# 摘 要

移动 Agent 是一种新型的分布式计算技术，具有良好的应用前景。作为一种能够在异构网络中自主迁移的分布式计算实体，移动 Agent 具有减轻网络负载、支持断连操作、动态适应网络等优势，在移动计算、协同计算、电子商务等领域得到了广泛的关注。然而，移动 Agent 的移动性即移动 Agent 的迁移使得移动 Agent 之间的通信产生了很多问题，现有的通信方面研究仅侧重于一对一的单播通信方式，对于移动 Agent 的组通信的研究存在不足，移动 Agent 组通信分为移动组间信息的正确到达和信息到达移动组后在组内的可靠传递。本文基于移动组的概念，对移动 Agent 的组通信展开了研究。

本文首先对移动Agent组通信问题进行了讨论。在传统的分布式组通信中，进程是静态的而在移动Agent组通信中是移动进程，是动态的，所以在移动Agent组通信中就存在四个问题：1、移动Agent的命名；2、移动Agent的定位；3、移动Agent组的消息处理；4、移动Agent组通信协调。其中，移动Agent组的消息处理包括组内信息的传递和组间信息的接收。

命名服务的提出，用以解决移动 Agent 的标识性问题。域名字解析器-DNR 负责为本域内所有的移动 Agent 提供登记服务，如 Agent 的注册、注销、查询等服务。通信服务构件 Communicator 为移动 Agent 间的通信提供服务，其具体的实现有赖于后面移动 Agent 组的组内组间通信方法的提出。

基于 ARC 的移动 Agent 组内可靠通信方法采用了基于邮箱的自适应可靠信息传递机制中的 MailBox 作为组间信息中转站和 Agent 消息接收缓存区，用以解决移动 Agent 组内 Agent 的消息处理。若 Token 标识者从 MailBox 中取出信息传递给组内的某个 Agent，此时 MailBox 用作 Agent 消息接收缓存区；若 Token 标识者从 MailBox 中取出信息将之在组内多播，此时 MailBox 用作组间信息中转站。基于 ARC 的移动 Agent 组内可靠多播通信方法实现了移动 Agent 组通信中组内信息传递的可靠性。

基于 SendBox 的组间通信方法采用了 SendBox－发送包的数据结构，发送包用于存储由于各自移动 Agent 组内的成员由于迁移时组内 Agent 未能及时收到的信息和对方移动 Agent 组由于网络通信故障而一时无法发送到对方移动 Agent 组的信息中转站的信息。基于 SendBox 的组间通信方法保证了移动 Agent 组通信中组间信息传递的可靠性。

最后，本文在 IBM 的 Aglets 平台上进行了实验，实验移动 Agent 组通信的移动 Agent 组内可靠多播通信和基于 SendBox 的组间通信方法，并对这两个方法进行了性能分析。性能分析主要对移动 Agent 组通信的通信速度和稳定性进行了分析。

**关键词：移动 Agent；移动组；组通信；可靠多播；发送包**

**Abstract**

Mobile agent is a new distributed computation technology and has a good application prospect. As an independently migrating distributed computing entity, mobile agent can alleviate network load, support disconnected computing, dynamically adapt to networking environments and so on. Therefore, mobile agent has been adopted in many application areas, such as mobile computing, cooperative computing and e-commerce. However, the mobility of mobile agent brings lots of problems in mobile agent communications. But current researches of mobile agent communications focus on one-to-one unicast protocols and lack practical solutions for mobile agent group communications. Mobile agent group communication is divided into agent information correctly arriving at each other mobile agent group and the message is reliably delivered in the group after arriving. In this paper, we intend to study mobile agent group communications based on the concept of mobile group.

This paper first discusses the mobile agent group communication problem. In traditional distributed group communication, the process is static, but in the mobile agent group communication, the process is mobile process which is dynamic. Therefore, mobile agent group brings out four problems:1、name for mobile agent;2、locating for mobile agent;3、message handling for mobile agent group;4、cooperaing during the mobile agent group. The one needed to be cared is that message handling for mobile agent group includes the message being delivered in mobile agent group and the message being received between mobile agent groups.

Naming service is used to resolve the identity of agent. Domain name resolver providers all the agents in this domain for registry service, such as logon, logout, query and so on. Communication service component provides for the communication service between mobile agents, the detail of its realization will be expounded in the method of mobile agent group communication.

Reliable communication in mobile agent group based on ARC adopts MailBox structure which is in adaptive and reliable message deliver mechanism based on MailBox to be the interim message station and message buffer, in order to resolve message handling in mobile agent group. If token holder takes the message from the mailbox and delibers it to the special agent in the group, the mailbox acts as the message buffer.If token holder takes the message from the mailbox and multicast in the group, the mailbox acts as the message station. Reliable communications in mobile agent group based on ARC realiszes reliable message deliver in mobile agent group.

The method of communication between mobile agent groups based on sendbox adopts the data structure of sendbox which is used to store the data if agent can't send message to another agent as the current migration or agent can't send message to another mobile agent group's interim message station as the network failure. The method of communication between mobile agent groups based on sandbox ensures the reliable communication between mobile agent groups.

At the end of this paper, we make the experiment on IBM aglet mobile agent platform, evaluating the performance analyse on the method of Reliable communication in mobile agent group based on ARC and The method of communication between mobile

agent groups based on sandbox. The performance analyse focuses on the communication speed and reliability.
**KEYWORDS**: mobile agent; mobile group; group communication; reliable multicast; sendbox

Zhou Chenye (Computer Application Technology)

Directed by Shi Xiaohong

## 论文独创性声明

本论文是我个人在导师指导下进行的研究工作及取得的研究成果。论文中除了特别加以标注和致谢的地方外，不包含其他人或其他机构已经发表或撰写过的研究成果。其他同志对本研究的启发和所做的贡献均已在论文中作了明确的声明并表示了谢意。

作者签名：周晨叶 日期：2007.6.18

## 论文使用授权声明

本人同意上海海事大学有关保留、使用学位论文的规定，即：学校有权保留送交论文复印件，允许论文被查阅和借阅；学校可以上网公布论文的全部或部分内容，也可以采用影印、缩印或者其他复印手段保留论文。保密的论文在解密后遵守此规定。

作者签名：周晨叶 导师签名： 日期：2007.6.18

# 第一章 绪论

最近二十年来，计算机的软硬件技术与通信技术得到迅猛的发展，应用领域日益广泛化和复杂化。特别是国际互联网在社会各个领域都得到大量的使用，Internet 已成为我们生活不可缺少的一个重要组成部分，这导致了计算机应用由集中式、单机式系统走向分布式系统。由于分布式人工智能技术与移动计算技术的发展，为了克服网络低带宽、高延时对网络应用普及的限制，移动 Agent 技术得到发展，并成为新的研究热点。

## 1.1 研究背景

移动 Agent本质上是代表用户在网上寻找合作伙伴，进行交互并最终完成用户指派的任务的一个对象，因此协作性是移动Agent的本质属性。同时由于单个Agent的能力有限，而不同用户往往具有相同的兴趣，所以可以利用Agent之间的通信(Communication)与交流，实现Agent之间的协作(Cooperation)以提高服务质量[1]。通信是协作的基础，就像我们人类社会一样，同样，Agent也可以通过彼此之间的协作来解决单个Agent所不能解决的问题，而群体的协作必然要求个体Agent之间可以自由地通信与交流。同时，各个Agent之间也是存在差异的，这种差异导致了Agent对同样的事务可能会有不同的解释，所以Agent之间也需要通过互相通信来解决这一问题。此外，由于Agent具有自主性(Autonomy)，导致Agent忽视了其它Agent的内部结构，所以在遇到与之有关的问题时需要与这个Agent进行通信。

目前在移动Agent系统中，对Agent协作性的支持主要集中于对通信机制的研究[2]代表性的研究工作有两类：一类是基于知识交换的KQML(Knowledge Query and Manipulation Language)等Agent通信语言(Agent Communication Language, ACL)研究工作，另一类是基于消息传递的Aglets等系统[3]。基于消息传递进行通信，其目的是在为高效地实现更复杂的通信协议提供基础的同时，允许需求比较简单的对象以较小的开销进行通信。Agent Tcl[4]系统中就提供了一种消息传递机制，利用agent_send和agent_ receive原语，Agent之间可以进行简单的消息发送和接收[5, 6]。 Aglet系统则通过称为“对象发送”的方法支持类似于消息传递的通信机制[7]。我们认为，高层的Agent通信语言必须建立在低层的消息传递机制的基础上，如果能够实现一种高效的消息传递机制，那么再在其上层实现ACL就可以做到水到渠成。而且，目前移动Agent系统尚未达到实用化的阶段，我们也应把主要精力投入到底层基础设施的构建之上。所以，本论文的工作属于第二种类型，即提出一种底层的移动Agent系统消息传递机制。

移动Agent领域主要从“如何支持移动”的角度研究通信，解决Agent移动产生的特殊问题。目前，有关移动Agent通信的研究主要集中在以下三个方面：(1)如何保证通信的可靠性；(2)如何提高系统的通信效率；(3)如何更好地支持MA协作。在现有的移动Agent系统中，Agent间的消息传递机制按照实现方法一般可分为如下几种类型：广播法，链状跟踪法和基站法和集中注册法。这些方法各有优点，但通过对这些方法的分析我们看出它们在快速寻址和可靠性消息传递方面都

存在缺陷，而且未给出通信失效的解决方案。

本文为了解决现有移动Agent计算模式下通信机制的不足，将移动Agent名字解析机制、移动Agent通信服务与消息传递机制相结合，提出可靠的移动Agent组通信模型，其中组通信分为组间通信和组内通信。该模型为了实现移动Agent在移动组内的可靠通信，引入采用MailBox结构的组内多播的方法，实现移动Agent组内通信的可靠性。在此基础上，对移动Agent组间可靠通信问题进行了研究，为了解决该问题，对移动Agent的状态进行划分，并引入了基于SendBox的Agent通信协作管理，解决移动组内组成员在迁移过程中的信息接收问题，为移动Agent系统的组间信息的可靠传递提供了一种通信方法。结合这次研究的组内组间通信的方法从而实现一种真正意义上的可靠的分布式移动Agent组通信机制。

## 1.2 移动 Agent 技术

1、 移动 Agent 的概念和特征

20 世纪 90年代初，General Magic公司在推出商业系统Telescript[8]，时提出了移动Agent的概念。简单地说，移动Agent是一独立计算机程序，它可自主地在异构的网络上，按照一定的规程移动，寻找合适的计算资源、信息资源或软件资源，利用与这些资源处于同一主机或网络的优势，就近处理或使用这些资源，代表用户完成特定的任务[9]。

移动Agent可以看成是软件Agent技术与分布式计算技术相结合的产物，它与传统网络计算模式有着本质上的区别。移动Agent不同于远程过程调用(RPC)[10]，这是因为移动Agent能够不断地从网络中的一个节点移动到另一个节点，而且这种移动是可以根据自身需要进行选择的。移动Agent也不同于一般的进程迁移[17]，因为一般来说进程迁移系统不允许进程自己选择什么时候迁移以及迁移到哪里，而移动Agent却可以在任意时刻进行移动，并且可以移动到它想去的任何地方。移动Agent更不同于Java语言中的Applet[11]，因为Applet只能从服务器向客户机做单方向的移动，而移动Agent却可以在客户机和服务器之间进行双向移动。

移动 Agent迁移的内容既包括其代码也包括其运行状态。运行状态可分为执行状态和数据状态:执行状态主要指移动Agent当前运行时状态，如程序计数器、运行栈内容等;数据状态主要指与移动Agent运行有关的数据找的内容。按所迁移的运行状态的内容，移动Agent的迁移[12]可以分为“强迁移”和“弱迁移”[13]。强迁移同时迁移移动Agent的执行状态和数据状态，但这种迁移的实现较为复杂;弱迁移只迁移移动Agent的数据状态，其速度较强迁移快，但不能保存移动Agent的完整运行状态。

移动Agent的工作过程可描述为:移动Agent被本地主机转移到远程主机上。远程主机将会为移动Agent提供一个合适的运行环境，移动Agent在此环境中完成收集信息、修改自身状态等操作后，移动到下一个远程主机上。移动Agent重复的执行工作的过程，直到回到本地主机。

显然移动Agent应用是一种分布式应用[14]。在概念上，一个基于移动Agent的应用由一组移动Agent构成(正如一个面向对象的应用由一组对象构成)，每一个Agent根据自身的目标和环境的状况移动到拥有计算所需资源的节点上进行计算。计算的过程中，可能需要与其它Agent(可以是同一应用中的其它Agent，也可以是运行系统提供服务的公共设施Agent，甚至可能是其它应用的Agent)进行

通信协作。这一步的计算完成以后，该移动Agent自主地决定下一步的动作，直至其任务完成并自动消亡。

与其它分布计算模式（如OMA/CORBA)相比，这种基于移动Agent的分布式计算模式有这样一些特征[15]。

(1) 从应用的角度看，真正实现了“网络就是计算机”的理想。不仅应用所需资源分布在网络中，整个应用逻辑都可以在网络上运行。

(2) 从系统的角度看，分布资源更充分的共享成为可能，但管理更为困难。从服务提供和服务使用的角度看，服务是客户化、可定制的，其使用不再限于既定的方式。

(3) 从通信协作的角度看，通信的主体是自主的Agent，可以实现对等的通信模式。

## 2、 移动 Agent 关键技术

移动Agent的实现涉及到多方面技术，其中主要有以下几项关键技术[16]。

1. 移动Agent系统的编程语言:可以是编译型的，也可以是解释型的，出于对平台无关性的考虑，一般都采用解释型的语言，由于Java语言具有良好的性能并得到广泛接受，所以，大多数移动Agent系统采用Java语言[17]。

2. 迁移机制[18]:根据移动Agent携带的内容不同，可以将移动Agent的迁移区分为强迁移和弱迁移，前者在Agent迁移之前系统捕获整个Agent的状态(数据状态和运行堆栈等执行状态)，将它们连同其代码一同传送至下一节点，到达目的地后其状态(数据和运行堆栈)被完全恢复，程序将在原来中断的地方继续往下执行。强迁移系统的Agent传输过程是完全透明的，提高了程序的模块化程度，容易理解和排除错误，因而提高了可靠性。但是该方法也提高了系统实现的复杂性，增大了通信流量。弱迁移系统如IBM的Aglet及其它一些基于Java的移动Agent系统则不允许方法执行状态的迁移(如本地变量、程序计数器等)，因此在这类系统中允许开发者将代码与某种与迁移有关的事件绑定在一起。弱迁移系统实现起来比较简单，使程序员在解决同步问题时有更大的灵活性，但是也带来了一定的问题，编程人员必须熟悉Agent的整个传送过程，并要编写内部状态捕捉及不同的入口代码，加大了编码量。

3. 通信机制[19]:在移动Agent系统中大体可分三种情况即:移动Agent与其执行环境、移动Agent之间和移动Agent与一组移动Agent通信。在移动Agent系统中可采用的通信手段很多，有RPC, RMI, Message、数据共享方式(仅在本地有效)等。

4. 容错机制[20]:为了保证移动Agent在异构环境中的正常运行，考虑到服务器异常、网络故障、目标主机不可达等各类异常情况的出现，必须提供容错手段，确保Agent完成任务。大多数的系统都用提供Agent非易失存储器备份的方法来确保其丢失后能恢复运行，比如Aglet系统中提供Agent快照(snapshot)功能，它是Agent内部状态的一个完整记录，用于出现故障时Agent的恢复。

5. 安全性:安全性是移动Agent系统中重要的问题，它是任何移动Agent系统都必须解决的。安全机制既要保证主机不受恶意Agent的攻击，又要保证合法Agent不受宿主机器的非法侵害。

本文着重于移动Agent组通信的研究。

## 3、 移动 Agent 技术的优点

移动 Agent 技术给分布式系统的设计、实现和维护都带来了新的活力。它的

新技术特性为分布式计算带来了新的计算模式和思路，具体有以下优点[21]。

(1)移动性能:移动 Agent 可以在异构网络和分布式计算机环境中自主、自动地迁移,携带信息或寻找适当的信息资源,进行就地的信息处理,代理用户完成信息传递、网页查询、数据和知识发现、信息变换等多种任务。

(2)异构和异步性能:移动 Agent 可以支持异构计算机软件、硬件环境,能进行异步通讯和计算。

(3)降低网络通讯费用:传送大量的原始信息不但费时还容易阻塞网络,如果将 Agent 移动到信息存储的地方,进行局部搜索和选择后,将选中的信息通过网络传送给用户,会大大减少远程计算机网络的连接费用。

(4)分布和并行性:移动 Agent 提供了一个独特的分布计算体系结构,为完成某项任务,用户可以创建多个 Agent,将它们同时在相同或不同的节点上运行,可将单一节点的负荷分散到网络的多个节点上,使小系统具有处理大规模、复杂问题的能力。

(5)智能化路由:移动 Agent 具有根据目标、网络通讯能力和服务器负载等因素,动态地规划下一步操作的能力.智能化路由能很好地优化网络和计算资源,实现负载均衡,提高问题的求解速度,避免对资源的盲目访问。

(6)具有自然异构性：网络计算平台往往是异构的。由于移动Agent通常独立于计算机和传输层，而仅仅依赖其运行环境，所以移动Agent提供了无缝系统集成的最优条件。

(7)健壮性和容错性：移动Agent具有对非预期状态和事件的应变能力，这使我们更容易创建健壮性和容错性好的分布式系统。Agent在移动过程中，可能存在网络发生故障或者关机等情况，从而使移动Agent遭到破坏。为防止因为移动Agent运行失败而造成数据丢失，移动Agent可以创建相同任务的多个备份在网络中独立运行，或者在特定的服务器端保留移动Agent的原始备份。而且，移动Agent可以采用分布式容错机制，将容错功能分配到网络中各个站点进行，从而提高整个系统的容错能力。

(8)提供平台无关性：移动Agent程序是跨平台运行的，移动Agent应用程序不存在程序的移植问题，便于应用的快速开发。

总之，在网络一体化的时代，移动Agent技术较之于传统的分布式技术有着明显的优势。移动Agent可在异构的软、硬件网络环境中移动。移动Agent计算模式集中了其他传统的分布式技术的优点并结合分布式人工智能技术提供了一个普遍的、开放的、综合的、简便的分布式应用开发框架，较之于传统的网络编程方式更适合于网络应用系统的开发。

## 4、 移动 Agent 技术的应用

目前，移动Agent技术已被大量的网络应用采用，同时许多相关的、深入的研究也在进行中，下面介绍当前移动Agent技术的主要应用[22,23]。

(1)移动计算：移动用户(如使用笔记本电脑或WAP上网的用户)的物理位置有可能在网络计算过程中不停地改变，因此需要不断地维护和重新配置当前的数据连接。而一旦这种连通性不能及时地维持，就需要额外地处理脱线阶段的交互数据。在这类应用中，采用移动Agent技术非常有利。用户可以在其发出的移动Agent中事先封装长期交互过程所需的全部用户端数据，由它携带用户请求移至服务器端与之进行所需的交互和计算，这期间客户端与服务器端的连接可以中断，而移动Agent仍可正常工作。Agent完成任务后，一旦连接恢复，即可携带交互结果返回初始用户处。另外，Agent程序通常有能力对交互结果进行预处理，所以对于传输和处理速度较慢的网络环境，移动Agent系统更为有用。

(2)分布式信息检索：分布式信息检索应用从散布在网络各处的信息源中搜索符合某个给定准则的信息。被访问的信息源可能是事先指定的，也可能是在访问的过程中动态决定的。因为本地交互的速度要比通过网络交互快得多，完成搜索过程的移动Agent可以通过移动到被搜索的信息库的站点来提高搜索效率。

