A programmable coordination architecture for mobile agents [J]. IEEE Internet Computing, July-August 2000: 26- 35.

[13] S Ahuja, N Carriero, D Gelernter. Linda and friends [J]. Computer, Aug. 1986, 19(8): 26- 34.

[14] A Omicini, F Zambonelli. Coordination for internet application development [J]. Journal of Autonomous Agents and Multi-Agent Systems, Sept. 1999, 2(3): 251- 269.

[15] G P Picco, A M Murphy, G-C Roman. LIME: linda meets mobility [A]. 1999 International Conference on Software Engineering [C]. Los Angeles(CA): ACM Press, 1999.

作者简介:

李　钢　男, 1973 年出生于保定市,在读博士, 1995 年毕业于河北大学计算机科学与工程系,取得学士学位, 1998 年在西安交通大学取得计算机软件硕士学位,现为西安交大计算机系统结构与网络研究所在读博士,主要研究方向为 Web 技术在 TMN 中的应用,网络协议工程.

李增智　男,博士生导师, 1938 年出生,现任西安交通大学电信学院计算机系统结构与网络研究所所长,今年来共获得省部级奖 4 项,出版专著 6 本,在国内外期刊上发表论文总计达 200 余篇,现任中国计算机协会理事长.

王　宇　男, 1974 年出生于西安市,在读博士, 1996 年毕业于西安交大计算机科学与工程系,取得学士学位, 1999 年在该专业取得硕士学位,现为西安交大计算机系统结构与网络研究所在读博士,主要研究方向为网络管理,分布式技术.