摘　要

随着网络技术的发展，嵌入式技术和网络技术的结合是大势所趋。然而，就目前而言，嵌入式设备大多采用无线网络接入的方式，无线网络具有低带宽、高延时、易断线等缺点，无法适应信息化社会的大量信息交换和共享的要求。移动代理技术的优势则决定了它能够帮助嵌入式设备更好地融入 Internet，因此嵌入式移动代理应运而生。

将移动代理技术应用于嵌入式设备首先需要有一个嵌入式移动代理平台，目前尚未有成熟的嵌入式移动代理平台可以直接使用，因此我们采用了自己研发的 μAglet 作为实验平台，它是以移植了 KVM 的 μClinux 为软件平台，以 IBM 的 Aglet 为改造蓝本设计并实现出来的嵌入式移动代理平台。

移动代理的安全问题一直是制约移动代理技术实现普遍应用的一个瓶颈。移动代理在传输中的安全、移动代理平台的安全以及移动代理本身的安全是移动代理安全策略要解决的三大问题。网络的开放性和代理的移动性，使得移动代理系统的安全问题与别的系统相比显得更加复杂。而无线网络的缺点以及嵌入式设备的有限性，则决定了嵌入式移动代理的安全问题比移动代理的安全问题更棘手，因此嵌入式移动代理系统的安全问题是竖在我们面前的一道高墙。我们就这个问题尝试提出一种嵌入式移动代理系统的安全问题解决策略。

论文介绍了移动代理的概念、特点以及发展现状，接着详细剖析移动代理面临的安全问题以及现有的解决方案。论文还介绍了本实验室开发的嵌入式移动代理平台 μAglet，详细阐述和分析嵌入式设备上移动代理系统的安全策略，并将该策略应用到实际项目中。

关键词：嵌入式；移动代理；安全；无线网络；μAglet

Research of Embedded Mobile Agent System Security Mechanism

Abstract

With the constant development of the network technology, embedded devices will be integrated into Internet. Currently, the embedded devices using a wireless connection way has disadvantages such as low bandwidth, instability and so forth.Therefore, It can't meet the needs of a large amount of information exchange and share in the information-based society.The advantages of mobile agent technology determines that it can be integrated embedded equipment into Internet better. So the embedded mobile agent emerged.

The precondition of using mobile agent technology in embedded equipment is to have an embedded mobile agent platform. However, there is not yet a mature platform can be used for embedded mobile agent directly. Hence, we adopted μAglet which developed by our own laboratory as the experimental platform. It chooses μCLinux with migrated KVM as software platform, and reconstructs IBM's Aglet system to complete the whole system.

Security of mobile agent has become the bottleneck that restricts mobile agent technology to achieve universal application. Security of mobile agent transimission, security of mobile agent platform and security of mobile agent is three problem of security of mobile agnet. Openning of network and migration of mobile agent make the mobile agent system security more complicated than other systems. Furthermore, because of limit of embedded device and the disadvantage of wireless network, the security of wireless network is more serious than the mobile agent. Embedded mobile agent application security problem likes a wall which was erected in front of us. This issue tries to make a solution of embedded mobile agent application security.

This paper introduces the concepts and the feature of mobile agent. And then we make a detailed analysis of the security problems and the existing solutions faced by mobile agents. The paper also introduces the embedded mobile agent platform μAglet developed in the laboratory. There is the detailed description and analysis of security strategy of mobile agent application in embedded equipment. Finally, the strategy was applied to a practical project.