(3)电子商务：移动Agent技术在解决电子商务问题上有着良好的前景。不同的移动Agent有不同的目标，并为实现各自的目标采取不同的策略。移动Agent体现了各自创建者的意愿，并代表他们运作和协作。除了简单的信息搜索外，移动Agent还可以完成许多商业交易，例如代表用户与电子商店进行价格协商、合同签署以及电子商品传输等等。一个典型的例子就是用户可以派出一个Agent按照预订的路线访问不同的电子商店服务器，最后携带用户所需商品的最佳报价返回。

(4)并行处理：移动Agent的一个潜在应用是管理并行处理任务，因为移动Agent能够在网络环境中生成多个副本。如果计算因为需要很强的处理能力而必须分布到多个处理器时，支持移动Agent的主机构成的并行处理设施可以提供强大的支持。

(5)个人助理：移动Agent能迁移到远程主机上运行，这使得它可以代表用户在网络中完成一些任务，从而称为人们的助手。移动Agent可以在远程主机上独立运行，这期间网络链接可以中断，发送者甚至可以关掉自己的计算机。例如，为了制定一个议会日程，用户只需发送一个移动Agent和代表其他与会者的移动Agent交互，由这些Agent制定时间表。

## 1.3 研究意义和目标

研究的意义：

虽然移动Agent具有强大的技术优势和巨大的应用前景，但至今还没有成为计算机软件业的主流技术，这是由很多原因造成的，其中，移动Agent在为分布式计算提供了一个新范例的同时，也暴露出了通信方面的不足。所以在当前，移动Agent通信成为其研究课题之一。

每个移动 Agent 总是运行在一定的主机平台下，每台主机可以运行一个或多个这样的平台。平台要接收来自不同地点的 Agent，如何使得 Agent 之间进行可靠高效的通信则是移动 Agent 通信研究的重点。

当Agent互相通信时，我们可以采用多播的方法来保证移动Agent组内多播通信的原子性，但对于移动Agent组通信中组内组间移动Agent在迁移时，对于消息追踪和超时问题则同样需要提供一个有效的处理方法，因此，我们必须设计一个可靠的移动Agent组通信的方法。

虽然移动 Agent 技术还存在着许多问题，但是它显示了相比传统的分布式计算技术在提供网络服务方面质的优势。一旦通信问题得到了较好的解决，它将成为基于 Internet 和 Intranet 的分布式计算技术的主流。因此，如何从各个方面对移动 Agent 这一全新的分布式计算模式进行研究，使其走向实用化、商业化，将对计算机科学的发展产生重要的影响。此外，许多研究者认为 Agent 将是面向对象技术之后软件工程领域的又一突破。

研究的目标：

对移动Agent进行研究的最终目标就是要实现移动Agent系统及其应用的实用化和商业化，这是需要通过研究和解决移动Agent技术所遇到的问题(如通信问题)来实现的。只有解决或者部分解决了这些问题，才能使移动Agent技术最终进入商业领域。而针对通信问题，目前还没有一种完善的解决方案。虽然也有人提出了一些方法，例如基于邮箱的方法等等，但这些方法都只是解决了问题的某一方面，需要进一步研究。目前，对移动Agent平台的保护已经做得比较好。在本文中，我们主要讨论移动Agent自身的通信问题。

## 1.4 本文的组织

第一章 绪论

本章首先介绍了本文的研究背景，然后介绍了移动 Agent 技术的优点及其应用以及本文的研究意义和目标。

第二章 移动 Agent 组通信问题研究

本章综述了移动 Agent 通信相关的研究工作。首先，阐述了移动 Agent 通信的特点，然后将传统分布式系统的组通信与移动 Agent 组通信进行对比，给出移动 Agent 组通信存在的问题，进而提出本文的最新研究成果，迁移过程中的组间可靠多播通信。

第三章 移动 Agent 组内可靠通信

本章首先给出了移动 Agent 系统的命名和通信服务层系统架构，接着给出了移动 Agent 全局寻址方式下的 Agent 通信服务和自适应可靠信息传递机制，然后指出其中的不足，进而提出采用 MailBox 存储结构的组内可靠多播通信。

第四章 SendBox 组间通信

首先是通信方法概述，用于阐述设计思想，其次是系统通信示意图，用于描述通信的过程，接着是移动 Agent 状态划分，Agent 信息显示结构，Agent 通信协作管理，最后是通信算法和有效证明。

第五章 通信实验

本章在 IBM 的 Aglets 平台上简单实现移动 Agent 迁移过程中的组间可靠多播通信，并对其性能进行分析。

第六章 总结和展望

对全文进行总结并就本文进一步的研究工作进行了讨论。

# 第二章 移动 Agent 组通信问题

移动 Agent 最重要的特点就是可移动性，在通信的过程中，发送消息的 Agent 和接收消息的 Agent 都存在位置改变的情况，这就对实现高效可靠的消息传输机制提出了一些要求，主要包括以下几个方面[24-30]：

位置透明性：由于移动 Agent 会自治地完成在主机之间的迁移，所以移动 Agent 之间的通信是一种位置透明的通信，也就是说移动 Agent 在向另一个移动 Agent 发送消息的时候，不需要知道接收消息的移动 Agent 的物理位置。因而，移动 Agent 通信协议应当能够跟踪并保存各个移动 Agent 的具体位置。

可靠性：无论接收消息的 Agent 的迁移频率有多快，消息必须保证能够到达这个 Agent。但在移动 Agent 的通信环境中，由于 Agent 的自主移动特性，常会导致消息被发送到某一网络节点但接收者已经离开而无法收到该消息。这种因为通信主体物理位置发生变化而造成的通信不正常现象，称为移动通信失效。值得注意的是，在 Agent 移动非常快的情况下，消息可能追随目标 Agent 在网络各节点间游荡却永远无法送达而产生消息追逐现象。通信失效和消息追逐都与网络和节点故障无关，是纯粹由 Agent 的移动造成的，它们使得协作中的 Agent 不能及时得到协同信息，从而导致协作的失败甚至造成整个系统的崩溃，是移动 Agent 系统的致命缺陷。

高效性：消息传输协议的成本可以用消息数量、消息大小和消息传输的距离来描述。一个高效的协议应该尽可能的降低消息传输的成本。具体而言，一个协议应该高效的支持两种消息操作：(1) Migration(迁移操作)，将 Agent 移动到一个新的网络节点；(2) Delivery(发送操作)，定位特定的 Agent 并将消息发送给它。实际上这两种操作是互相矛盾的，一个高效的协议应该在支持这两种操作上进行权衡和折衷。

异步性：异步包括两个方面，即异步迁移和异步执行。尽管 Agent 的消息传输和 Agent 迁移要求可靠性，但是并不意味着对 Agent 的移动性做出过于严格的同步要求。因为断连操作是移动 Agent 计算的重要优势，所以 Agent 的断连执行能力不应该由于 Agent 的跟踪定位需求而受到太多约束。因而，对于协议而言，要求能够保证迁移和执行计算的异步性。

自适应性：应用的需求是各种各样的，因而对于以上的需求应该有所侧重。例如，有的应用对异步性要求较高，这就要求 Agent 的定位寻址方式不能是集中式的；有的应用的可靠性要求较高，这就应该多考虑一些同步方面的要求。虽然可以针对特定的应用来设计消息通信协议，但是设计一种适应于通用 Agent 平台的自适应协议还是最佳的选择。

安全性：为保护通信安全，要对所有的信息提供通信认证，对任何可能的安全危害进行检测，比如，“加密信道－权限控制”的 Agent 系统通信安全性实现方法，提供了多层次检查机制，这种方法用基于 RSA 和 Rabin 算法的加密信道来提供底层的签名和加密服务，而在高层提供权限控制机制。需要说明的是，安全性不属于本文工作的研究范畴。

## 2.1 分布式系统的组通信与移动 Agent 组通信比较

在容错分布式系统构造中，组通信是非常重要的方法。分布式系统的组通信最早产生于上世纪 80 年代末至 90 年代初期 Birman、Joseph 所提的虚拟同步(Virtual Synchrony )概念和 Isis 组通信系统的开发[31]。自此以后，以进程为研究对象的组通信系统成为分布式通信研究的重要分支，人们提出了一系列的组通信系统与协议，如 Amoeba[32]、Consul[33]、Newtop[34]、Transis[35]、Horus[36]、Phoenix[37] 、Totem[38]等。早期的组通信研究侧重于局域网络环境，随着 Internet 的发展，面向广域网络或多个局域网络的组通信系统成为组通信研究的重点，如基于 Totem 的扩展协议[39], Inter-group 协议[40]等。在这些组通信研究工作中，组内各个进程在其整个生命周期中位置不发生改变，本文将此类组通信称为传统分布式系统的组通信。

组通信服务使分布于不同网络节点的应用进程能够作为一个组实体执行计算任务，并使多播消息能够按照一定的应用语义约束传递到组内的所有成员。通过组实体的抽象，组通信可以简化容错分布式系统的构造方法，使得分布式应用可以将系统看作一组协作进程并能够与组通信服务交互。组通信服务最核心的两个方面是组成员管理(Group Membership Management)和可靠多播(Reliable Multicast)。组成员管理用于记录参与当前组的各个进程，检测组内进程的故障或处理组内进程退出组的请求，为高层的应用维护一个组成员列表，也称为组视图。可靠多播则使组内进程可以发送消息给组内的其它进程，并保证通信的可靠性和消息传递的次序性，有时还需要保证消息传递的最高时间延迟，比如实时分布式系统的组通信[45]。由于组通信服务使分布式应用从故障检测与处理的工作中解脱出来，所以组通信对于建立容错分布式系统至关重要，它可以用于分布式协商、分布式复制、大规模数据传送等诸多应用场景。本质上，组通信服务是一种针对容错分布式系统的中间件服务，如图 2-1。

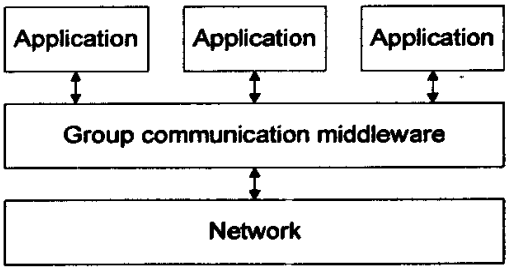


图 2-1 组通信的层次

移动 Agent 的组通信与移动环境中移动主机的组通信问题有些类似，因为二者都需要解决组成员的移动性问题。图 2-2 是一个典型的移动蜂窝网络，它由一组称为蜂窝(Cell)的几何空间构成。

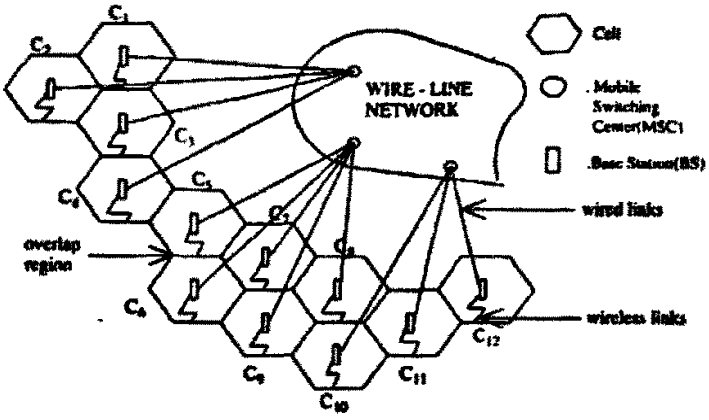


图 2-2 移动蜂窝网络模型

在每个蜂窝中都有一个服务基站 BS(Base Station)，这些 BS 与固定网络中的移动交换中心 MSC(Mobile Switching Center)连接。用户的移动主机 MH(Mobile Host)在各个单元中漫游，通过 BS 与 MSC 提供的服务实现与整个网络的连接。这样，移动环境中的组通信通常使用间接通信方式实现。间接通信方式的特点包括两个方面：(1)对于无线网络和固定网络采取不同通信路径；(2)移动主机和基站采取不同的通信策略。当一个移动主机发送多播消息的时候，会先发送消息给移动主机所在蜂窝的基站，然后基站会将此消息多播到目标多播组中的各个移动主机。文[41]通过实验分析表明，在移动网络环境中，间接通信方式在性能和灵活性方面优于直接通信方式。

然而在传统的分布式系统组通信中，进程本质上是静态的。为了保证基于移动 Agent 的应用程序像传统的分布式应用程序一样稳定，我们提出了移动组的概念，移动组是对传统概念上的进程组的扩展，这样能直接支持作为组成员的移动代理进程。

我们假设分布式系统是由移动的和静态的进程、站点和通信通道组成。一个站点表示在分布式环境中的一个逻辑位置，在这里进程可以执行。一个进程的移动就是从一个场所移动到另外一个场所。在不同场所的进程通过通信通道交换消息来进行通信。

设$L=\{l_1, l_2, \cdots, l_n\}$是一个可能存在的所有场所的集合。设P是所有可能存在的进程的集合。用$g=\{p_1, p_2, \cdots, p_3\}$，g属于P，表示一个移动组。在移动组中定义了四种操作：

- join(g, p)：由进程p调用，当p想加入组g
- leave(g, p)：由进程p调用，当p想离开组g
- move(g, p, l)：当进程p想从组g移动到组l时调用
- send(g, p, m)：当进程p想对g阻中的成员进行多播时调用

我们要知道一个消息通过通信通道被场所收到和一个消息被发送到进程p是不同的。一个消息发送给一个进程可以被延迟，为了系统能满足它的同步需求。

一个进程 p 也应该有一个 views。在移动组中，一个 view 就是 $v(g)=\{(p_1, l_1), (p_2, l_2), \cdots, (p_n, l_n)\}$，对于任意的 i，$p_i \in g$，$l_i \in L$，它是一个组中的进程和它们所在场所的一个映射。一个 view 表示了在组中存在的相互操作的组成员，并

指明了它们的位置（在 view 中（p, l）表示进程 p 在场所 l）。这个集合会动态的改变，由于进程被的挂起、离开、加入或者从一个场所移动到另外一个场所。每次一旦组的 view 有所改变的话，一个新的 view 会在进程成员中被装载。每一次进程装载的新的 view 都有一个编号，这个编号表示已经装载的 view 的数量。

图 2-3 表示一个组开始有三个进程成员 $p_1$, $p_2$和 $p_3$。这些进程分别装有 $v^1{}_1(g)$, $v^1{}_2(g)$, $v^1{}_3(g)$。这些 views 是相同的，表示进程知道的其它可操作的进程且每个每个进程都知道其它进程的当前所在场所（$v^1{}_1(g)=v^1{}_2(g)=v^1{}_3(g)=\{(p_1, l_1), (p_2, l_2), (p_3, l_3)\}$。后面进程 p3 将移动到 l4(在图 1b 中用虚线表示它的移动)。现在每个进程都装载了一个新的 view，反映了 $p_3$ 在 $l_4$ 这个场所（views $v^2{}_1=v^2{}_2=v^2{}_3=\{(p_1, l_1), (p_2, l_2), (p_3, l_4)\}$。后来，p1 所在的场所 l1 崩溃（如图 1c）。一个新的 view 又被装载到了 $p_1$, $p_2$，反映了以上情况。由于失败，p1 被从组中删除。（views $v^3{}_12(g)=v^3{}_3(g)=\{(p_2, l_2), (p_3, l_4)\}$）。

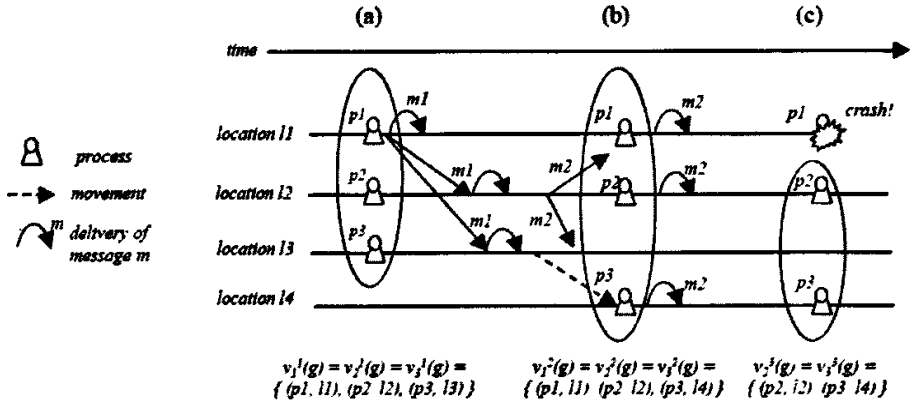


图 2-3 一个移动组的实例

我们认为通信通道是可靠的，例如消息按顺序（FIFO）发送是可以信任的。我们假设消息的发送和处理的时间是不能被精确地计算出来(比如在一个异步的系统中)，并且进程的失败只能由于系统的崩溃引起，比如停止一个功能而没有后续行为。我们也假设组 view 中的大多数进程是不会崩溃的。

通过属于某个移动组的移动进程有能力在分布式系统中改变它的位置。尽管移动组的成员是移动的，移动组也能保证消息的发送成功和一种虚拟同步。

## 2.2 移动 Agent 组通信存在的问题

移动Agent系统主要从“如何支持移动”的角度来设计通信机制[12]。在传统的分布式计算环境中，计算实体一旦被创建，他们的位置便固定下来，在整个生命期内不会改变，发送方只需得到接收方的当前位置即可始终与之通信。在基于移动Agent的计算环境中，由于移动Agent的位置可能经常变动，因此Agent之间进行通信需要考虑四个问题:

### 2.2.1 移动 Agent 的命名

用于解决分布式环境下移动Agent标识的一致性。简单的解决办法是提供基

于主机名和端口号网的名字解析机制对对象进行命名，通过域名系统(Domain Name System, 简称DNS)来完成名字解析。采用这种命名机制对于静态对象的命名和名字解析非常有效，(例如Ajanta, Aglets, Tacolna 等移动Agent系统)，但对于移动对象(如 移动Agent)却存在很大的缺陷，典型的是，当一个Agent移动时.它的名字需要同步的改变以反应其最新的位置,这就使应用跟踪Agent的移动过程变得非常复杂。因此应该使用全局的、与位置无关的命名方法对Agent进行统一命名，这需要系统提供名字解析机制[43]实现移动Agent的按名跟踪和通信。

### 2.2.2 移动 Agent 的定位

即当 Agent 位置改变时，如何让其它 Agent 知道这一变化，实现 Agent 按名寻址(即透明寻址)。有两种极端的解决方法，一种是主动通知，即当移动 Agent 位置改变时通知所有其它 Agent;另一种是通信前查询，即每个 Agent 在通信前，都需要由自己去查询接收方的当前位置。显然，第一种方法将严重占用网络带宽，而第二种方法将增加 Agent 的负担，降低计算的灵活性，因此 Agent 的位置追踪应该由移动 Agent 系统中某一模型完成，即系统应为 Agent 提供位置透明的通信机制。

### 2.2.3 移动组的消息处理

即当组内 Agent 迁移时，如何处理移动 Agent 与移动组和移动 Agent 与移动 Agent 间的通信。一种简单的处理方法是，Agent 在迁移过程中拒绝接收任何发送给它的消息，但这种处理方法可能引起消息丢失，并会给消息发送增加负担，较好的处理方法是在移动 Agent 系统中引入消息缓存和转发机制[44]。

### 2.2.4 移动 Agent 组通信协调

用于协调移动 Agent 组间信息的接收。简单的处理的办法是在移动 Agent 系统中采用集中式同步控制策略[45]，通过集中管理对目标移动 Agent 地址信息的互斥访问来解决，但该解决办法限制了移动 Agent 移动的自主性。更好的方法是设计合理高效的通信模型，在较好地解决了前面三个问题的前提下第四个问题就能顺理成章地得到解决。

## 2.3 移动 Agent 可靠组通信设计方案

为了解决移动 Agent 的组通信问题，本文对移动 Agent 迁移过程中可靠组通信进行研究，其中组通信包括组间通信和组内通信。组间通信包括不同组内移动 Agent 之间的通信；移动 Agent 向另一个组传递信息。组内通信使得信息能够最终传递到组内的每个移动 Agent 成员。

一个移动 Agent 计算网络包括移动 Agent 和移动 Agent 平台 MAP(Mobile Agent Platform)。MAP 是分布式抽象层，提供移动和通信机制以及底层系统的安全机制。使用 MAP 的服务，移动 Agent 可以在网络上进行交互、通信和迁移。①当一个应用程序通过 Agent 系统来完成一项任务时，MAP 就会产生一个或多

个不同标识的移动 Agent，用于解决分布式环境下 Agent 标识的一致性问题。②为了能协调完成任务，这些移动 Agent 就组成了一个移动组。在移动组中，我们将组成员构成一个逻辑令牌环，在环中传递 Token，只有在组成员要离开组或移动时才将 Token 传递给下一个组成员。得到 Token 的组成员来辅助完成一些移动组管理和全序多播的功能，用于解决移动 Agent 组内的消息处理问题。为方便说明系统，在本文的论述中，我们只考虑系统只存在两个移动组的情况。在 MAE 中，我们采用邮箱机制来解决不同硬件设施环境下的数据接收缓冲问题和不同组间的移动 Agent 通信问题。③在系统中，我们在每个 MAE 中设置一个 SendBox 用来保存属于两个不同移动组之间的 Agent 未发送成功的消息，用于解决移动 Agent 组间的消息处理问题。④同时建立一个 HomeServer 来协调两个移动 Agent 组之间的通信问题，用于解决移动 Agent 的定位和移动 Agent 组间信息的接收，以便及时有效地处理 SendBox 中的消息。

在通信实验中，我们假定系统是异步的。MAP 或者 Agent 可以发生故障并在以后恢复。本文中，我们主要关注崩溃型故障(Crash Failure)，如 MAP 崩溃或移动 Agent 崩溃。我们不考虑灾难性故障，如一个组内所有移动 Agent 都崩溃。通信链路故障会造成消息或者正在迁移的移动 Agent 的丢失，并会导致网络的分区(Partition)。但在通信实验中，我们假定底层的网络提供可靠的信息传输服务。我们将研究重点放在应用层问题上，如消息无漏性、传递的原子性，并假定系统架构于一个普通网络上，如 Intranet 和 Internet。

此次研究借鉴了组通信和移动 Agent 消息通信中的研究成果，总的来说具有以下特点:(1)基于移动组内部通信的全序多播；(2)避免了由于移动 Agent 的移动造成的组间通信消息丢失；(3)综合①②③④来保证移动 Agent 通信的可靠性。

# 第三章 移动 Agent 组内可靠通信

## 3.1 组内通信服务层系统架构

### 3.1.1 移动 Agent 系统的命名及名字解析机制

移动 Agent网络环境中的每一个Agent都需要标识一个名字，利用该名字可以方便地对Agent进行管理和协作。对于移动Agent，它的名字有更为重要的意义，一个合理的移动Agent系统应该提供移动Agent的按名寻址，即不管移动Agent处于什么位置，仅根据其名字就可方便的定位它，与之协作或进行监控，这就是移动Agent的透明寻址。

这种由移动Agent系统提供的根据对象的名字找到其所在的实际位置的过程就叫做名字解析机制[46]。因此名字解析机制的任务首先是为每个对象分配一个名字，用以惟一的识别它们，其次是在交互过程中根据名字准确地定位对象。

目前大多数移动Agent系统常用的名字解析机制是基于主机名字和端口号对对象进行命名，通过域名系统(Domain Name System, 简称DNS)来完成名字解析，如Ajanta Aglets, Tacoln等移动Agent系统。显然，采用基于主机名字和端口号机制对于静态对象(如移动Agent服务器或资源)的命名和名字解析非常有效，因为这种机制在唯一命名对象的同时，自身也包含了定位对象的路由信息。但对于移动对象(如移动Agent)却存在很大的缺陷，典型的是，当一个Agent移动时，它的名字需要同步的改变以反应其最新的位置，这就使应用跟踪Agent的移动过程变得非常复杂。目前存在两种解决方法，第一种是为移动对象提供本地代理(proxy)，把对象的位置封装起来，当对象移动时，系统在本地代理中修改其位置信息。这样，就在应用级实现了位置透明性，voyager[47]就采用这种命名和名字解析机制，但是其静态对象仍使用位置相关的DNS命名方式;另一种方法是使用全局的、与位置无关的命名，即当对象的位置发生变化时名字不会随着改变，但是在名字解析时需提供名字服务，把一个符号名字映射到命名对象的当前位置，这就是基于全局命名模式的移动Agent系统名字解析机制[58]。由于全局的、与位置无关的命名使对任何对象的访问都可以以一种统一的方式进行，因此对移动对象的跟踪和通信可以完全实现位置透明性，这极大地方便了移动Agent系统导航模型和通信模型等的建立，显然它是解决移动Agent系统中名字解析问题的最佳选择。

### 3.1.2 域名字解析器 DNR

我们将整个网络划分成很多个域(Domain)，域的划分原则是保证域内主机间通信速度快、网络带宽高，一般同一局域网内的主机可划分为同一个域，也可根据具体的应用场合自由划分，这增强了系统的灵活性和可伸缩性。假设该移动Agent对象应该是“轻量级”的，即除了完成用户规定的任务外，所有其它方面的任务(如通信、控制、标识、数据管理等)都应该交给通信服务层的Agent信息控制服务来完成，这样既提高了开发效率、降低了维护费用，又增加了系统的扩

展性。

每个域内均存在一个域名字解析器(Domain Name Resolver，简称DNR)，负责为本域内所有的移动Agent提供名字服务，如Agent的注册、注销、查询、安全认证等服务，多个域的DNR还可以联合实现全局名字解析[46]。通常，为保证域内所有移动Agent系统高效可靠的名字服务，DNR需要以稳定的高带宽连接到网络上，并保持永久网络连接，若条件允许，还应对每个域指定一个主DNR和若干备用DNR。在正常情况下，由主DNR提供域内的移动Agent注册、查询等服务，而副DNR只是定期检测主DNR的工作状态。一旦主DNR发生故障，副DNR可及时检测到它的异常，从而立即接替主DNR工作，保证了移动Agent名字服务的正常运行，提高了系统的可靠性和健壮性。

在每个DNR上都有两个目录表，一个是原藉目录表(HT)，另一个是访问者目录表(VT)。DNS负责动态的维护HT和VT的信息，HT和VT中移动Agent的描述信息可根据实际情况进行扩展。下面对它们进行介绍:

原藉目录表(HT):表3-1存放了本域上创建所有的Agent的信息，当移动Agent创建时，向其创建域DNS的HT注册其信息，如下图所示，其中最主要的是该Agent的ID和其当前所在域(domain)。每一个Agent仅在创建时在HT中注册一次，AgentId在Agent的生命周期内保持不变，每次迁移时根据迁移后的地址修改其Domain值，Agent死亡时在HT中注销。

表 3-1 原藉目录表

| | 字段名称 | 字段含义 |
|---|---|---|
| 1 | AID | Agent标识 |
| 2 | Domain | 域Id |
| 3 | Home_Address | 创建时的结点地址 |

对于静态Agent，其Home_address既是创建它的结点的地址，又是它的通信地址，不能为空，对于移动Agent Home_address仅代表了创建它的主机的地址，可以为空。

访问者目录表(VT):表3-2中存放了当前域中所有的移动Agent、通信地址、状态信息和主机名，VT主要是为了定位移动Agent而设置的表，因此在VT中的Agent信息只是针对域中的移动Agent。这其中包括了本域中创建的和外域迁移过来的所有移动Agent。详细作用在第四章还会提到。

表 3-2 访问者目录表(信息注册表)

| | 字段名称 | 字段含义 |
|---|---|---|
| 1 | AID | Agent标识 |
| 2 | CurrentLocation | 移动Agent当前地址 |
| 3 | AgentState | 移动Agent当前状态 |
| 4 | HostName | 移动Agent驻留的主机名 |

### 3.1.3 通信服务构件 Communicator

在每个域内都存在若干个物理结点，每个物理结点上都应配置一个移动Agent Server来负责管理当前运行在其中的所有Agent，其中包括其自己生成且仍停留在本地的Agent和由别的主机迁移过来的Agent，该Server应能完成本物理

结点上Agent的创建、消亡、执行、迁移、通信、安全保障及资源访问等运行需求的支撑。每个移动Agent Server必须且只能在一个域服务器上注册，享受该域服务器提供的服务。

其中Server上提供通信最主要的系统构件是Communicator，它的主要功能是提供Agent间消息的发送的接收，同时负责向DNR注册新生成的Agent及和DNR协作实现Agent的全局寻址。多个Communicator之间可以协作实现消息的发送和接收。

基于上述通信服务架构，我们对移动Agent系统的通信服务进行分析，我们通过创建、迁移和寻址来详细说明通信服务过程。

## 3.2 全局寻址方式下移动 Agent 通信服务

### 3.2.1 移动 Agent 的创建

Agent创建时应进行命名和注册，其过程比较简单。移动Agent创建时，系统根据命名规则为其分配惟一的AID，同时Communicator向它所在域的DNR发出注册申请，DNR通过身份验证后将Agent的信息存入HT中(Domain值设为本域IP)。同时，DNR将AID和Agent当前地址存入VT，并将其State设为静止态。该过程是移动Agent Server通过调用DNR上的HTRegistration接口服务实现的(HTRegistration同时调用了VTAdd接口)。其过程如图3-1所示：

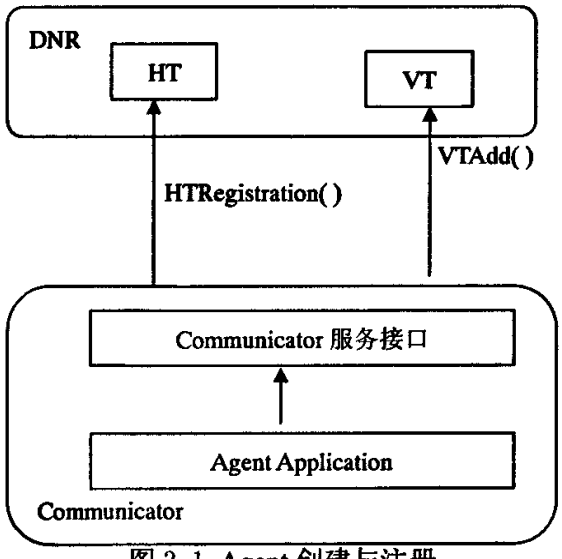


图 3-1 Agent 创建与注册

### 3.2.2 Agent 的迁移

移动Agent在从原结点成功迁移到目的结点后， Communicator向Agent原所在域的DNR发出请求，对迁移后的移动Agent的注册信息进行修改，修改分两种情况：

第1种情况是移动Agent在域内进行迁移，如图3-2中的Movement1。移动Agent A从Server1迁移到Server2，由于Server1和Server2位于同一域，HT中A的记录没有改变，只有VT中Agent的通信地址变了，所以DNR1只需修改VT中Agent对应的

通信地址即可 (①)。

第2种情况是在移动Agent不同域间进行迁移，如图2中的Movement2。Server3在成功接收该移动Agent后，通知本域的DNR2对它进行注册。DNR2检测到A从不同域中迁移过来，所以首先在VT中添加A的记录(包括其AID, Agent地址等信息)(②);同时，DNR2根据其AID解析出它的原藉域(HDN)，向原藉域的DNR1发出请求，修改HT中A的Domain参数(③)，另外，DNR2向A迁移前的域的名字解析器发出请求，删除其在VT上的注册项 (④)。

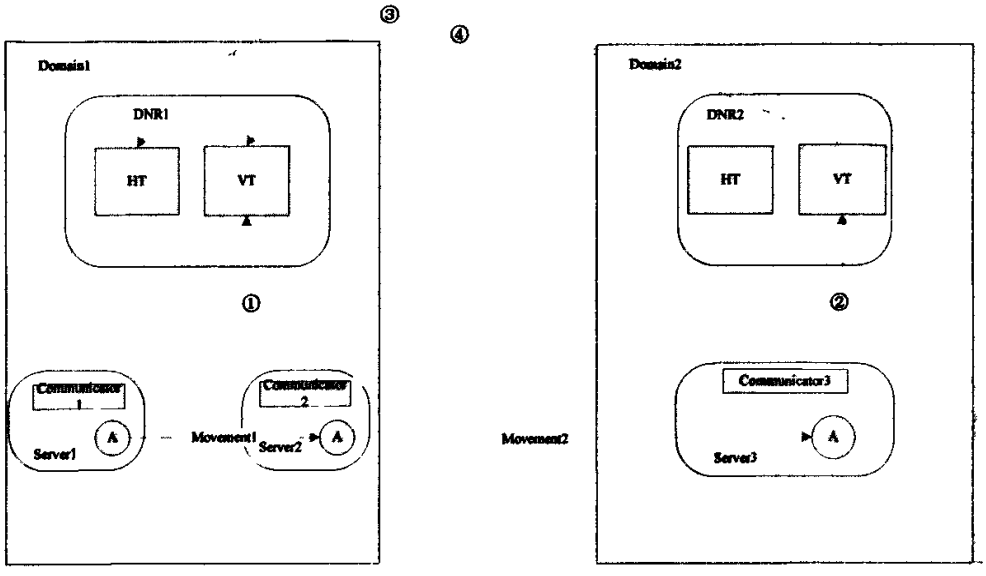


图3-2 移动Agent的迁移

### 3.2.3 Agent 的定位

当系统中的某一对象需要与某个移动Agent通信时，在通信的发起阶段，发起方所在的移动Agent Server的Communicator需要通过名字解析服务首先定位目的移动Agent。

具体的定位算法如下:

发起方通过Communicator请求本域上的DNR根据目的移动Agent的AID在VT中查找它，如图3-3的过程(①)所示。若查找成功，则返回它的描述信息，从中提取出其Agent地址。

若寻找失败，表明目的移动agent与通信发起方可能处于不同的域，定位过程比较复杂。通信发起方DNR1首先根据目的移动Agent的AID解析出它的原藉域，假设原藉域为Domain2, Domain2的DNR2根据其AID在HT中查找目的移动Agent当前所在的域，如图3-3的过程(②)。假设目的方现在在Domain3中，则DNR2向DNR3发出GetVTInfo调用请求，由DNR3在VT中查找目的移动Agent,若找到发起方返回其描述信息，最后从中提取出其Agent信息，如图3-3的过程(③)。若仍没有找到目的移动Agent，则向发起方返回目的移动Agent不存在的消息。

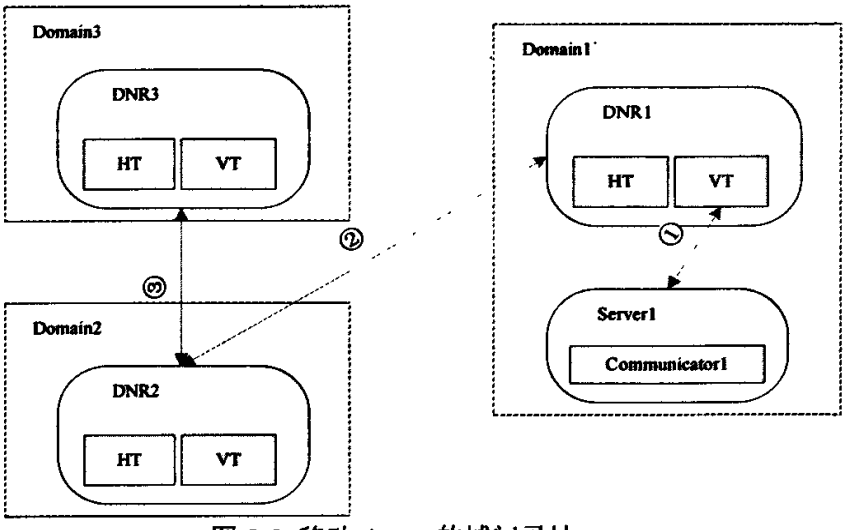


图 3-3 移动 Agent 的域间寻址

### 3.2.4 全局寻址方式下移动 Agent 通信的不足

在移动 Agent 的通信服务中，采用域名字解析器为移动 Agent 通信双方提供唯一的标识。通信服务构件实现 Agent 的全局寻址，当双方 Agent 定位后，实现 Agent 间消息的发送的接收并采用了 MailBox 消息缓存机制。虽然通信服务构件的通信双方 Agent 的全局寻址和信息发送方法能够解决移动 Agent 之间的定位和消息发送，但其性能却随着所创建的移动 Agent 的数量的增加而下降，其原因在于：①当进行全局寻址时，将从移动 Agent 所在的原籍域开始查找，根据其 AID 在 HT 中查找目的移动 Agent 当前所在的域，如果所要定位的移动 Agent 所驻留的域过多，这样将会浪费大量的时间在查找定位上。②当信息发送到目标 Agent 时，可能目标 Agent 正在迁移而导致信息无法及时到达目标 Agent。因此，如何更加有效地进行移动 Agent 的寻址定位和互相通信将在后面章节给出相应的方法。

## 3.3 自适应可靠信息传递机制-ARC

在这节中，我们描述一个通用的基于邮箱的可靠和自适应的移动代理通信方法-Adaptive and Reliable Communication。用于解决不同硬件设施下的 Agent 之间的通信和迁移过程中 Agent 信息的接收问题。

### 3.3.1 ARC 通信原理和模型

在这个方法中，每个移动代理配有一个邮箱(Mailbox)，邮箱的数据结构如表 3-3 所示。

表 3-3 邮箱数据结构表

| | 字段名称 | 字段含义 |
|---|---|---|
| 1 | MailboxId | 邮箱标识 |
| 2 | AgentId | Agent 标识 |
| 3 | MessageContent | 信息内容 |

邮箱负责这个 Agent 的消息通信的接收和发送[5,28,33,34]。邮箱只是可以移动的对象而不是 Agent，因为它不能够自主的决定自己的迁移。邮箱的迁移频率远低于其所对应 Agent 的迁移频率，并且可以设置一个阈值来确定邮箱的迁移条件。邮箱实际上就是一个消息缓冲，用来接收发送给其所属 Agent 的消息，可以把这个 Agent 叫做宿主 Agent(Owner Agent)。邮箱和宿主 Agent 是分离的，二者可以位于不同的网络节点。如图 3-4 所示：

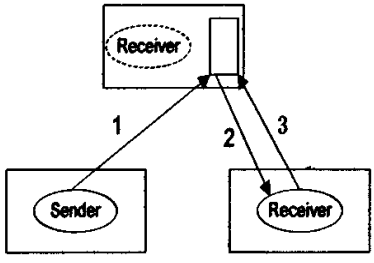


图 3-4 基于 ARC 的 Agent 通信

如图 3-4 中所示，当 Agent-Sender 要发送消息给 Agent-Receiver 的时候，它只需要将消息发送给目标 Agent-Receiver 的邮箱(步骤 1)。目标 Agent-Receiver 使用 Push 方式或 Pull 方式从自己的邮箱接收消息(步骤 2 或步骤 3)。在 Push 方式中，邮箱会维护 Agent 的最新位置信息(每次 Agent 迁移后就会通知邮箱其最新的位置)，接收到消息后，就会转发给 Agent。在 Pull 方式中，邮箱只充当消息缓冲，不主动转发消息，Agent 会按照一定的时间间隔周期性地从邮箱取消息，如此能够有利于不同硬件设施下的 Agent 通信问题。