Key Words: Embedded; Mobile Agent; Security; Wireless Network; μAglet

独创性说明

作者郑重声明：本硕士学位论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得大连理工大学或者其他单位的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

作者签名：[signature] 日期：2006.12.18

大连理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“大连理工大学硕士、博士学位论文版权使用规定”，同意大连理工大学保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权大连理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：张雷

导师签名：郭平

2006 年 12 月 18 日

引　言

随着计算机在全球范围内的普及，计算机技术与人们的日常生活和工作越来越密不可分。然而，个人电脑在便携性上有时还不能满足需要，在一些特殊环境下也无法发挥作用。随着嵌入式技术的飞速发展，嵌入式系统在工业控制、军事、日常生活等领域应用愈加广泛，掌上电脑与多功能手机也逐渐普及，成为人们办公、娱乐的重要载体。

当今的信息化社会中，以 Internet 为代表的计算机网络技术和通信技术已经渗透到各个领域，嵌入式技术势必要和网络技术相结合。然而，将嵌入式设备完全融入到 Internet，需要解决以下两大问题：

(1) 异构问题

现在的 Internet 存在一系列的异构问题，包括平台异构，操作系统异构，计算机语言异构，协议异构，版本异构。对于嵌入式设备，异构问题比较突出的是平台异构，操作系统异构和计算机语言异构。现阶段嵌入式应用程序通常为了追求效率，往往采用 C 语言和汇编语言，可移植性差，不利于软件重用。

(2) 网络问题

嵌入式设备通常具有微型化、可移动的特点，这样就决定了其接入 Internet 的方式为无线接入(wireless connection)。无线接入具有低带宽、易断线等缺点，这远远不能满足大量信息传输和共享的需求。这样，不可避免地带来了一些问题：

① 网络的低带宽、高延时逐渐成为一个限制网络应用和发展的瓶颈；

② 嵌入式设备资源受限、计算能力弱，难以维持连续的网络连接；

③ Internet 上剧增的数据量和用户的个性化需求都需要有更好的信息检索手段支持。

20 世纪 90 年代，人们提出了移动代理技术。移动代理是一个代替人或者其他程序执行某种任务的程序，它在复杂的网络环境中能够自主地从一台主机移动到另一台主机。该程序能够选择何时、何地移动。在移动时，该程序可以根据要求挂起其运行，然后转移到网络的其它地方重新开始或继续其执行，最后返回结果和消息。

移动代理所能带来的好处有：

(1) 减少网络负载，节省带宽。移动代理允许用户把交互一次打包，发送到目标主机，使得交互可以在本地运行。

(2) 克服网络延迟。代理可以从客户端发送到事件发生地，直接进行本地操作。

(3) 支持断接。移动环境下，要使移动节点之间或者移动节点和固定节点之间保持连续不断的连接，从技术上和经济上都不太可行。要支持断接，可以装入移动代理，利

用有限的网络连接传递代理，之后代理的执行就可以独立于发送节点。

(4) 灵活性和可适应性。移动代理封装代码和状态，可以感知环境，对事件作出反应，从而适用不断变化着的移动环境。

移动代理技术的特点决定着它很适合于解决构建在嵌入式设备上网所面临的上述问题，有利于用户的个性化、智能化、自主化需求。因此在嵌入式设备中使用移动代理技术是发展的必然并且会迅速成为现实。

嵌入式设备要想生成或者执行移动代理，首要条件就是在嵌入式设备中要有一个移动代理平台，目前已有的移动代理平台都是基于 PC 机，不能在嵌入式设备上直接使用，因此，嵌入式移动代理平台的开发也是必要工作。本论文中的嵌入式移动代理平台将采用本实验室自行研发的 μAglet。

移动代理的关键技术包括移动机制、通信机制以及安全机制。安全性是制约移动代理技术广泛使用的重要因素之一，因此研究移动代理的安全问题具有重要意义。

本课题主要讨论嵌入式设备上的移动代理系统的安全问题及解决办法，属于嵌入式研究领域和分布式计算研究领域。

嵌入式设备(这里指受限设备)的典型特征就是只拥有受限的处理器速度和存储器。尽管它们的能力在不断提高，但目前它们还不能与服务器甚至桌面系统相提并论。我们主要描述受限设备的一般特点，包括处理器速度有限、存储器和存储设备受限、联网能力和带宽受限。虽然可能这些设备运行的大多数是小型的移动代理应用，但其安全保护措施大多是必须的，特别的由于嵌入式设备的联网往往采用无线的方式这就带来了更大的基于通信的安全问题。