邮箱可以和 Agent 分离开，在移动 Agent 迁移到一台新的主机上的时候，Agent 可以离开邮箱按照迁移路径从先前的结点迁移。在每次迁移之前，移动 Agent 决定是否需要带着邮箱一块到达新的站点。移动 Agent 的迁移路径和邮箱的迁移路径以及两者的关系如下定义：

定义 1：一个移动代理 A 的迁移路径表示为 $Path_a(A)$，是 A 所访问的主机($h_{a0}$, $h_{a1}$, ..., $h_{an}$)顺序的序列表。$h_{a0}$ 是 A 的第一个访问站点。路径上的主机集合被表示为 $S_a(A) = \{h_{ak} \mid Path_a(A)$上的 $h_{ak}\}$。

定义 2：一个移动代理 A 的邮箱的迁移路径被表示为 $Path_m(A)$，是邮箱按照顺序所访问的主机($h_{m0}$, $h_{m1}$, ..., $h_{mn}$)的序列集合。路径上的主机集合表示为 $S_m(A)= \{h_{mk} \mid Path_m(A)$上的 $h_{mk}\}$。按照定义，我们有 $S_m(A) \subseteq S_a(A)$以及 $h_{m0} = h_{a0}$。

定义 3：对于每个 $h_{ak} \in S_a(A)$,函数 $f_A : S_a(A) \rightarrow S_m(A)$映射了一个移动代理 A 所对应的邮箱的位置。我们有：

①(当 K=0)或(K>0 且 Agent A 同邮箱一起迁移)时

$f_A(h_{ak})=h_{ak}$　　其中 K 表示移动 Agent A 所访问的主机的顺序号。

②当 K>0 且 Agent A 不同邮箱一起迁移时

$f_A(h_{ak})= f_A(h_{a(k-1)})$　其中 K 表示移动 Agent A 所访问的主机的顺序号。

### 3.3.2 移动 Agent 迁移过程中信息发送与接收

**i)ARC 下 Agent 迁移**

一个移动 Agent 可以一直单独移动，离开他最初的信箱不带着他迁移；或者他可以一直带着信箱迁移，结果导致满迁移(FullMigeration)。图 3-5 表明了移动代理的迁移过程。

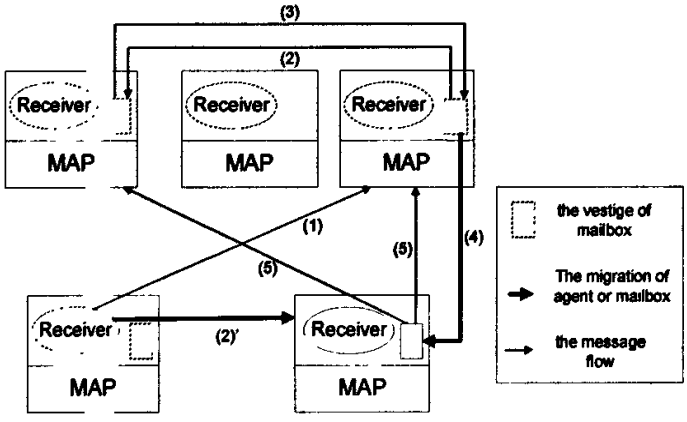


图 3-5 ARC 下 Agent 的迁移过程

如图 3.2 所示，假设存在一个 Agent A，当他移动到一个新的节点 $h_{ak}$ 之前，Agent A 先决定是否要带走邮箱 $M_A$。如果他决定这么做，他将会发送一个"MVMB"信息到 $M_A$ 告诉邮箱迁移到 $h_{ak}$。在接收到"MVMB"信息后，$M_A$ 将会发送"DEREGISTER"信息到所有驻留在 $Path_m(A)$上的主机上的 Agent，告诉他们 1)设置地址表中相应的合法标识为假，2)暂停信息传送，3)开始缓冲到达 $M_A$ 的消息。所有 $Path_m(A)$上的主机会发送"REPLY"信息给 $M_A$，其中包括已经到达 $M_A$ 的信息数量。在收集完所有的"REPLY"信息后，先在执行迁移之前等待信息到达，当再收集完所有 $Path_m(A)$上的主机上所驻留的 Agent 发送的消息后，$M_A$ 开始迁移。当 $M_A$ 到达 $h_{ak}$ 时，通过向每台 $Path_m(A)$上的每台主机发送一条"REGISTER"信息来注册他的新地址。这个"REGISTER"将会 1)重置邮箱目前 $M_A$ 的地址，2)设置合法标识为"真"，3)恢复信息传送。

ii）ARC 下 Agent 信息通信

图3-6表明了Agent通信的过程。

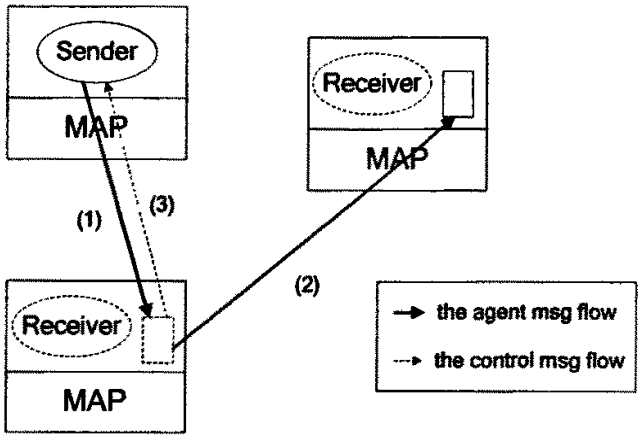


图3-6 ARC中的Agent消息通信

假设一个移动代理要发送一条消息到Agent A。他会首先检查本地MailBox表中Agent A的目前邮箱地址。如果地址存在，消息将会发送到移动Agent A的邮箱去。否则，消息将会被发送到A的初始访问的那台主机去(1)。

当主机接收到一条传送给A的消息时，将会从地址表中检查$M_A$是驻留在本机上还是在远程机子上。如果在本机上，将会把消息直接给$M_A$，否则将会把信息传送给$M_A$的目前地址(2)，与次同时发送回一个“UPDATE”信息到发送者来刷新有关$M_A$目前地址的高速缓存(3)。

通信过程中，发送者无需知道接收者的目前地址。信息会首先发送到高速缓存中的地址或者接收者的初始地址，然后再送往接收者的信箱，接收方则周期性地从邮箱中取信息。为了避免信息丢失，在信息发送和邮箱迁移间要实现信息同步。协议保证在信息到达接收者邮箱之前最多发送一次。因为同步仅仅涉及到邮箱，代理移动性的限制被删除了而异步性被增强了。无论何时移动代理都可以迁移到新的站点而不用等待信息传输。

### 3.3.3 自适应性和可靠性证明

我们假设网络没有问题，各台主机上的移动 Agent 之间能够正常通信且各移动 Agent 能够从自己的 MB 中取到信息。

①自适应性证明：

当 MB 开始迁移之前，会有以下两步：

(1)Agent 通知其邮箱 MB 开始迁移操作。

(2)MB 发送消息“DEREGISTER”，给所有在其路由路径中的节点，包括本地节点。

此时，所有驻留在 $Path_m(A)$上的主机上的 Agent 暂停信息传送。从而使得 Agent 能够对需要与之通信的 Agent 的 MB 及时作出反应。

②可靠性证明：

当 MB 发送完消息”DEREGISTER”给所有在其路由路径中的节点，包括本地

节点后，再会进行以下步骤：

(3)所有 $Path_m(A)$上的主机会发送“REPLY”信息给 $M_A$，其中包括已经到达 $M_A$ 的信息数量。在收集完所有的“REPLY”信息后，先在执行迁移之前等待信息到达，再收集完所有 $Path_m(A)$上的主机上所驻留的 Agent 发送的消息。

(4)MB 迁移到目标节点。

(5)如果 MB 与其宿主 Agent 不在同一个节点上，MB 需要发送“REGISTER”消息给其路由路径中的所有节点，否则 MB 不需要发送注册消息。

经过步骤(3)、(4)、(5)，则避免了信息的丢失，所以是可靠的。

### 3.3.4 ARC 通信存在的不足

在本节中，首先给出了 ARC 通信模型，接着详细介绍了 ARC 下 Agent 的迁移方式和移动 Agent 之间的通信过程。最后给出了 ARC 通信自适应性和可靠性证明。虽然 ARC 通信能够很好地解决移动 Agent 迁移过程中的通信问题，但是当每创建一个移动 Agent 时，将会分配一个 MailBox 作为他的信息缓存，但当移动 Agent 数量增多时，其 MailBox 无疑将会成为一笔巨大的内存资源开销，况且在次情况下再让各自的邮箱进行迁移则显然会大大增加网络上的通信流量。而且对于移动组内通信的情况却没有给出有效的解决方案即没有涉及移动组的情况。

## 3.4 采用 MailBox 存储结构的组内可靠多播通信

### 3.4.1 组内多播通信原理

在前面，虽然通过给每个移动 Agent 分配一个 MailBox 来解决移动 Agent 在迁移时无法通信的问题，但却因为 MailBox 的数量会随着移动 Agent 的数量的增加而呈递增趋势，会占用太多的内存资源，同时若 MailBox 中信息积累过多也会使得移动 Agent 在到达一个新的驻点后花费大量的时间不断地作取信息的操作而可能无法及时向其他 Agent 传递信息而降低了通信性能。同时，若是组内移动 Agent 之间的通信则可直接进行通信而省略从邮箱中取信息的操作。本节给出了移动组内可靠多播通信，只为每个移动组分配一个 MailBox。当组内信息传递时交由 Token 标识者对信息分配标识并直接多播；Token 标识者将定期从 MailBox 中提取信息,若信息中存在 AgentId 则查找本地 LocalGroup 表，若该 Agent 在本组中，就将信息发送给指定 Agent，若该 Agent 不存在本组中，则将信息送入 SendBox(SendBox 的阐述将在第四章给出)，当该移动 Agent 再次向 DNR 注册后，将会触发 SendBox 的信息发送操作，将其中的指定信息发送给迁移后重新定位的 Agent；若信息中不存在 AgentId 就将该信息在组内多播。该通信方法实现了移动组内信息传递的可靠性，旨在帮助解决 Agent 组通信时各自组内的可靠通信。

### 3.4.2 组内多播通信模型

**假定 1.1** 通信信道无错，节点间的消息传递是可靠的；

**假定 1.2** 故障只因移动 Agent 和位置的崩溃所引起，位置的崩溃导致该位置上的移动 Agent 的崩溃。

**定义 1.1** 使用 $\xrightarrow{S}$ 表示多播消息的序列号顺序。对于任一对多播消息 $MM_i$ 和 $MM_j$，如果 $MM_i.Seq<MM_j.Seq$，则可以表示为 $MM_i \xrightarrow{S} MM_j$。(其中 Seq 表示消息序列号,$i \neq j$)

**定义 1.2** 使用 $\xrightarrow{D}$ 表示多播消息的传递顺序。对于任一对多播消息 $MM_i$ 和 $MM_j$，如果 $MM_i$ 被所有的移动 Agent 接收先于 $MM_j$ 被所有的移动 Agent 传递，则可以表示为 $MM_i \xrightarrow{D} MM_j$。

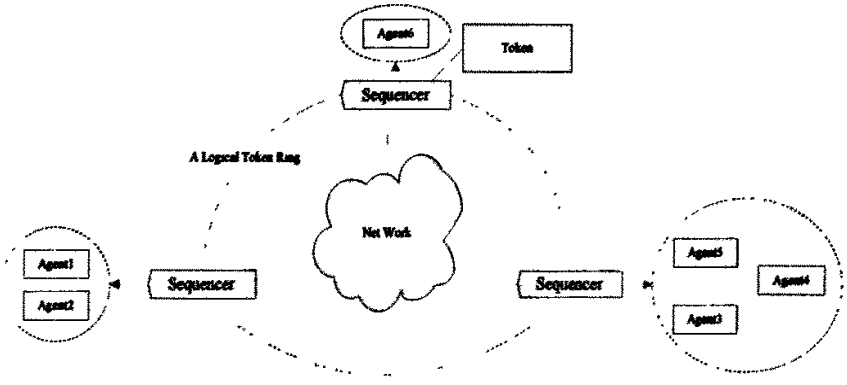


图 3-7 移动组分布示意图

如图 3-7 所示，三个 Sequencer 是 CoordinationAgent 的候选者，简单起见，我们用 Token 来标识 CoordinationAgent。逻辑环结构是实现可靠多播通信的一种常用的方法，例如在 Totem[48]和 RMP [49]中都使用了这种结构。每个候选者也是一个 Agent，每个 Sequence 管理一组 Agent,负责管理信息的发布，和对消息分配序列号，其中，只有标识为 Token 的 Sequence 才能对移动组间多播消息分配序列号，对多播消息进行排序并传递给其他的 Sequencer，而且随着已标识为 Token 的 Sequence 的迁移和离开组，新的 Sequence 会从 Token 候选表中选出而 Token 标识则传递给新的 Sequencer。

当移动 Agent 要在组内多播消息 m 或接收外组 Agent 传来的消息时，则有以下两步：

(1) 移动 Agent 发送消息 m 给 Token 标识者。

(2) Token 标识者收到消息 m(包括从 MailBox 中取到发给组内的某个 Agent 的信息)后，给消息 m 分配一个序列号，if (m.GroupId !=Token.GroupId) then {Token 标识者将 m 发送给组内的指定 Agent; }else{multicast(LocalhostGroup); }

### 3.4.3 多播通信数据结构

(1) Token

Token 是一个三元组 Token = [GID, $Seq_{Max}$, SMList]。

- GID 是移动组 g 的标识符。
- $Seq_{Max}$ 是当前传递的最大的消息序列号。
- SMList 是记录已分配序列号的多播消息的标识符和序列号的对应表，它由两个字段组成：消息标识符(MID)和消息序列号(Seq)。

(2) 消息

移动组间多播通信方法中用到的控制消息类型有 ACK、NACK、AllReceive 和 MSeq 类型，其中 ACK 表示当移动组的每个成员收到其它组成员发来消息后用此消息表示确认收到，NACK 表示如果有消息没有收到则用此消息通知消息的发送者要求重传。当多播消息发送者确定所有的组成员已收到消息时，它就通过 AllReceive 类型的控制消息通知 Token 标识者。当 Token 标识者收到 AllReceive 控制消息后，它给相关的多播消息分配一个序列号，然后通过 MSep 类型的控制消息发送给所有逻辑环上的成员。

(3) 移动 Agent 的消息发送情况表

移动 Agent 的消息发送情况表用于记录移动 Agent 发送的多播消息被哪些组成员接收到了，它是一个二元组 SS = (Message, ReceivedAgentList)

- Message 是发送的多播消息。
- ReceiveAgentList 记录了接收到此消息的组成员。

(4) 移动 Agent 的消息缓存

这部分数据由移动 Agent 携带，由 NextSeq、MessageList 和 DeliverList。

- NextSeq 是下一个可以传递的多播消息序列号。
- MessageList 是消息缓存表，其中存放那些收到的但并没有被传递消息。
- DeliverList 是消息传递表，用于存放多播消息的序列号和它们的状态，如表 3-4 所示。当移动 Agent 收到无序列号的消息后，将消息插入表中，IsDeliver 状态为“N”，当消息得到序列号后，则将对应消息的 IsDeiver 状态改为“Y”。

表 3-4 DeliverList 表

| MID | Seq | IsDeliver |
|---|---|---|
| M1 | 1 | Y |
| M2 | 2 | N |

(5) LocalGroup

LocalGroup 是一个二元组 LocalGroup= [AgentID, AgentIP]，用作 Token 标识者查找从 MailBox 中取出的发送给特定 Agent 的信息。

- AgentID 是发送信息的 Agent 的标识。
- AgentIP 是发送信息的 Agent 的当前驻留位置。

(6) 邮箱— Mailbox

Mailbox 它是移动 Agent 组的信息接收站，需要说明的是，Mailbox 采取 Agent 通信缓冲服务，Agent 每处理完一条消息就会从中再取出另一条消息直至将其中的消息全部取完，。这里，每个移动 Agent 组只有一个 MailBox。

—MobileGroupId 是移动 Agent 组的标识

—MailBoxId 是邮箱的标识

—AgentId Agent 标识

—MessageContent 信息内容

### 3.4.4 组内可靠多播通信算法

1) 多播消息发送者端算法

当移动 Agent p 要发送消息 m 时，它就在移动 Agent 的消息发送情况表中为 m 创建一个新的表项，并启动 MSA(m)线程，用来判断是否有组成员崩溃而导致 p 收不到来自它的"ACK"类型的消息。然后发送消息 m，接着等待组成员的"ACK"消息，如果收到了全部组成员的"ACK"消息，则发送一条"AllReceive"类型的消息给 Token 标识者。算法具体内容如下,算法中的组视图为移动 Agent 组成员列表。

```
mcast(m)
1 /*在发送情况表中为 m 创建一个新的表项*/
  SS.Add(m, ReceiveAgentList);
2 /*启动 MSA(m)线程, 用来判断是否有组成员崩溃*/
  MSA(m);
3 /*由 Token 标识者从 MailBox 中取出信息，根据其类型在组内发送消息 m*/
Message m=Token.getMessage();
Switch (m.type){
Case m.type= MG send(m,g); //如果 m 的消息类型是 MG 则在组内多播
Try{
Case m.type=MSeq send(m,g.Agent)} //如果 m 的消息是 MSeq 则发送给组内指定的
                                  Agent
}catch(Exception e){
sendToSendBox(m); //指定 Agent 已经迁移，将之放入发送包
}
4 /*等待组成员的"ACK"消息*/
  allReceive = false;
  while (!allReceive)
  { /*判断收到的消息是否符合条件*/
    if (Received(cm) && cm.Type == "ACK" && cm.Param.MID==m.MID)
    { /*在 ReceiveAgentList 中添加接收到消息的 AgentID*/
      ReceiveAgentList.Add(cm.Sender);
      /*是否所有的组成员都收到*/
      if (ReceiveAgentList.AgentID == currentView.P)
      { allReceive = true; /*退出等待*/}
    }
    /*如果组视图更改，则重新判断是否所有组成员都收到*/
    if (ViewChanged())
       if (ReceiveAgentList.AgentID == currentView.AgentID)
       { allReceive = true; }
  }
5 /*发送一条"AllReceive"类型的消息给令牌持有者*/
  c = new CM();
  c.Type = "AllReceive";  c.GID = g; c.sender = p;
  c.MID = 全系统唯一标识符;
```

```
    c.Param.MID = m.MID;
    sendToToken(c);
  6 /*在发送情况表中删除 m 这一项*/
    SS.Delete(m);
```

**2) 多播消息接收者端算法**

当组成员收到来自于其它组成员的多播消息时，它将消息放在消息缓存中，等待 Token 标识者发送分配给它的序列号。如果收到了来自于 Token 标识者分配序列号的“MG”类型的消息，则将消息 Para 中的消息标识符和序列号插入到消息传递表 DeliverList 中，IsDeliver 状态为“N”。然后判断序列号是否为此移动 Agent 的下一个可传递的序列号 NextSeq。如是，则将对应的消息传递，并在消息缓存删除此消息，不是则等待 NextSeq 等于它的序列号后再传递。如果在 timeout 时间里没有收到来自于 Token 标识者分配序列号的“MSeq”类型的消息，则发送给 Token 标识者一个“NACK”类型的控制消息，要求它重传。具体算法如下所示：

```
receive(m)
1 /*在消息缓存中存放收到的多播消息 m*/
  MessageList.Add(m);
2 /*为多播消息 m 设置一个 timeout 时间*/
  Set timeout(m);
3 等待多播消息 m 的序列号;
4 /*若在 timeout 时间里没收到多播消息 m 的序列号则发送一个“NACK”消息要求 Token
  标识者重传多播消息 m 的序列号*/
  if (在 timeout 时间里没有收到多播消息 m 的序列号)
  { c = new CM();
    c.Type = “NACK”; c.GID = g; c.sender = p;
    c.Param.MID = m.MID;
    goto 2;
  }
5 /*收到多播消息 m 的序列号*/
  else if (received(cm) && cm.Type==“MG” && cm.Param.MID==m.MID)
       { MulticastMessage(m);}
  6
      /*在消息缓存中删除多播消息 m*/
      MessageList.Delete(m);
```