移动代理技术被广泛地认为是一种新兴的分布计算机制，它为受网络带宽限制比较大的分布计算提供了一种很好的解决方法。但由于 MA(mobile agent)是在主机上运行，理论上主机可以控制 MA 的代码，所以有人认为如果主机有恶意，那么它对 MA 的攻击是不可避免的[1]。为此，很多学者针对移动代理的安全问题提出了解决方案，有基于检测入侵的[2]，也有基于防止入侵的[1]。对这个问题的解决方案各种各样，但都无法完全解决这个问题，它们既有自己的优势也有自身的缺陷。本课题将尝试改进前人的工作，综合出一种策略来解决这个问题。

移动代理的安全问题主要包括[3,4]：

(1) 移动代理在传输过程中受到攻击的问题；

(2) 主机或者移动代理环境受到恶意移动代理的攻击；

(3) 移动代理受恶意主机或者移动代理环境的攻击。

我们可以认为适合嵌入式移动代理的安全策略是对普遍对策的一个精简，主要是考虑到这些嵌入式设备的特点而定的，其基本指导思想是尽量少用复杂的计算量较大的算法，协议交互尽量简单，尽量考虑采用多一些策略性质的措施。

移动代理技术的发展和网络的分布化要求相吻合，将其应用于嵌入式系统是未来网络发展的必然趋势，而解决嵌入式移动代理的安全问题则是嵌入式系统完全融入网络技术的必经之路。在现有的硬件、软件技术条件下和现有的理论基础支持下，本课题内容是完全可行的，并且具有一定的技术理论价值和实际应用价值。进行嵌入式领域的移动代理安全机制的研究，既是大势所趋，又有实际意义。

1 移动代理简介

1.1 移动代理技术的相关概念和标准

代理(Agent)一般而言指的是一个具有自主性、社会能力和反映特征的计算机软/硬件系统，广义上是指具有智能的任何实体，包括人类、智能硬件(如机器人)和智能软件。FIPA(Foundation for Intelligent Physical Agent)对Agent的定义为："Agent是驻留于环境中的数据，并执行对环境产生影响的行为"。其中，软件Agent研究者的定义如下(非FIPA定义)："智能软件Agent是能为用户执行特定的任务、具有一定程度的智能以允许自主执行部分任务并以一种合适的方式与环境相互作用的软件程序"。

随着网络技术和的发展和分布式计算的需要，进一步产生了移动代理(Mobile Agent)的思想。

移动代理是一个能在异构网络中自主地从一台主机迁移到另一台主机，并可与其他代理或资源交互的程序[5,6]。移动代理迁移的内容既包括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代理当前运行时状态；数据状态主要指与移动代理运行有关的数据堆的内容。移动代理具有跨平台持续运行、自我控制移动的能力，能够模拟人类行为关系，并提供一定人工智能的服务程序。它继承了某些代理的特性。一个移动代理必须包含以下所有的模型：代理模型、生命周期模型、可计算模型、安全模型、通信模型、和导航模型[7]。

在以上的定义中，移动代理存在的软件环境被称为移动代理环境(MAE)。移动代理环境是分布在一个由异构计算机组成的网络中的软件系统。它的主要任务是提供移动代理可以执行的环境。移动代理环境实现了大部分移动代理定义的模型。移动代理系统的组成如图 1.1 所示。

为了加快移动代理技术的发展和推动移动代理的具体应用，与移动代理技术研究相关的各机构和企业形成了一个标准化组织，旨在规范移动代理技术以实现平台之间的互操作性。标准化工作中最有影响力的国际组织是OMG下属的Agent Working Group和FIPA。

现今，移动代理技术主要有两大标准：由对象管理组织于 1998 年发布的 MASIF(The Object Management Group's Mobile Agent System Interoperability Facility)标准和由智能物理 Agent 协会发布的 FIPA(Foundation for Intelligent Physical Agents)标准。