**3) Token 标识者端算法**

当 Token 标识者收到“AllReceive”的消息后，它就给多播消息分配一个序列号，然后将它记录在 SMList 表中，接着通过多播“MSeq”消息来通知所有的组成员，算法描述如下：

```
Upon_Received_Message(cm)
  1 /*判断是否是“AllReceive”类型的消息，不是就退出*/
    if (cm.Type != “AllReceive”)
    { exit; }
  2 /*最大消息序列号加 1*/
```

$Seq_{Max} = Seq_{Max} + 1$;

3 /*将序列号分配给多播消息，并在 SMList 中保存*/

SMList.Add(cm.Param.MID, $Seq_{Max}$)

4 /*生成一条“MSeq”类型的多播消息*/

c = new CM();

c.Type = “MSeq”; c.GID = g; c.sender = p;

c.MID = 全系统唯一标识符;

c.Param.MID = cm.Param.MID;

c.Param.Seq = $Seq_{Max}$;

5 /*将消息发送给组内成员*/

MulticastMessage(c, g);

**4) 通信故障处理**

这里的故障是指持有令牌者发生了崩溃，不能在作为一个 sequencer 对多播消息进行排序了。如果 Token 标识者崩溃我们可以从 MobileGroup 组的 Token 候选表中再选，其中如前面所提到的 Token 候选表，我们采用哈希表存储结构，其中存放移动组内现有的 AgentId，每当 Token 标识者被确定时就从中将之取出作为 Token 标识者并删除他，而新的 Token 标识者则以 Token 候选表中的第一项作为候选。选出新的 Token 中后要在新的 Token 中恢复它的数据，使它继续能作为一个 sequencer 对多播消息进行排序。具体算法如下：

**createNewToken()**

1 所有的组成员将自己的消息缓存 MessageList 和不包括“IsDeliver”字段的消息传递表 DeliverList 发送给新的 Token 标识者。

2 Token 标识者收到所有组成员的 MessageList 和 DeliverList 后进行如下运算：

MList = new List();

for each MessageList

{ MList = MList ∪ MessageList; }

DList = newList();

for each DeliverList

{ DList = DList ∪ DeliverList; }

其中 MList 是中所有 MessageList 中消息的并集，DList 中是所有 DeliverList 中表项的并集。

3 从 DList 中选取最大的消息序列号，使 Token 的 $Seq_{Max}$ 等于这个序列号。

4 比较 MList 和 DList，将 MList 中尚未传递的多播消息分配序列号，并在 DList 表中记录。然后使 MList 为 Token 的 SMList。

5 将 MList 和 DList 发送给所有的组成员。

6 组成员接收到 MList 和 DList 后，将 MList 作为新的 MessageList，再根据 DList 更新 DeliverList，然后传递消息。

**5) 组成员的故障处理**

当目标移动 Agent p 收到消息 m 后，它并不立刻传递 m，而首先将 m 放进缓存器中。当消息 m 稳定后，也将是说所有的目标移动 Agent 都收到了消息 m，这时才传递消息 m，如果一个消息经过了很长的时间没有被所有的目标移动 Agent 收到，则将从移动组中移除被怀疑崩溃的移动 Agent。

**算法 3.4 移动组组成员故障处理**

MSA(m) /*当发送消息 m 时执行*/

```
1 set timeout(m);
2 unstable = unstable∪{m};  /*将消息 m 放进 unstable 集合*/
3 等待所有的组成员的确认或者超时;
4 if (收到所有的组成员确认)
  { cancel timeout(m);
    unstable = unstable-{m}; /*将消息 m 从 unstable 集合中删除*/
    exit; /*退出*/
  }
  else /*生成一个 "ChangeViewRequest" 类型的控制消息*/
  { cm = new CM();
    cm.Type = "ChangeViewRequest"; cm.GID = g; cm.Sender = p;
    cm.MID = 一个全局唯一的标识符;
    cm.Param.CID = 一个全局唯一的标识符;  /*申请组视图更改的标识符*/
    rmcast(cm); /*进行多播，通知其它组成员*/
  }
5 等待组视图更改完成;
6 if (receive(CM) && CM.Type == "View"  && CM.Param.CID == cm.Parm.CID)
  { if (p∈CM.Param.View.P) /*判断 p 是否在新视图中*/
    { 装入新视图; currentView = CM.Param.View;
    }
    else {return null;  exit; } /* p 被怀疑崩溃了，所以被移出组*/
  }
```

### 3.4.5 组内多播通信可靠性证明

引理 5.1 移动组中的任一多播消息都有唯一的消息序列号。

证明：此通信方法中，多播消息都由移动组中的 Token 标识者来分配序列号。根据系统设计，移动组中只存在唯一的 Token，任意时刻只有获得 Token 的移动 Agent 才能执行多播消息的序列号分配。当 Token 标识者收到多播消息发送者的 "AllReceive" 消息，就会为多播消息分配序列号，序列号为 $Seq_{Max}$ 加 1，并将 $Seq_{Max}$ 更新为 $Seq_{Max}=Seq_{Max}+1$。因而移动组中的任一多播消息都有唯一的消息序列号。

**定理 5.1 任一消息都能最终被移动组内的所有移动 Agent 接收。(通信的可靠性)。**

证明：

① 当消息是多播信息时

我们知道只有当组成员得到分配给多播消息的序列号后多播消息才会被传递，所以我们将分两步来证明：(1) 只有多播消息被移动组内所有的成员接收到了才分配序列号；(2) 序列号最终会被所有的组成员接收到。

首先证明(1)。由多播消息发送者端算法，当组内所有的成员发送了 "ACK" 消息后，多播消息发送者才会发送给 Token 标识者 "AllReceive" 消息，通知它为多播消息分配序列号。由假定 1.1 和假定 1.2 知道，可能由于组成员崩溃而不能回复"ACK"。因为算法中启动了 MSA 线程，即可以判断组成员是否崩溃，因

此可以将崩溃的组成员及时发现，然后更改组视图，保证在新的视图中所有的组成员都收到多播消息。假如 Token 标识者崩溃，由故障处理算法，我们可以知道所有的组成员将会最终拥有相同的消息缓存，这也保证了消息被所有组成员接收到。故障处理算法会将消息缓存中未传递的消息分配序列号。

接着证明(2)。由假定 1.1，我们知道只要所有的组成员正确，就会收到 Token 标识者分配的多播消息序列号。又由假定 1.2 我们知道组成员可能会崩溃。假如崩溃的不是 Token 标识者，我们只需要将视图简单的更新一下，去处崩溃的组成员就可以了。假如崩溃的是 Token 标识者，在故障处理算法中，我们将所有成员的 DeliverList 表项进行了并集处理产生了 DList，再将消息缓存中未分配序列号的消息分配了序列号，记录到 DList 中，最后将 DList 发送给所有的组成员，因此保证了所有的组成员都能得到多播消息的序列号。

② 当消息是指定发给组内的指定 Agent 的消息时

假设网络通信无故障且迁移过程中指定 Agent 没有 Crash

(1) 当指定 Agent 在本组时，则 Token 标识者从 LocalGroup 表中查找成功，返回“存在”信息，指定信息则将被传递给组内指定的移动 Agent。

(2) 当指定 Agent 不在本组时，则 Token 标识者从 LocalGroup 表中查找失败，返回“不存在”信息，指定信息将被送入发送包，等待某一时刻当指定移动 Agent 迁移完毕后，重新向 DNS 注册触发发送包发送消息给指定 Agent。

所以任一消息都能最终被移动组内的所有移动 Agent 接收。

证毕。

**定理 5.2** 对于任一对多播消息 $MM_i$ 和 $MM_j$，$MM_i.Seq!=MM_j.Seq \rightarrow (MM_i \xrightarrow{D} MM_j) \vee (MM_j \xrightarrow{D} MM_i)$。（多播通信的全序性）

证明：根据引理 5.1，对于任何一对多播消息 $MM_i$ 和 $MM_j$ 满足 $MM_i.Seq!=MM_j.Seq$。因而，仅需要证明 $MM_i \xrightarrow{S} MM_j \rightarrow MM_i \xrightarrow{D} MM_j$。因为所有的多播消息在被传递之前都被放入了消息缓存 MessageList 中，只有当移动 Agent 收到分配给它的序列号后才传递多播消息，所以存在三中情况：(1)先收到分配给 $MM_1$ 的序列号；(2)同时收到分配给 $MM_i$ 和 $MM_j$ 的序列号；(3)先收到分配给 $MM_j$ 的序列号。根据多播消息接收端算法，移动 Agent 只会根据序列号的顺序接收多播消息。因而，对于情况(1)和(2)，$MM_i$ 会先于 $MM_j$ 被移动 Agent 传递。情况(3)中，因为每一个移动 Agent 都有一个 NextSeq，表示下一个可以传递的序列号，它是将已传递的最大序列号加 1 得到，所以先收到 $MM_j$ 的序列号，Agent 也会等待小于 $MM_j$ 的序列号的所有多播消息被传递后，才会传递 $MM_j$。所以，$MM_i$ 和 $MM_j$ 按照其序列号顺序被移动组内所有的移动 Agent 接收，即 $MM_i \xrightarrow{D} MM_j$。证毕。

### 3.4.6 组内可靠多播通信存在的问题

虽然基于 ARC 的组内多播通信能够高效可靠地在组内传递信息但是对于移动组内 Agent A 的迁移而使得移动 Agent 组的 MailBox 中 Agent A 的信息无法到达 Agent A 而造成信息丢失从而影响组间通信的问题则在下一章给出。

## 3.5 小结

移动 Agent 组内通信是移动 Agent 通信研究的热点。本章提出基于邮箱的移动组间可靠多播通信，旨在帮助解决 Agent 组间通信中在各自组内的可靠通信，保证数据能够及时传递到组内的各个 Agent。通过 Token 标识者进行 Agent 组内通信，算法主要有两种情况：(1)若是多播消息则 Token 标识者将消息在组内多播，等待所有的组成员收到消息； 所有的组成员收到消息后，Token 标识者对消息分配一个序列号；(3) 若是发送给组内指定 Agent 的消息则直接将他发送给组内指定 Agent。本章还讨论了 Token 标识者崩溃后，如何对 Token 数据的恢复。最后，证明了基于 ARC 的移动 Agent 组内通信的可靠性。

# 第四章 SendBox 组间通信

## 4.1 SendBox 通信原理

在 Agent 通信中，往往会发生因为 Agent 的迁移而使得 Agent 之间发生消息追踪问题，或者因为信息没能发送出去而使得移动 Agent 必须带着没有发送成功的消息在网络上到处移动从而增加了大量数据在网络中的流量。

基于邮箱的移动 Agent 通信的办法虽然能很好解决在软、硬件异构环境下的数据通信问题而且降低了网络负载，但如果出现消息追踪现象时，为了防止消息丢失，移动 Agent 则要带着没有发送成功的消息进行迁移，从而使网络上的数据流量又再次大了起来，同时在上一章中也可能存在信息丢失从而影响组间通信的问题，即对于未发送成功一类的消息却没有提到该怎么办，而基于 SendBox 的组间通信方法却可以处理这类问题。同时要说明的是，发送包 Sendbox 是用来处理移动 Agent 在迁移过程中用来存储未能发送成功的消息的一个存储区，为简便起见，我们也把 SendBox 称为发送包，而 SendBox(发送包)通信方法则在此数据存储结构上对存储的信息进行协调处理，以保证信息最终能够到达目标 Agent。

在发送包机制中，分为直接模式和间接模式，每一个平台 P 有一个发送包来提供间接消息发送时的一个存储点。当接收代理驻留在节点上时，发送 Agent 直接将消息发送给接收代理，这种情况下称为直接发送模式。如图 4-1 所示

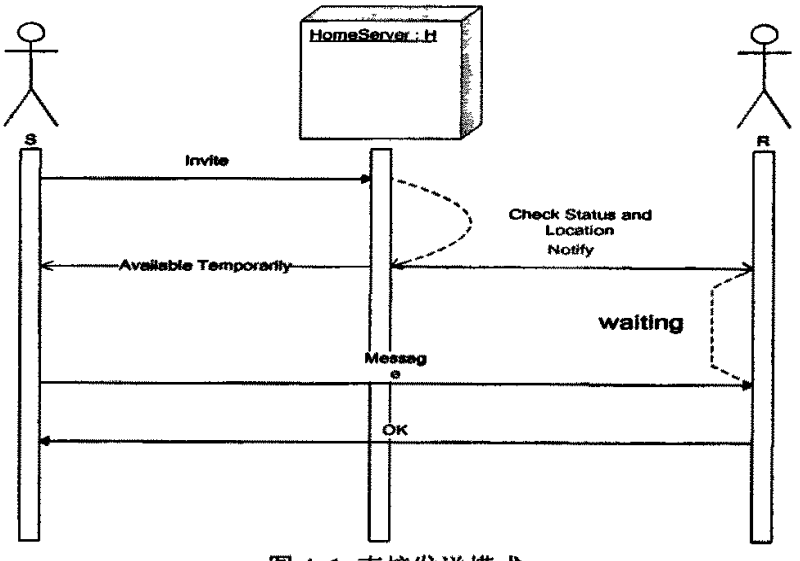


图 4-1 直接发送模式

发送方 S 驻留在平台 $P_S$ 上，接收方 R 驻留在平台 $P_R$ 上，接收方 R 注册到 H 上。每个 S 和 R 都同时处于活动或等待状态。当 S 要发送消息给 R，S 首先要发送一个 Invite request 给 H 来查询 R 的目前状态。

在 H 找到了 R 的目前状态和位置后，H 会发送一个带有 R 的 SIP URI 的 Available Temporarily 回应给 S，H 再发送一条 Notify request 给 R 要求 R 等待接收消息。这样两个 Agent 就同步了，此时双方处于等待状态。之后，S 直接发送一条信息给 R 并请求回应。在 R 接收到消息后就发送一条 OK 回应给 S，通

信就恢复到原来的状态即 S 和 R 同时转变为活动状态。一旦 R 收到一条 Notify request，R 就呆在原地等待直到信息到达后才能迁移。这样就能处理通信不可靠的问题。

每个平台都有一个通信模块采用 SendBox 发送包用于保存在直接模式下无法发送的消息，SendBox 发送包用于记录的信息如表 4-1 所示：

表 4-1 发送包信息表

| AgentID | AgentLocation | Info |
|---|---|---|
| Agent 标识 | Agent 位置 | 未发送成功的消息 |

其中，AgentID 是接收方的 Agent 标识，AgentLocation 是接收方的目前地址，Info 是要传送给接收方的 Message。

而当 R 正在迁移时，R 还没有驻留在某一个具体的站点，S 无法采用直接发送模式而只好采用间接模式。间接发送模式如图 4-2 所示：

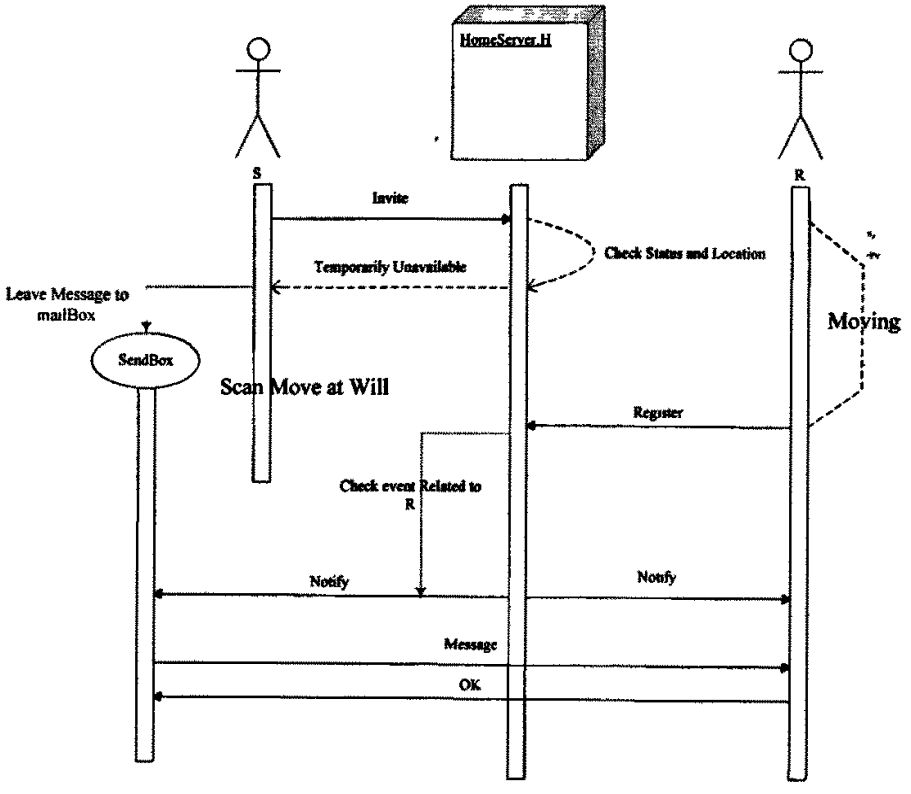


图 4-2 间接发送模式

在发送了 Invite request 信息给主机 H 后 S 变为等待状态，H 会回复一条 Temporarily Unavailable 信息给 S，S 收到 H 发来的这条信息后会把要发送的消息 $M_{SR}$ 放入到 $P_S$ 的发送包 sendbox 中并创建一个事件 $E_{SR}$ 到事件列表中。一个事件列表 Event_Info 至少包含两个主要字段，$P_S$ 的位置和接收方 ID，如表 4-2 所示。

表 4-2 发送包事件登记表

| $P_S$_Location | ReceiveID |
|---|---|
| 接收方 Agent 地址 | 接收方 Agent 标识 |

当 H 创建完事件后会发送一个 OK 信息给 S，S 然后就转为活动状态并执行其他的任务或迁移。等 R 提交了一条 Register request 给 H 后，由于注册表内容的增加会触发 H 检查事件列表是否有和 R 有关的事件，如果 $E_{SR}$ 存在，H 就会发送一个带有 R 的位置的 Notify 消息给 $P_S$ 并清除 $E_{SR}$。这时 $P_S$ 检索 $M_{SR}$ 并发送给 R。这样即使 R 在网络中频繁迁移，也能保证一旦在 R register 了后能收到原先 S 发送给他的消息。如果 $E_{SR}$ 不存在，R 要频繁迁移也没关系，这就解决了频繁迁移所引起的信息追踪问题。依靠发送包 sendbox 的帮助，Agent 就不需要带着没有发送成功的消息到处移动了。有利于降低网络负载，减少大量原始数据在网络中的流量。

## 4.2 SendBox 通信服务结构

### 4.2.1 移动 Agent 状态划分

一个 Agent 有三种状态，如图 4-3 所示。

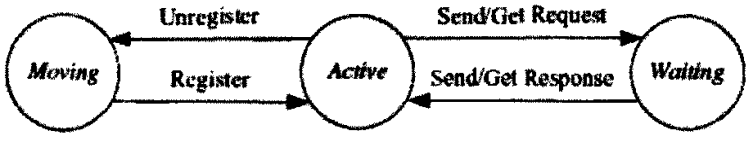


图 4-3 Agent 状态图

通常情况下，一个 Agent 处于活动的状态，这样他可以执行他所要求的操作或迁移到任何其他的机子上。当一个活动 Agent 发送或接收来自服务器或 Agent 的信息时，他会转变为等待状态继续他的通信会话。当 Agent 发送或接收回应，他将从等待状态转变成活动状态。当发送一个 Unregister 要求到主服务器 H 并得到 H 的认可后，agent 转变成移动状态并迁移到其他的节点上。在到达一个节点后，他再向 H 发送 register 信息然后从移动状态转变为活动状态。

另外，我们要为等待状态的 Agent 设置一个阈值以避免当一条信息在低层通信时遇到故障而没能及时到达目的主机从而使处于等待状态的 Agent 无限地等待。当超出这个阈值时，我们将使 Agent 重做处于等待状态前的工作。

### 4.2.2 Agent 信息显示结构

在我们的发送包通信机制中，我们采用 SIP 信息格式来表示 Agent 的传递信息，因为 SIP 提供了可行的方法，诸如 notify, invite 等。如图 1 中所示，两个信息的模板大致上表明了 SIP 的信息格式[50]，图 4-4（a）是一条 invite 需求的需求信息。图 4-4（b）是一个 Available Temporarily 的回应的例子。

```
INVITE sip:agent_R@home.agent.com
Via: home.agent.com
To: agent_R <sip: agent_R@home.agent.com >
From: agent_S <sip:agent_S@node1.agent.com>
Content-Type:  text
```

图 4-4(a) SIP Invite 请求信息格式

302 Moved temporarily
Contact: sip:agent_R <sip:agent_R@1.1.1.10>
From: Home <sip: Home@home.agent.com>
To: agent_S <sip: agent_S@node5.agent.com>
Content-Type: text

图 4-4(b) SIP Invite Available Temporarily 信息格式

每一个 Agent 有一个唯一的 SIP URI(SIP Uniform Resource Indicators)，格式为 agent_id<agent_id@hostname>来表示 Agent 的名字和目前的位置，这里 agent_id 是唯一的 agent 名字，在 Agent 的生命周期中保持不变，SIP URI 的主机名字可能随着 Agent 驻留的主机的不同而改变。

### 4.2.3 Agent 通信协作管理

所设计的通信方法要求主服务器 H 作为监督者来管理和跟踪 MAS 中所有的 Agent。Agent 可以在 H 的控制下，移动到任何一个平台去执行指定的任务或进行数据采集。无论是被分派还是安全抵达目标主机，每个移动 Agent 必须发送一个 Register request 到 H。在 Agent 准备迁移时，都要向 H 发送一个 Unregister request。所以在 H 上留有一张 Register 信息表 RegisterInfo 用于记录 Agent 的 Register 信息，表的结构如下：

表 4-3 Agent 注册信息表

| AgentID | CurrentLocation | AgentStatus | GroupId |
|---|---|---|---|
| Agent 标识 | Agent 当前地址 | Agent 状态 | Agent 所在的移动组标识 |

表中 AgentID 是移动 Agent 的标识，用以唯一标识一个移动 Agent；CurrentLocation 是移动 Agent 所驻留的那台主机 IP；AgentStatus 是移动 Agent 目前所处的状态；HostName 是移动 Agent 所在主机的计算机名。当移动 Agent 迁移到一台主机时，RegisterInfo 表就会相应地增加一条记录，当移动 Agent 迁移时，RegisterInfo 表就会相应地删除一条记录。

## 4.3 SendBox 组间通信算法

**Agent，HomeServer 和 Platform 的算法概述如下：**

**Agent:**

**Case1: Send()**

**X requests for sending MSR to R:**

**X sends Invite to H, waits for a response from H,**

**State(X) =Waiting.**

**Case2: Receive ()**

**X receives a message m from (S|H| PS ):**

**(1)If m=Moved Temporarily with R's SIP URI, /***

**X=S**

**Sends MSR to R, State(X) =Waiting.**

(2)If m=Temporarily Unavailable, /*X=S
Sends $M_{SR}$ to $P_S$'s sendbox, sends Subscribe to
H, State(X) = Waiting.
(3)If m = $M_{SR}$ from (S| $P_S$ ), /* X=R
Replies OK to (S| $P_S$ ), State(X) = Active.
(4)If m =Notify from H, /* X=R
Waits for receiving $M_{SR}$ , initiates a waiting timer,
State(X) = Waiting.
(5) If m = OK,
State(X) =Active.
Case 3: Migration
(1)If agent A reaches a platform, sends a Register to
H, then receives OK, State(X) = Active.
(2) If S is ready to migrate, sends Unregister to H,
State(X) = Moving.
**HomeServer:**
Case 1: Receives an Invite from S for requesting R's
SIP URI:
(1)If R has registered, replies Moved Temporarily
with R's SIP URI to S, sends Notify to R.
(2)If R is unregistered, replies Temporarily
Unavailable.
Case 2: Receives Subscribe from S:
Creates an event $E_{SR}$ with two fields, receiver ID
and $P_S$ 's location, in its event table.
Case 3: Receives Register from agent A:
Checks if some events $E_{XA}$ exist
(1)If true, for each $E_{XA}$ sends Notify to A and $P_S$ ,
clears $E_{XA}$ .
(2)If false, updates A's SIP URI, changes A's status
to Registered.
Case 4: Receives Unregister from agent A:
Changes A's status to Unregistered.
**Platform:**
Case 1: Receives a Notify with R's SIP URI from H:
(1)Sends $M_{SR}$ to R, waits for receiving OK from
R.
(2)After receiving OK from R, removes $M_{SR}$ from SendBox.
Case 2: Receives $M_{SR}$ from a local agent:
Saves $M_{SR}$ in its sendbox.

## 4.4 SendBox 可靠通信证明

**降低网络负载** 在每个移动 Agent 平台上都留有一个发送包（SendBox），当发送方从 HomeServer 处得知接收方正在迁移中时，便把要发送的信息放入发送包然后自己就迁移了，等接收方到达一个站点后，会到 HomeServer 处注册，由 HomeServer 向接收方发送一个 Notify 控制信息叫接收方暂时别迁移并通知发送包来发送信息从而避免了发送方因为原先没能发成信息而带着冗余数据在网络上迁移，从而降低网络负载。

**移动 Agent 异步自主的能力** 由于 HomeServer 会在发送方要求与接收方通信时用 Check Status and Location 的方法定位接收方目前的地址，当 Agent 迁移完后向 HomeServer 注册时 HomeServer 会用 Check Event 的方法来查找某一个平台的发送包中是否有信息要发送给接收方，所以地址定位和检查是否在某一平台的发送包中有未发送的信息这些工作都由 HomeServer 来做而与移动 Agent 无关，接收方主机与目标方主机可以断开连接，这样移动 Agent 就可以安心地执行自己的任务，独立地创建自己的进程，随时可以移动而且要发送的信息总能在某个时刻到达接收方，进行异步、自主地操作，保证了信息的可靠到达。在这里假定我们的网络通信是正常的。

**动态自适应性** 由于在网络环境中可能有突发事件发生，所以当发送方与接收方进行传输时，通信平台应时刻检测网络通信方面的故障，并设置一个等待阈值，当故障发生时在规定的时间内若不能排除故障则将消息放入发送包等待下次当 HomeServer 做 Check Event 的操作后再重传，一旦超过阈值则彼此立即取消通信，继续做接下来要做的任务或迁移，这样移动 Agent 可以感知运行环境，并对变化自主、迅速地做出反应。

## 4.5 小结

本章中，针对因为 Agent 的迁移而使得 Agent 之间发生消息追踪进而影响移动组间通信问题，或者因为信息没能发送出去而使得移动 Agent 必须带着没有发送成功的消息在网络上到处移动从而增加了大量数据在网络中的流量的问题，我们给出基于 SendBox 的组间通信方法并对其有效性进行了证明，结合第三章的移动 Agent 组内可靠通信可实现移动 Agent 迁移过程中组间可靠通信。

# 第五章 可靠组通信实验

第 3 章给出了移动组的组内可靠通信方法，第 4 章提出了移动 Agent 迁移过程中移动组间的可靠通信方法，因此本章将在 IBM 公司的 Aglets 平台上对以上两种方法进行实现，并给出实验结果和性能分析。

## 5.1 实验环境和移动 Agent 平台介绍

### 5.1.1 实验环境介绍

在实验中，移动 Agent 组通信系统是建立在 Aglets 平台之上，通过继承 Aglet 类来实现本文中的协议，每一个协议都以 Aglet 实例的形式实现[51,52]。移动 Agent 组通信系统将 Aglets 提供的通信作为它的低层通信机制，在这个基础上实现移动组的组成员管理协议和移动组的全序可靠多播协议。
本实验在局域网中进行，网络速度为 100M，并利用了 Sun 公司的 Solaris 工作站和装有 Windows XP 系统的兼容机作为硬件平台，其中 Solaris 工作站的型号为 sun-blade-100，内存 256M，兼容机的配置为 PIII，内存 256M。

### 5.1.2 移动 Agent 平台介绍

Aglets 是由 IBM 东京实验室开发的移动 Agent 系统，它提供了一个实用的平台——AgletWorkbench[53]，其设计简洁小巧，紧随 Java 模型，特别是它的源代码公开，开发文档详尽，非常适合开发。Aglet 是一个能够自动从一台主机移动(或被派遣)到另一台主机上的 Java 对象，能够进行这种迁移的前提是在这些主机中预装了 Tahiti 服务器，它是用 Java 实现的 Aglet 服务器程序，为本地或者到达的 Aglet 提供一个 Aglet 上下文环境（Context）以供其运行。

（1）Aglet 的系统框架

Aglet 的传输协议是 Aglet Transfer Protocol（简称 ATP）[54]，是一个分布式的基于 Agent 系统的应用程序级协议。Aglet 的系统框架如图 5-1 所示，图中展示了 Aglet 执行的若干阶段。首先当一个正在执行的 Aglet 想要将自己送到远端时，会对 Aglet Runtime 层发出请求；接着 Aglet Runtime 层把 Aglet 状态信息与代码转化成序列化的字节数组；这时如果请求成功，系统会将字节数组传送至 Agent 传输通信层（Agent Transport and Communication Interface，简称 ATCI）处理，此层提供 ATP 接口，在此 ATP 为一个简单的应用层协议。之后，系统会将字节数组附上相关的系统信息，如系统名称以及 Aglet 的 ID 等，并以比特流方式通过网络传送至远端机器。远端机器利用 ATCI 层提供的 ATP 接口接收到传来的字节数组及系统信息，然后 Aglet Runtime 层对字节数组反序列化，得到 Aglet 的状态信息与代码，此时 Aglet 便可在远端机器上执行。

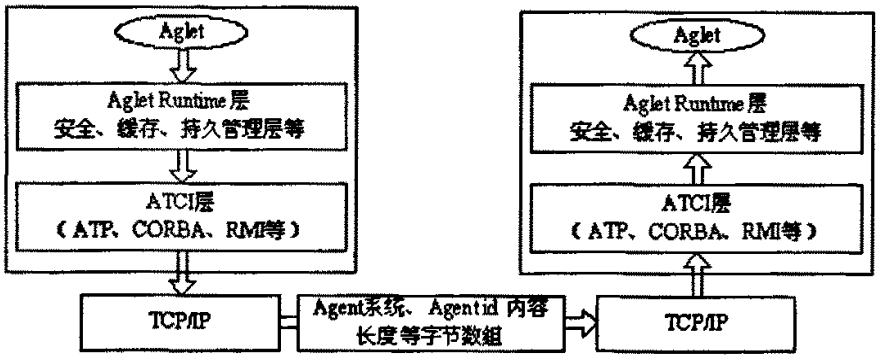


图 5-1 Aglet 的系统框架图

(2) Aglet 的对象模型

Aglet 系统提供一个上下文环境来管理 Aglet 的基本行为：如创建（create）Aglet、复制（clone） Aglet、分派（dispatch）Aglet 到远端机器、召回（retract）远端的 Aglet、暂停（deactive）、唤醒（active）Aglet 以及清除 Aglet 等，如图 5-2 所示：

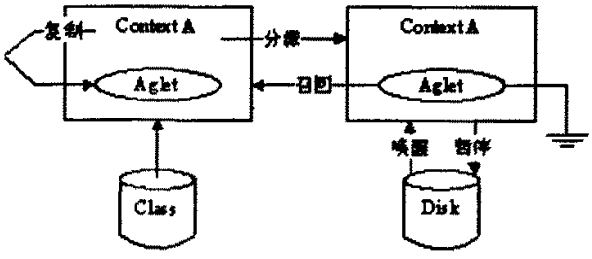


图 5-2 Aglet 的对象模型

Aglet 之间采用消息传递的方式来进行通信。此外，基于安全上的考虑，Aglet 并非让外界直接存取其信息，而是通过一个代理（proxy）提供相应的接口与外界沟通，如图 5-3[65]所示。这样做有一个优点，即 Aglet 的所在位置会透明化，也就是 Aglet 想要与远端的 Aglet 沟通时，只在本地主机的上下文环境中产生对应远端 Aglet 的代理，并与此代理沟通即可，不必直接处理网络连接与通信问题。

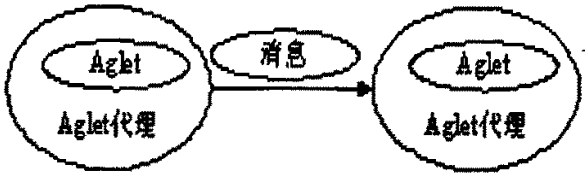


图 5-3 Aglet 的基本通信模型

## 5.2 移动 Agent 组通信系统 CISOM 实现

### 5.2.1 移动 Agent 组通信系统 CISOM 的设计与实现

① 移动 Agent 组通信空间

我们用空间和子空间的概念对移动 Agent 组通信系统做一个描述。

一个空间指的是一个可以跨越分布式系统并且各种对象可以共存其中的对象空间，而且，生存在这个空间中的同一个特定类型的所有对象都可以被同时调用。

一个空间由单个的容器构成，这些容器成为子空间。一个空间只能被它的这些子空间所感知，并通过将这些子空间连接在一起构成，这些子空间，甚至包括远程的子空间，可以互相连接起来。通过连接子空间，我们就建立了一个无向图，也就是一个子空间的网格图。这个图中所有可以抵达的子空间就构成了一个空间，如图 5-4 所示。

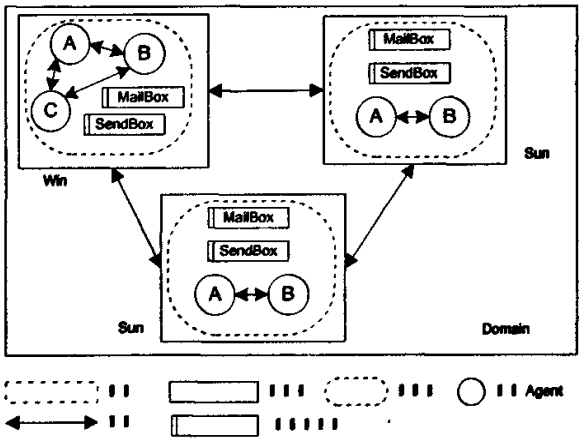


图 5-4 移动 Agent 组通信空间构成图

图 5-4 中，各台工作站所在域是一个空间，而各台工作站上的移动组是个子空间，各个子空间之间进行互相通信和内部信息的传递。

② 移动 Agent 组通信系统 CISOM(Cooperative Information System Oriented Middleware)结构与原理

CISOM 系统结构由分布在不同信息节点上的互相协作的一组 Agent 子系统，以及管理各 Agent 子系统中移动 Agent 信息和发送包事件信息的中介 Agent 系统所组成，如图 5-5 所示。

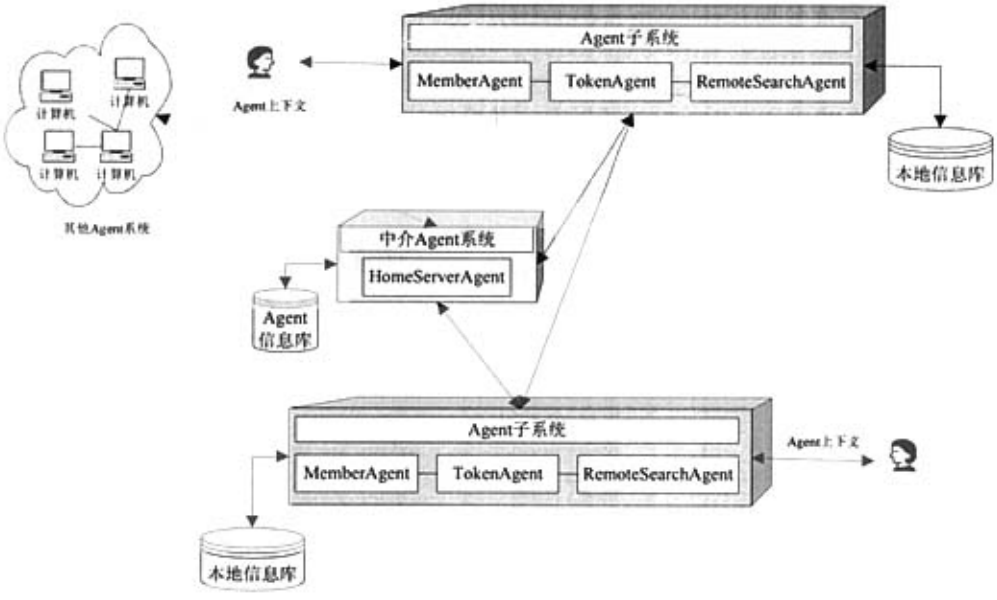


图 5-5 CISOM 系统结构与原理示意图

1、中介 Agent-HomeServerAgent

中介 Agent 在整个 CISOM 系统是至关重要的，是实现 CISOM 系统分布式透明性的关键。他主要用于解决系统中移动 Agent 的定位和实现协调移动 Agent 组间信息接收的部分功能。其界面如图 5-6 所示：

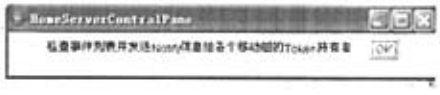

图 5-6 HomeServerAgent 控制面板

2、MemberAgent

MemberAgent，顾名思义他就是组成员，可以在组内传递信息，将用户要求转化为agent能够识别的命令，并会自动为用户寻找TokenAgent提供相关的服务。其界面如图 5-7 所示：

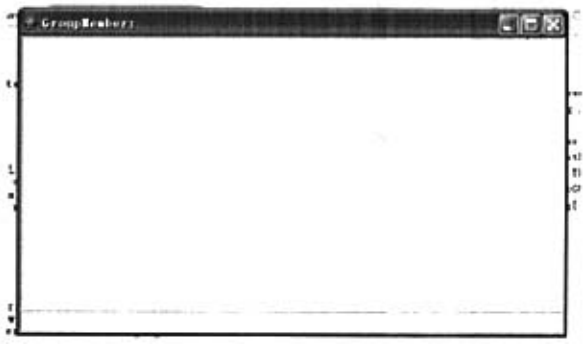

图 5-7 组成员 Agent 对话框

3、TokenAgent

TokenAgent 接收来自 MemberAgent 代表用户的请求，根据不同的请求采取相应的操作，可以完成各种不同的任务，小到将信息站 MailBox 中的信息在组内进行传递，达到一次复杂的远程查询会话，其界面如图 5-8 所示。如图 5-9 所示，LocalGroupMember 表中记录移动 Agent 组成员信息，如图 5-10 所示，RegisterInfo 表中显示移动组的可访问性信息。