MASIF 标准首先规定了通用概念模型，该模型包含一个分布式代理环境 DAE(Distributed Agent Environment)和一个分布式处理环境 DPE(Distributed Processing

Environment)，并涵盖了现有移动代理平台的所有主要要素，如 Agent 状态、Agent 授权者、Agent 名字、位置(Place)、域(Region)、代码库(Codebase)和通信基础等概念；其次说明了各要素之间的关系；再次介绍了移动代理与 CORBA 的结合；最后，用 IDL 定义了两个接口：MAFFinder 和 MAFAgentSystem[8]。

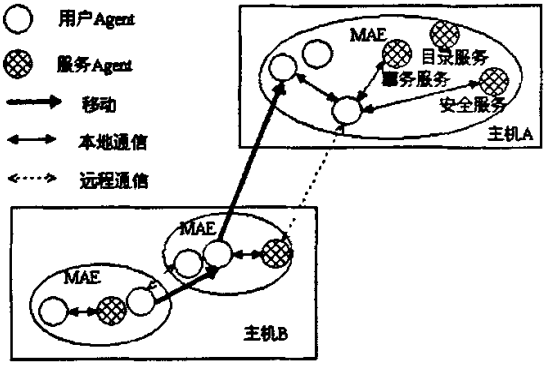

图 1.1 移动代理系统的组成

Fig. 1.1 Components of the mobile agent system

移动代理平台是建立在操作系统之上的虚拟机环境，并实现 Agent 的创建、执行、移动、终止等基本功能。每台计算机上可以有一个或多个移动代理平台，Agent 运行于这些平台的不同位置(Place)，Agent 之间的通信和移动都是通过通信基础来实现的，如图 1.2 移动代理系统所示。

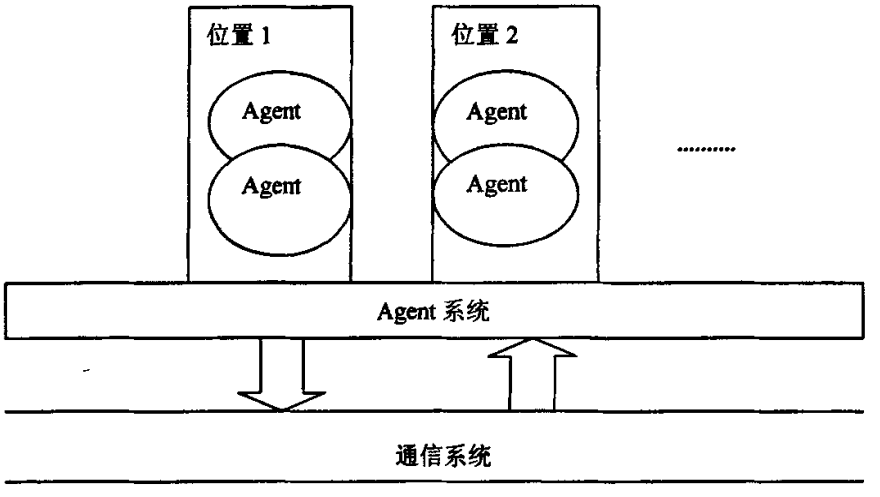

图 1.2 移动代理系统

Fig. 1.2 Mobile agent system

MAFFinder 接口通过维护一个动态的名字和地址映射关系，实现了 Agent 系统的注册、注销、定位、跟踪等操作。通常情况下，在多域系统中每个域都有单独的 MAFFinder，如图 1.3 所示。

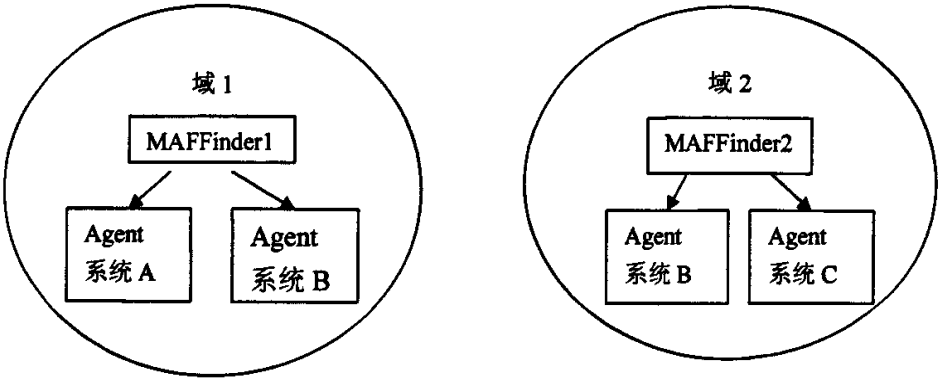

图 1.3 多域系统

Fig. 1.3 Multi-domain system

MAFAgentSystem 定义了对 Agent 的操作，如：创建、暂停、接收、恢复等。它详细定义了方法名、参数类型、含义、数量、返回值等，这些方法提供了 Agent 生命周期的基本服务。

MASIF 标准的优势在于保证了每个移动代理平台的自身特性，并且充分利用了 CORBA，使其与现有的 CORBA 系统能够较好的融合。

FIPA 标准的目的是使代理和代理系统的交互简易化[9]。作为 Agent 管理系统的附加，FIPA 涵盖了代理通信和代理消息传递，这些在 MASIF 标准中并未提及。FIPA 提出了代理通信语言 ACL(Agent Communication Language)。ACL 起源于知识查询和操纵语言 KQML(Knowledge Query and Manipulation Language)，基于语言行为理论(Speech Act Theory)。这是一种具有精确表达式和语义的高级语言，通过它使代理之间进行通信。一个代理消息包括两部分，信封和消息体。信封传递了欲传输的信息，而消息体包含了真正的消息内容。此外 FIPA 还涉及了代理安全管理和移动管理。

通过这两个标准的比较，它们在结构上和功能上有类似之处，可大致总结如下：

(1) MASIF Region 和 FIPA Domain 都是属于经过授权的分布式和互协作的代理平台，通常它们被认为是 Agent 活动的工作安全域。

(2) MASIF 代理系统接口 MAFAgentSystem 和 FIPA 代理管理系统(AMS)部件提供平台中管理代理生命周期的机制，它们提供生成、挂起、恢复、中止和迁移 Agent 的管理操作。

(3) MASIF MAFFinder 接口和 FIPA 的目录设施(DF)部件为管理动态注册服务提供方法。

(4) MASIF MAFFinder 接口和 FIPA 代理管理系统(AMS)部件定义命名和定位目录。

但是，它们也有各自的侧重点，OMG MASIF 侧重域具有相同特征的代理系统之间由 CORBA IDL 接口的代理移动性，而并不关心内部代理的通信。另一方面，FIPA 规范侧重于由内容语言(Content language)的智能代理通信，而不关心代理的移动性。不过，目前 OMG 和 FIPA 已经成立了联络机构(OMG-FIPA)联络处，以协调两个组织关于 Agent 技术的工作。

1.2 移动代理技术的特点和优势

移动代理技术作为一种极具潜力的技术，吸引着越来越多的研究者，其得以存在的依据主要是其“代码和数据迁移”的传输机制适应了现有网络技术的发展和网络应用的需求，主要表现在：

(1) 在需要大数据量传输和实时控制中的应用

分布式系统通常依赖不同的协议以沟通协调完成工作，这无形中增加了网络的负担。移动代理将计算代码迁移到数据端，而不是将数据传送到客户端计算，这样不但可减少网络负担，而且移动代理就近执行可减少网络的延迟。因此，在完成数据传输量较大的任务如网络管理及实时控制等应用上，移动代理是相当合适的。一般来说，只要传输代码在带宽和时间上的耗费小于传输数据的耗费时，使用移动代理就有意义。

(2) 可对网络协议进行封装

分布式系统的协议往往是程序更新时的包袱。因为效率或安全问题而需要重新设计协议时，新旧协议间的相容性是必须考虑的问题，而移动代理将client-server间的协议封装起来，不会再出现服务器端的协议改变后，客户端也需要更新的情况。