图 5-8 Token 持有者对话框

| | AgentId | CurrentLocation | GroupId | isToken |
|---|---|---|---|---|
| 1 | fb6e3c5688ef4a09 | atp://zhou:4434/ | 1 | true |
| 2 | fd036e541c60dbfd | atp://zhou:5000/ | 1 | false |
| 3 | c759e9a11e39c998 | atp://zhou:4434/ | 1 | false |

图 5-9 LocalGroupMember 表中相应的 Token 持有者信息

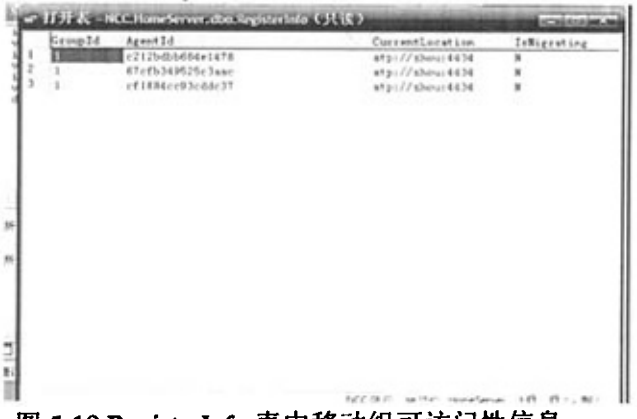

图 5-10 RegisterInfo 表中移动组可访问性信息

4、协作信息查询 RemoteSearchAgent

协作信息查询 RemoteSearchAgent 是可移动 Agent。他从 TokenAgent 处接收远程 Agent 信息查询请求，在从中介 Agent 系统获得目标 agent 信息后，将信息传送给 TokenAgent，完成信息查询任务后自动销毁。

③ 移动 Agent 组通信系统框架

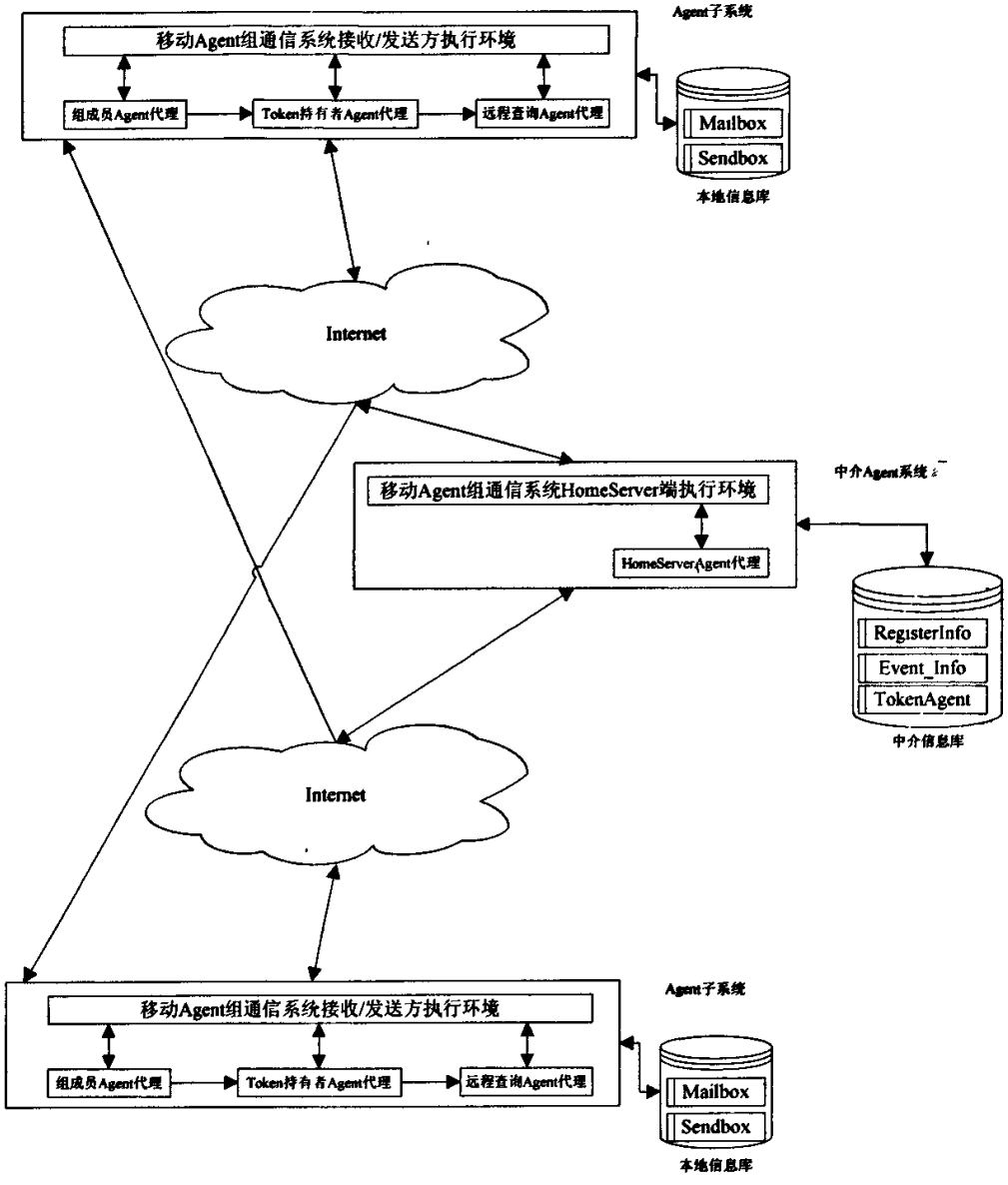


图 5-11 移动 Agent 组通信系统框图

CISOM 整个系统框架如图 5-11 所示。

CISOM 中介 Agent 系统：提供移动 Agent 的定位和协调移动 Agent 组间信息的接收。

CISOM Agent 子系统：处理移动 Agent 组内信息的传递。

整个系统工作流程如下：

(1) 用户向 CISOM Agent 子系统前端 Frame 窗口输入数据或命令请求。

(2) MemberAgent 代理判断是否为本地操作。

(3) 如果是本地操作，直接启动 CISOM Agent 子系统 TokenAgent 代理，进行相关操作，把结果返回给 CISOM Agent 子系统前端 Frame 窗口。

(4) 如果不是本地操作，由 TokenAgent 代理创建远程协作查询 agent 代理到 CISOM 中介服务器，并提出查询信息的请求，然后把得到的结果返回 TokenAgent 代理，启动 CISOM Agent 子系统进行发送远程信息的操作。

关键代码如下：

```
addMobilityListener(new MobilityAdapter(){
                public void onArrival(MobilityEvent e){
                    self_id=getAgletID();
                    System.out.println("my Self id is:"+self_id);
                    AgletContext cxt=getAgletContext();
                    try{
                        Enumeration aglets=cxt.getAgletProxies();
                        while (aglets.hasMoreElements()){
                            AgletProxy tmp=(AgletProxy)aglets.nextElement();
                            AgletInfo inf=tmp.getAgletInfo();
                            System.out.println("Agent信息:"+inf);
                            String agName=tmp.getAgletClassName();
                            if(agName.endsWith("HomeServerAgent")){
                                    //找到HomeServerAgent
                                sid=tmp.getAgletID();
                                sname=tmp.getAgletClassName();
                                System.out.println("the sname is:"+sname);
                            }
                        }
                        System.out.println("the AgletContext of HomeServer
is:"+cxt);
                    }catch(Exception ex){
                        ex.printStackTrace();
                    }
                }
                });

public boolean handleMessage(Message msg){
            if (msg.sameKind("QueryRemoteAgentStatus")){//查询远程Agent状态
                String
remoteAgentId=msg.getArg("QueryRemoteAgentStatus").toString();
                Hashtable
record=messageOperation.QueryRemoteAgentStatus(remoteAgentId);
                 Message queryResult=new Message("RemoteAgentInfo");
                 queryResult.setArg("RemoteAgentInfo", record);
                 msg.sendReply(queryResult);
```

```
        }
        return true;
          }
```

如图 5-12 中，我们在每个移动 Agent 平台上初始化拥有 3 个组成员的移动组。图 5-13 是 Solaris 平台下组成员的初始化信息。

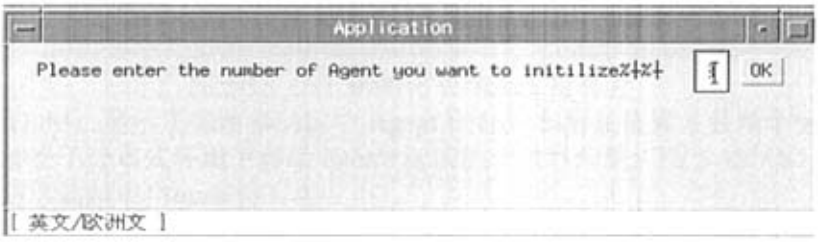


图 5-12 移动组的初始化

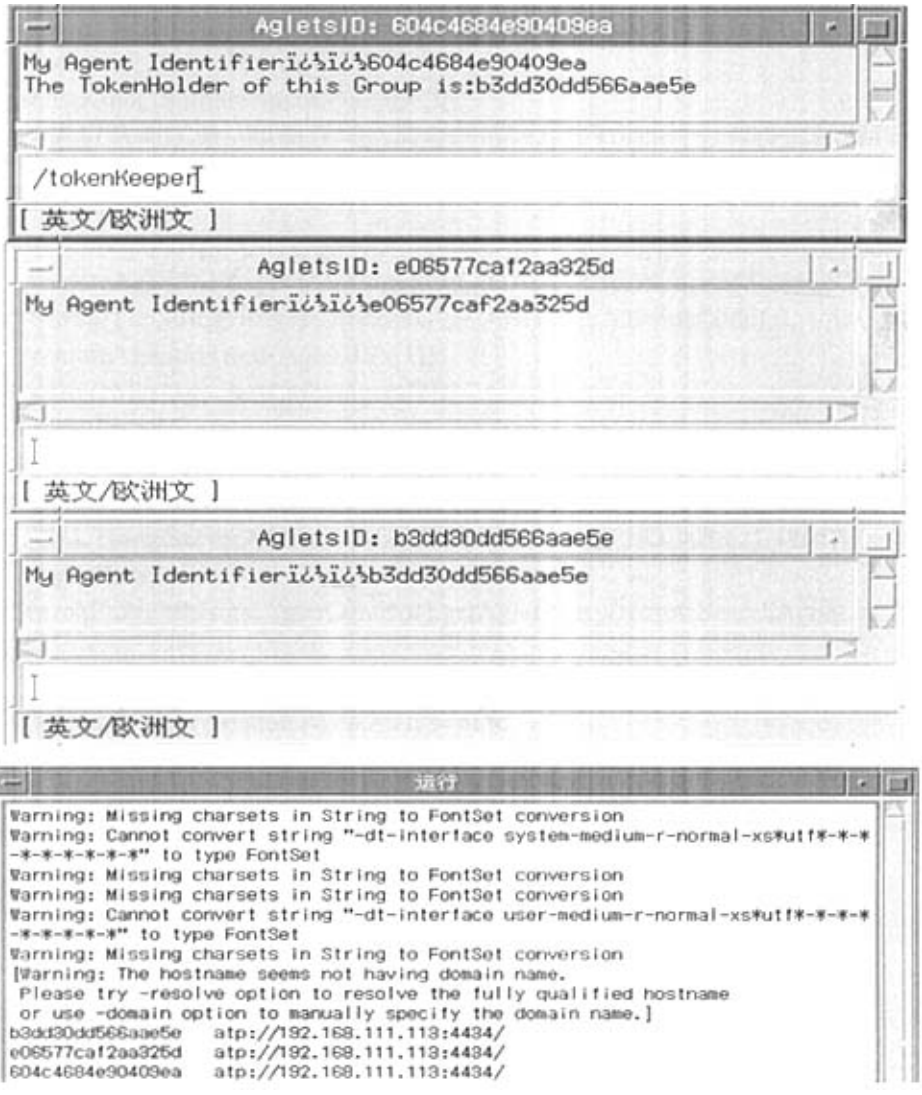


图 5-13（a）初始化后组成员标识

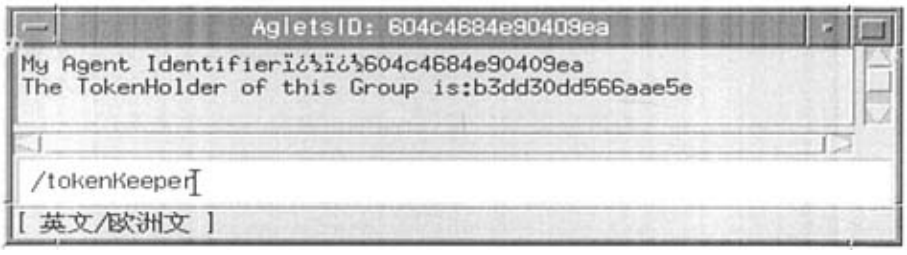


图 5-13（b）初始化后 Token 持有者

图 5-13（a）中，显示了当前移动组中 Agent 的标识和所在位置并且每个标识是唯一的，解决了分布式环境下移动 Agent 标识的一致性问题。图 5-13（b）中，显示了当前移动组中 Token 持有者。

### 5.2.2 移动 Agent 组间可靠通信功能的实现

关键代码如下：

```
if (msg.sameKind("RemoteAgentMessage")){
        //派出查询代理，根据查询代理返回的信息由Token持有者将信息发到另一个移动组的MailBox
        try{
            //remoteAgentMessage样式:remoteAgentMessage&AgentId&Message
        remoteAgentMessage=(Message)msg.getArg("RemoteAgentMessage");
        String[] s=StringUtils.split(remoteAgentMessage.toString(),'&');
        remoteAgentId=new AgletID(s[1]);
        deliverMessage=new Message(s[2]);
        AgletProxy
remoteSearchAgent=this.getAgletContext().createAglet(this.getCodeBase(),
"groupCommunication.RemoteSearchAgent", null);
        URL HomeServerAddress=new URL(new String("192.168.111.110"));
        AgletProxy
remoteSearchProxy=this.getAgletContext().getAgletProxy(remoteSearchAgent.getAgletID());
        remoteSearchProxy.dispatch(HomeServerAddress);
        Message QueryRemoteAgentStatus=new
Message("QueryRemoteAgentStatus");
        QueryRemoteAgentStatus.setArg("QueryRemoteAgentStatus",
remoteAgentId);
        FutureReply  reply=remoteSearchProxy.sendAsyncMessage(new
Message("QueryRemoteAgentStatus"));
        Message remoteAgentInfo=(Message)reply.getReply();
        Hashtable
queryRecord=(Hashtable)remoteAgentInfo.getArg("RemoteAgentInfo");
        String isMigrating=(String)queryRecord.get("isMigrating");
        remoteAgentLocation=(String)queryRecord.get("currentLocation");
        if (isMigrating.equalsIgnoreCase("n")){
```

```
            messageOperation.sendToAgent(new URL(remoteAgentLocation),
remoteAgentId, deliverMessage);
        }else{
            //送入发送包，向HomeServer登记发送包事件

    messageOperation.sendToSendBox(remoteAgentId.toString(),remoteAgentLocati
on,deliverMessage.toString());

        }
        }

public synchronized void sendToSendBox(String remoteAgentId,String
remoteAgentLocation,String deliverMessage){
    //送入发送包并向HomeServer注册未发送成功的消息
      DBConnection connection=new DBConnection();
      Connection conn=connection.getLocalConnection();
      Connection conn1=connection.getHomeServerConnection();
      String sql1="insert into Event_Info values(?,?)";
      String sql="insert into Sendbox(AgentID,AgentLocation,UnsentInfo)
values(?,?,?)";
      try{
      PreparedStatement stmt=conn.prepareStatement(sql);
      PreparedStatement stmt1=conn1.prepareStatement(sql1);
      stmt.setString(1, remoteAgentId);
      stmt.setString(2, remoteAgentLocation);
      stmt.setString(3, deliverMessage);
      stmt1.setString(1,remoteAgentId);
      stmt1.setString(2, remoteAgentLocation);
      stmt.execute();
      stmt1.execute();
       stmt.close();
       conn.close();
       stmt1.close();
       conn1.close();
      }catch(SQLException e){
          e.printStackTrace();
      }
 }
```


图 5-14 组成员迁移前所在平台

如图 5-14 所示为组成员迁移前所在平台 atp://zhou:4434。


图 5-15 移动组中要迁移的组成员

如图 5-15 所示为移动组中要迁移的组成员，其标识为 3d81c46ba2d77fd5。

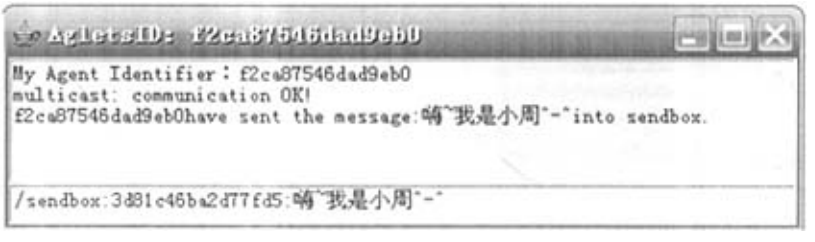


图 5-16 发送包发送信息成功界面

如图 5-16 所示为 Token 持有者向迁移中的组成员 3d81c46ba2d77fd5 发送信息“嗨~我是小周^-^”，由于组成员在迁移中，故将信息转入发送包中。如图 5-17 所示的发送包中的最后一条信息。

打开表 —ZHOU.MobileAgentGroup.dbo.Sendbox（只读）

| | AgentID | AgentLocation | UnsentInfo |
|---|---|---|---|
| 1 | 2be011c6e1a99589 | atp://zhou:5000 | HelloWorld! |
| 2 | 2be011c6e1a99589 | atp://zhou:5000 | 各位同学老师，早上好 |
| 3 | 4a76ce78c15dc57f | atp://zhou:4434/ | nice |
| 4 | f45948eac3e9f95b | atp://zhou:4434/ | lll |
| 5 | 26d726a886d4afd3 | atp://zhou:4000/ | zhou |
| 6 | 14a972f4ddbbf201 | cf794b1074a36c40atp://zh... | zhou1 |
| 7 | ab706cb3e46b329f | atp://ncc:4434/eae5aed82... | ncc |
| 8 | 6fb03833cdd4fa3b | atp://zhou:4434/40375135... | zhou2 |
| 9 | 6bcb0ef62a93eeb6 | atp://ncc:4434/8d396cb1b... | ncc1 |
| 10 | d024f8c3739b4d82 | atp://SHMTU-cs04:4434/c2... | solaris |
| 11 | 3d81c46ba2d77fd5 | atp://zhou:4434/f2ca8754... | 嗨~我是小周~-~ |