(3) 自主性适用于网络终端设备的动态改变

只要将移动代理放到网络的主机上，移动代理就可自主执行，不需使用者一直连接在网络上。这对使用经常离线的嵌入式设备如PDAs和laptops等的客户特别实用，能够把任务委托给移动代理而移动终端不用长时间与网络连接。同时移动代理对于网络设备配置的动态改变反应迅速，一般只需增派或撤消移动代理。

(4) 动态迁移的特性提供极大的灵活性

移动代理会根据既定的路线和执行状况动态迁移，调整到最佳的执行状态，例如：选择最靠近数据的主机执行或者选择计算能力最强的主机执行，这也是移动代理不同于其它分布式技术的特色之处。

(5) 对网络和设备的鲁棒性好

当移动代理发现目前的执行环境不再合适时，可动态迁移到其它主机，从而增加了容错能力。例如：所在主机准备关机时，主机会通知在它上面执行的移动代理，移动代理则尽快迁移。此外移动代理对网络连接的质量要求不高，它只有在移动的过程中才会受到网络的影响，而在与其他主机或代理交互的过程中不受网络状态的影响，因此它比传统的面向连接的通信方式更为严重可靠，对底层网络的质量要求更低，这种特性使其适用于可靠性和带宽都较低的无线网络，可应用在通信质量并不高的无线网络中。

(6) 满足网络使用者个性化需求的增加

由于网络上的通信量激增，同时使用者数目及使用者之间的差异也快速增加，而且使用者不再满足于单一信息存取，希望有个性化的信息表示及存取方式。现在一般由拥有信息的网络或Proxy提供特定的工具来完成对个性化服务的支持，同时客户端的工作开始整合到主机或Proxy，让Proxy根据使用者的需求来收集、预先下载、获取信息。这种方式存在的缺点主要有：无法提供全面的个性化；网站往往各自提供个性化工具，使用者在不同的网站不能使用相同的个人设定和存取界面；当使用者迁移到距Proxy较远的地点，固定位置、特定目地的Proxy往往成为瓶颈。而移动代理则为个性化的服务提供了合适的选择，主机或Proxy只要提供适合的执行环境，使用者即可以自行设计个人的移动代理，将其放到合适的主机上后，使用者可自由断线，移动代理会根据实际状况迁移到更合适的机上，在恢复连接后，再向使用者报告工作进度。

1.3 移动代理技术面临的关键问题

虽然移动代理在很多方面都显示出其极大优势，甚至预言它将在下一代网络分布式应用中占据主导地位，但其反对者们也从不同的侧面对这一新的不完善的技术提出质疑，如安全性问题、标准化问题等，因此应对移动代理技术的发展前景有一个客观科学的分析。其中制约移动代理广泛应用的几个问题，也即移动代理的研究者们致力于解决的问题主要有：

(1) 安全性有待提高

一个代理平台必须能够接受其他的移动代理实体，如果进行诸如电子商务等活动，必须要提供足够的安全保证。移动代理系统的安全问题是制约其真正走向实用化的瓶颈，有很多研究在讨论这方面的问题。但拒绝服务攻击、洪泛攻击、代理的完整性问题、

由签名问题引发的匿名性问题等难题还没有得到很好的解决。

(2) 执行效率有待提高

在通常的分布式应用中，采用现有的移动代理系统并不能带来效率上的好处，只有在高延迟、低带宽、连接不可靠的移动计算或者中间数据传输量非常大的情况下，移动代理技术才表现出效率上的优势。为了保证平台无关性和安全性，绝大多数移动代理系统都采用解释型语言，与编译型语言相比，要慢很多。另外，代理的移动也非常费时。

(3) 当前移动代理系统的容错能力不高。

(4) 自治性差

移动代理系统的许多优点是建立在代理自治基础上的，因此需要增强其智能性。

(5) 移动代理的标准化工作还有待进一步完善和推广。

1.4 现有移动代理平台的对比

移动代理系统(MAS)的研究异常活跃。迄今为止，已有约60多种移动代理系统存在，绝大多数系统都是在Java语言之后出现的，大部分系统都还只是原形系统。它们的针对性不同，实现方法也各有千秋。下面我们将讨论当前比较成熟而具有代表性的一些移动代理系统[10]。

(1) Ara

Ara是移动代理的运行平台，它致力于把移动编程和当前的编程实践无缝连接，即尽可能用现有的编程模型或语言来实现程序移动。为此，Ara系统提供了一个核心，它直接运行在未经任何修改的操作系统上。核心之上是一个多语言的解释器。Ara移动代理是一个能根据自己的选择移动的程序，其执行不需要外部的干涉。代理可以用Tcl、C/C++、Java等语一言编写，运行在解释器内。C语言传统上是编译型语言，Ara专门设计了C语言编译器，把C程序预编译成可移动的MACE解释字节码，然后在MACE解释器中运行。Ara的核心只提供一些最基本的服务，如资源管理、移动性、安全性等，一些高层服务则由专门的服务代理提供。服务代理是静态编译的对象，分布在网络接点上。

在任何状态下，Ara代理的移动只需调用一个Go指令。代理在创建时，被赋予全局Allowance，它用来限制Agent的行为。Agent移动后必须进入叫Place的虚拟位置。一个Place如果接受Agent，它将赋予Agent一个本地的Allowance，它是一个系统资源访问权限向量，进一步对Agent行为进行限制。在同一Place中，Agent间的通信采用一种类似于Client/Server方式的消息交换。核心为此提供服务节点的概念，一个Agent可以用广播的方式公布服务点，扮演服务Agent的角色，其它的Agent则以客户Agent的身份和它交互。Ara中对传送的代理进行加密，用数字签名来验证Agent的用户或发送主机。和其它多语

言系统不同的是整个Ara系统运行时是一个多线程的进程，迁移到系统内的Agent也以线程的方式运行。这种方式有利于提高系统效率。

(2) Telescript

Telescript由General Magic公司开发，是第1个商品化的MAS，首次应用于AT&T公司PersonalLink网络中。它是一个与平台无关的系统，包括面向对象的移动代理语言及解释器。在Telescript中，每个网络节点运行一个服务器，管理着一个或多个虚拟的Place，也即代理运行环境。Agent用Go命令来使自己移动，在进入Place前，Agent要向Place提供加密证书，并被分配一定的权限。Agent用meet指令和同一Place中的Agent交互，用connect指令和不同Place中的Agent进行会话。Telescript还提供基于设备的验证手段和复杂的授权策略。从技术角度讲，Telescript已经是一个非常完整的移动代理系统，它的缺点是消耗计算机资源太多(要求96M的内存资源)，售价也高。随着基于Java的MAS出现，它已没有了生命力。在Telescript之后，General Magic公司开发了纯Jave版的移动代理系统—Odyssey。Odyssey扩展了Java的RMI，提供了对Lorba的Hop和Microsoft的DCOM支持，是基于Java的移动代理系统中的唯一支持各种传输机制的系统。但有关它的文献太少，不便于对它进行更多的描述。从已有的文献看，Odyssey继承了许多Telescript中的概念。