ZHOU (8.0) sa (53) MobileAgentGroup 11 行 行 1，列 1

图 5-17 发送包数据图


图 5-18 组成员迁移目的地平台

如图 5-18 所示为组成员 3d81c46ba2d77fd5 的迁移目的地 atp://ncc:4434。

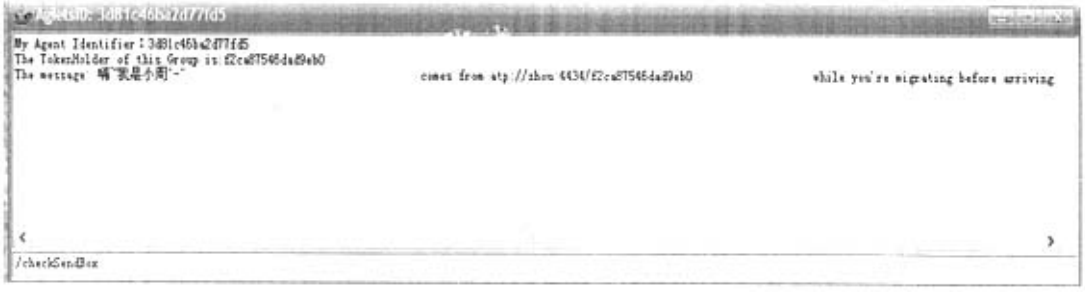


图 5-19 迁移完毕从发送包中接收数据

如图 5-19 所示为组成员 3d81c46ba2d77fd5 迁移完毕后，触发发送包发送信息并成功地接收到。实现了 Agent 透明寻址和组内 Agent 迁移时移动 Agent 组通信协调问题。

### 5.2.3 移动 Agent 组内可靠通信功能的实现

关键代码如下：

```
public synchronized void multicastMessageInGroup(Vector muliticastMessage){
        context = this.getAgletContext();
      URL url=context.getHostingURL();
     try{
      InetAddress group=InetAddress.getByName(url.toString());
      int port=6789;
      MulticastSocket socket=new MulticastSocket(port);
      byte[] buffer=new byte[1000];
      DatagramPacket datagram=new
DatagramPacket(buffer,buffer.length,group,port);
      Iterator muliticastMessage1=muliticastMessage.iterator();
      while (muliticastMessage1.hasNext()){
          String message=(String)muliticastMessage1.next();
          datagram.setData(message.getBytes());
          socket.send(datagram);
      }
     }catch(UnknownHostException e){
         e.getMessage();
     }catch(IOException e){e.getMessage();}
 }
```

如图 5-20 所示为组内可靠通信示意图。实现了移动 Agent 与移动组的消息处理问题，使得消息能够可靠地传递到每个组成员。

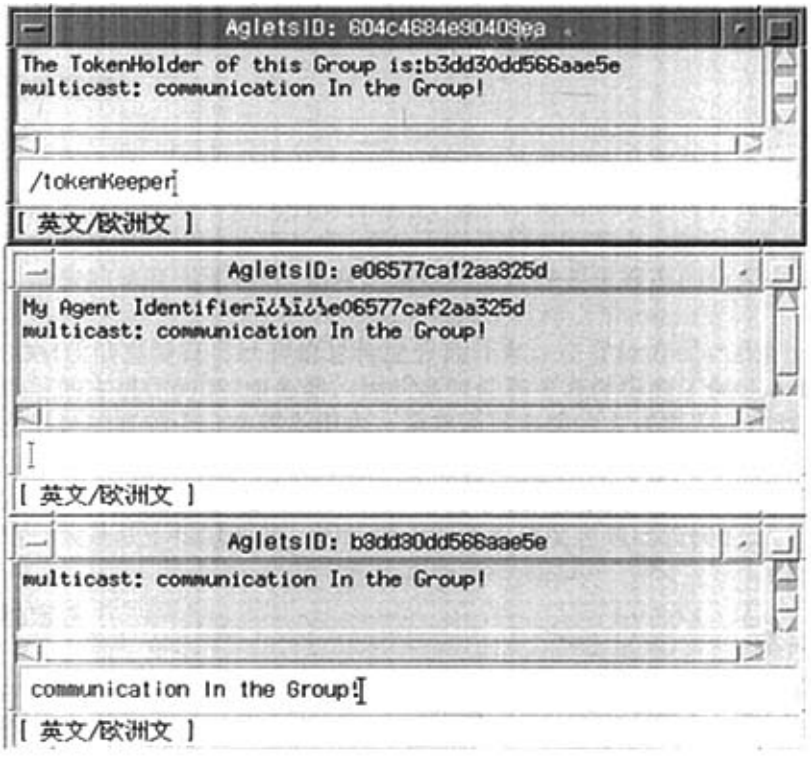


图 5-20（a）Solaris 环境下组内可靠通信示意图

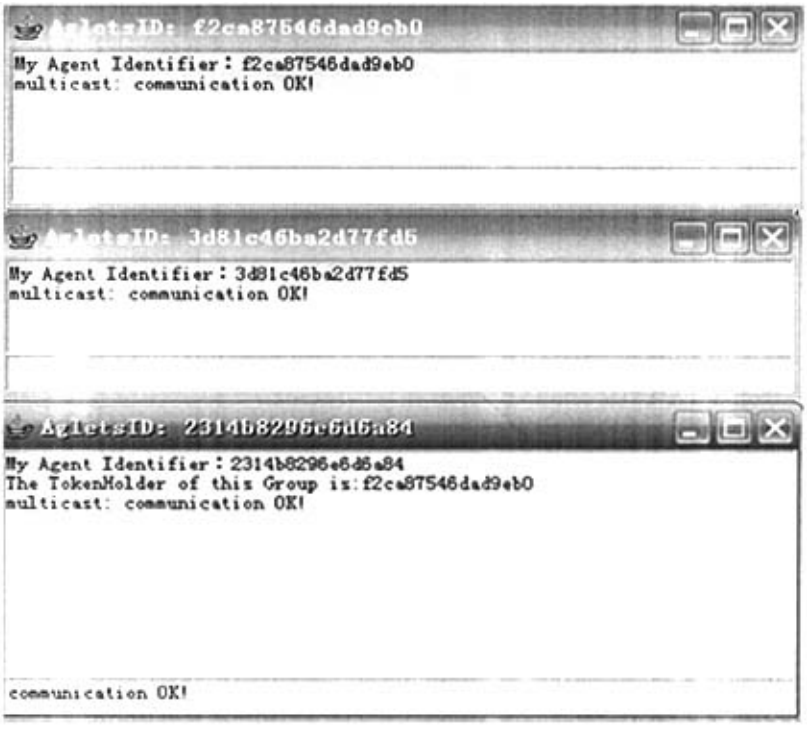


图 5-20（b）Windos 环境下组内可靠通信示意图

## 5.3 性能分析

### 5.3.1 通信速度

通信速度也是一个重要的衡量指标，它直接关系到通信系统设计目标的实现与否以及用户对于使用本系统的满意程度。显然，通信系统的设计目标总是希望尽可能地缩短消息传递的链路长度、避免不必要的中间节点、减小网络传输延时，从而能以最快的速度实现通信，这在对通信的实时性要求很高的应用中显得尤为重要。

CISOM通信模型将寻址和消息传递分离开来，在寻址方面，我们一般选用一个高性能的主机作为DNR服务器，因此高效性和可靠性得到了保证。而消息传递建立在寻址的基础上，在移动组内，消息通过多播的方式群发，在移动组间，消息仅由通信发起方向目的方的MailBox传递一次，避免了全局寻址法的消息的多次转发，因此效率要高。而我们划分域的原则是保证域内主机的高速网络连接，域的构造一般是基于局域网的，因此域内的通信速度也是比较快的。

### 5.3.2 稳定性

对于分布式异构系统而言，稳定性是指系统的性能不应随着软硬件环境和系统运行负载的变化发生剧烈波动，而是可以维持一个相对稳定的水平。

本章所提出的移动Agent组内组间消息传递机制实现了一种真正意义上的分布式通信模型，整个系统中不存在任何以集中方式进行控制的节点，系统的运行负荷也不是集中在少数通信服务器上，而是尽可能均匀地分布到了所有参与通信的主机之上，仅是由域内的DNR提供查询和通知发送包事件的服务，从而尽量减轻了域名字服务器的负担。而且，即便DNR或主机由于各种原因发生故障，也只会产生局部的影响，而不会危及到整个分布式系统的正常运行，因而整个系统的稳定性较强。

## 5.4 小结

本章首先介绍了实验环境和移动 Agent 平台 Aglets，然后介绍了对于移动组的组内和组间通信的实现－CISOM，以解决移动 Agent 的命名，Agent 寻址的位置透明性，移动组的消息处理，移动 Agent 组通信协调问题。最后对这两个方法进行了性能分析。

# 第六章 总结和展望

## 6.1 本文总结

随着计算机的软硬件技术与通信技术得到迅猛的发展，以及分布式人工智能技术与移动计算技术的发展，为了克服网络低带宽、高延时对网络应用普及的限制，移动 Agent 技术得到发展，并成为新的研究热点。在移动 Agent 的通信方面已经作了很多的研究工作，提出了很多比较好的通信方法，但是这些方法大多数只适用于一对一的通信方式，而对于移动 Agent 的组通信方面的研究还很少，存在很多不足。因此，本文对基于移动组的移动 Agent 的组通信进行了研究，给出了一个移动 Agent 的组通信系统 CISOM。主要的工作可以归纳为以下三点：

- 本文为了解决内存消耗资源过多的问题，采用 MailBox 的组内可靠多播通信技术，来减少内存的消耗。
- 本文提出了 SendBox 组间通信模型。成功消除了消息追踪，同时减小了网络负载的问题。

在 IBM 的 Aglets 平台上设计了 CISOM 系统，并对其性能进行了分析。

## 6.2 未来展望

本文所提出的移动Agent组内组间通信方法具有效率较高、通信坚定性较强等优点，虽然，但是也应看到其中还存在以下的不足：

组内组间通信算法来源是基于对对象访问的局部性原则，因此我们以域来划分移动Agent环境，域内保证高速和可靠的网络连接。在这种模式下，如果移动Agent在域内的迁移和通信概率要远大于跨域的概率，该模型具有较小的时间开销和较高的性能。但如果移动Agent在域间迁移和通信比较频繁，组间通信的时间开销要比一般算法大。所以在实际应用中应对具体情况进行分析，以改进组间通信的的算法。

# 致谢

攻读硕士是一个既艰苦又令人激动的过程。在这期间，许多人在很多方面给我提供了大量的指导和帮助。因此，在论文结束之际，我非常诚挚地、也很荣幸地向他们表达由衷的感激之情!

首先要感谢我的导师史小宏副教授和冯嘉礼教授，在论文的写作过程中，对论文提出了许多指导性的意见。在此，向史小宏老师和冯嘉礼老师致以最衷心的感谢。

感谢上海海事大学刘扬，陶冬霞和于杰，在论文的整理阶段和移动 Agent 软件制作的过程中给予了细心的指导。

# 参考文献


[1] 刘斌，王兰邵，王浩军.基于CORBA的分布式Agent通信构架[J] 小型微型计算机系统，2001，22(6):728-731.

[2] J . Baumann, F. Hohl, N. Radouniklis et al. Communication concepts for mobile Agent systems[C], In the Proceedings of the First International Workshop on Mobile Agents(MA' 97) , Berlin, Germany, April 1997. Rothermel Kurt, Popescu-Zeletin Radu(Eds.)Lecture Note: Science 1219, Springer Verlag, Berlin. 123-135.

[3] 陶先平，冯新宇，李新等.Mogent系统的通信机制[J]。软件学报。2000, 11(8)：1060-1065

[4] R. Gray. Agent TCL: A flexible and secure Mobile-Agent system[C]. In the Proceedings of the Fourth Annual Tcl/Tk Workshop(Tcl' 96). MarkDiekhan and Mark Roseman(editors). Monterey, California, USA. 1996.

[5] R. Gray. Agent Tcl: A flexible and secure system[D]: [PH. D. Thesis]. Department of Computer Science, Dartmouth College, 1997.

[6] R. Gray, D. Kots, S. Nog Computer Science, Dartmouth al. Mobile agents: The next Mobile-Agent College, 1997. generation In computing[C]. In the Proceedings of the Second Aixu International Symposium Algorithyms/Architectures Synthesis(PAS' 97). Fukushirna Japan. March 1997.

[7] G Karioth, D. Lange, M. Oshima. A security model for Aglnts[J]. IEEE Internet Computing, 1997, 1(4):68-77

[8] 王萍，王国仁等.移动Agent迁移机制的研究与实现[J].东北大学学报，2001,22(6):600-603

[9] 杨晟，魏海平，吴兴.基于Java语言的Mobile Agent实施方案[J].抚顺石油学院学报，2002,22(1):61-64.

[10] 吕玉海，徐学洲.移动agent技术的发展[J].西安电子科技大学学报(自然科学版)，2002,29(3):392-397.

[11] Gianpaolo Cugola, Analyzing mobile code languages [J], Computer Science、Springer-Verlag, 1996:93-110

[12] 陈剑芸，吴玲达，张茂军.分布式虚拟环境[J].计算机工程与应用，2002,07:119-122

[13] 唐浩坤,移动Agent可靠位置透明通信方法的研究[D]:[硕士学位论文].四川:西南师范大学计算机系，2003.

[14] 田敬军.荃于移动Agent的信息检索系统的研究与实现（D]:（硕士学位论文）.西安:西北大学计算机学院，2002.

[15] Robert S, G. Agent TCL A flexible and secure mobile-agent system[D]:[Ph. D Thesis]. Computer Science of Dartmouth College, June, 1997

[16] 李淑琴，王 诚，蔡月茹.Internet上的移动agent技术[J].计算机工程与设计，2001,22(3):16-19.

[17] 王昭光，韩京才，王洪锋，Java语言与软件agent[J]计算机应用,2001,21(8):77-79.

[18] 吕玉海，徐 学洲.移动agent技术的发展[J]，西安电子科技大学学报(自然科学版)，2002,29(3):392-397

[19] 王红，曾广周，林守勋.可移动agent系统位置透明通信的一种实现[J].计算机学报，

2001,24(4): 442-446

[20] 薛丰华，韩露，周曼丽. 移动代理者安全体系的研究[J]，计算机工程，2001,27(5):134-136

[21] Lange D B, Mitsuru Oshima. Seven good reasons for mobile agents. Communications of the ACM, 1999, 42(3): 88～89.

[22] Miaoliang Zhu, Yu Qiu. A survey of mobile agent system. Journal of computer research and development, 2001, 38(1).

[23] Kozt, R.S.Gray. Mobile Agents and the Future of the Internet. ACM Operating Systems Review, 1999, 33(3): 7～13.

[24] JN Cao, XY Feng, J Lu, et al. Design of Adaptive and Reliable Mobile Agent Communication Protocols. In Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS 2002), IEEE Computer Society, 2002: 471～427.

[25] MP Flavio, A Silva, RJA Mace. Reliability Requirements in Mobile Agent Systems. In Proceedings of the II Worshop on Tests and Fault Tolerance, Curitiba, Brazil, 2000.

[26] Rolf. Security Issues Related to Mobile code and Agent～based systems. Computer Communications, 1999 22(12): 1165～1170.

[27] Aridor. Infrastructure for Mobile Agents: Requirements and Design. In Proceedings of the Second International Workshop on Mobile Agents (MA'98), Stuttgart, Germany, September, 1998, 38～49.

[28] M. Straber, K. Rothermel. Reliability Concepts for Mobile Agents. International Journal of Cooperative Information Systems (IJCIS), 1998, 7(4): 355～382.

[29] D.E. White. A Comparison of Mobile Agent Migration Mechanisms. Senior Honors Thesis, Dartmouth College, June, 1998.

[30] DY Liu, B Yang, K Yang, SS Wang. Intenarary Graph-based Migration Strategy of Mobile Agents. Chinese Journal of Computer Research and Development, 2003, 40(6): 838～845.

[31] K.P. Birman, R. van Renesse. Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society, 1994.

[32] A. Tannenbaum, R. van Renesse, H. van Staveren, G. Sharp, S. Mullender, J. Jansen, G. van Rossum. Experiences with the Amoeba distributed operation system. Communications of the ACM, 1990, 33(12): 46～63.

[33] S. Mishra, L. Peterson, R. Schlichting. Consul: A communication substrate for fault～tolerant distributed programs. Distributed Systems Engineering, 1993, 1(2): 87～103.

[34] P. Ezhilchelvan, R. Macedo, S. Shrivastava. Newtop: A Fault-Tolerant Group Communication Protocol. In Proceedings of the 15th IEEE International Conference on Distributed Computing Systems (ICDCS'95), IEEE Computer Society, 1995: 296～306.

[35] D. Dolev, D. Malki. The Transis Approach to High Availability Cluster Communications. Communications of the ACM, 1996, 39(4): 64～70.

[36] R. van Renesse, K.P. Birman, S. Maffeis. Horus: A Flexible Group Communication System. Communications of the ACM, 1996, 39(4): 76～83.

[37] C. Malloth, P. Felber, A. Schiper, U. Wilhelm. Phoenix: A Toolkit for Building Fault-Tolerant Distributed Application in Large Scale. Technical report, Department d'Informatique, Ecole Polytechnique Federale de Lausanne, July 1995.

[38] Y. Amir, L.E. Moser, P.M. Melliar-Smith, DA Agarwal, AP Ciarfella. The Totem Single-Ring Ordering and Membership Protocol. ACM Transactions on Computer Systems, 1995, 13(4): 311～342.

[39] D.A. Agarwal, L.E. Moser, P.M. Melliar-Smith, R.K. Budhia. The totem multiple-ring ordering and topology maintenance protocol. ACM Transactions on Computer Systems, 1995, 13(4): 311～342.

[40] S. Jhonson, F. Jahanian, J. Shah. The inter-group router approach to scalable group composition. In Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99), IEEE Computer Society, 1999: 4～14.

[41] A.V. Bakre, B. Badrinath. Implementation and Performance Evaluation of Indirect TCP. IEEE Transactions on Computers, 1997, 46(2): 353～361.

[42] 邓卫移动Agent技术及其在移动通信系统中的应用[D]:[博士学位论文]北京:北京邮电大学计算机学院，2001.

[43] 万燕，朱向华,孙永强.基于移动代理系统的名字服务定位算法[J].计算机工程,2001,27 (11):11-13.

[44] 贾忠伟，唐功友，焦润海，郭山清.一种基于Agent位里透明及消息级冲体的消息传递机制[J] 中国海洋大学学报，2004,34 (1):115-120.

[45] 吴刚，王怀民，吴泉源.一个移动智能体位里管理与可靠通信的算法[J]. 软件学报，2002,13:269-274.

[46] 乔树清，王巍一种容错的移动Agent名字解析机制的研究[J].计算机应用，2004,24 (6):84—87

[47]ObjectSpace.VoyagerCoreTechnology[DB/OL].http://www.objectspace.com/voyager,2000.

[48] Y. Amir, L.E. Moser, P.M. Melliar-Smith, DA Agarwal, AP Ciarfella. The Totem Single-Ring Ordering and Membership Protocol[J]. ACM Transactions on Computer Systems, 1995, 13(4): 311～342.

[49] B. Whetten, T. Montgomery, A. Kaplan. A High Performance Totally Ordered Multicast Protocol[J]. Lecture Notes in Computer Science Vol.938, Springer, 1994: 33～57.

[50] Internet Engineering Task Force (IETF)," RFC3261 (SIP: Session Initiation Protocol)," 1999.http://www.cs.columbia.edu/sip/drafts.html

[51] D.Lange, M.Oshima. Programming and deploying Java mobile agents with Aglets. Addison Welsey, 1998.

[52] B.Lange, O.Mitsuru. Programming Mobile Agents in Java-With the Java Aglet API. IBM Research, 1997.

[53] Mitsuru Oshima, Guenter Karjoth. Aglets specification 1.1 draft. http://www.trl.ibm.com/ aglets/spec11.htm, 1998.

[54] Danny B. Lange, Yariv Aridor. Agent Transfer Protocol—ATP/0.1.

# 附录 发表论文及参加的科研项目

一、参加的科研项目

上海市教育委员会科研项目“移动 Agent 系统的移动组通信机制”(06FZ021)

二、发表论文情况

1、周晨叶. 基于移动组的全序可靠多播通信. 舰船电子工程，2007 年 06 期. 刊号：CN 42-1427/U (已录用)