(3) D'Agent

D'Agent是由Dartmouth大学开发的移动代理系统，以前称为AgentTcle。D'Agent有类似Telescript的服务器模型，它和Ara一样支持多语言，目前支持Tcl、Java和Scheme。它的体系结构可以分5层：最低层是实现传送机制的API；第2层是运行在各个网络节点的服务器，负责Agent的管理和状态俘获、移动、通信和Agent的备份等功能。和Tacoma一样，一些高层的服务如资源目录、网络检测、位置无关命名、高层通信、访问控制等由其它Agent提供。其中最重要是资源管理器Agent。体系结构的第3层是一个和语言无关的核心层，它在服务器的协同下，实现Agent的操作命令，如Agent在服务器中注册或注销(begin and end)、移动(jump)、发送或接收消息(send and receive)、查询其它Agent的状态(status)等。第4层是语言解释器，每一种支持语言各有一个解释器。每个解释器有四个组成部分：解释器本身、安全模块、状态模块及与核心的接口。安全模块的作用是确保Agent不违反资源管理器规定的访问限制。状态模块向核心提供俘获和恢复Agent内部状态的函数。在Ara中，Agent作为线程运行，由解释器调度时序和利用核心提供的函数来分配内存和访问系统。但在D'Agent中，每个Agent有自己的进程，进程的调度依赖于操作系统。每种语言实现自己的安全访问模块。这种方法的好处是对标准解释器的修改不多，只需增加状态俘获函数，因此易于随着标准解释器版本升级。但随着支持语言

的增多，解释器的工作量很大，不如Ara易于实现。D'Agent提供两个层次的通信机制，低层的通信包括消息队列、字节流。高层的通信如RMI，KQML则建立在Agent层次上，由通信服务Agent提供。Agent传送时要被加密以保护隐私。用RSA公开密钥密码方法来验证Agent的拥有者，由资源管理器决定Agent的访问权限。每种资源(CPU、文件系统、屏幕、网络等等)各有一个管理器。来访的Agent运行在一个不受信任的解释器中，所有的资源接触命令被隔离的可信任解释器捕获，该解释器再向相应的管理器咨询可否访问资源。

(4) Aglet

Aglet是最早开发的基于Java的移动代理系统，也是最受欢迎的移动代理系统。它的设计非常简洁，紧紧追寻Java模型。Aglet移动代理被视为一个移动的Java对象。一个Agent包含核心(core)、代理(Proxy)、路由计划(itinerary)和全球唯一的标识符(identifier)。核心拥有Agent所有的内部变量和方法，提供它与环境交互的接口。Proxy则封装了核心，防止对Agent私有方法的直接读取。在Aglet系统中，Context是Agent工作场所，它是一个静态对象，维护和管理Agent的运行并保护主机系统不受恶意Agent的攻击。Aglet对象支持的行为包括创建(creation)、克隆(cloning)、派遣(dispathing)、回收(retraction)、挂起(deactivation)、激活(activation)、终结(disposalof)和消息传递(messaging)。克隆将产生一个除标识符外和原Agent完全一样的对象。Aglet模型采用事件驱动方式。当一个Agent要移动时，它将调用Dispatch方法。Aglet系统通过代理传送协议(ATP)把Agent传送到目的地，每个Agent有全局独一无二的名字。Agent在迁移过程中并不去俘获自身线程的状态，因此它到达目的地后将从特定的入口点开始重新运行程序。Aglet的移动代理模型提供的是一个简单的框架结构，开发者可重写预定义的方法来增加所需的功能。

Aglet利用一个消息类密封代理间的消息交换，它提供同步或异步的消息传递。它采用白板支持多Agent间协作和信息共享。Aglet模型还提供一个简单的持久化工具，允许代理把自身代码和状态写入辅助的存储设备。Aglet安全模型建立在标准的Java安全机制之上，通过访问权限的限制来保护主机和代理免受恶意攻击，其规则和D'Agent、Ara等系统相若。

(5) Concordia

Concordia也是基于Java的移动代理系统。在每个网络节点上，Concordia都有一个运行在Java虚拟机上的服务器，它是一些称之为管理器的服务组件集合，负责Agent的移动、持久性、安全、通信等事项，是Agent执行任务的环境。在一个Concordia系统中，通常还单独存在一个叫Administrator的管理组件，能对系统内的多个服务器进行远程集中管理，如启动或关闭服务器内的某个组件，它还通过整个网络对Agent进行监视，保留Agent

和系统的统计数据。

Concordia非常注重系统的安全和容错性，提供了比较丰富的安全管理手段。它的安全模型支持3种保护：Agent存储保护、Agent传送保护和服务器资源保护。Agent存储保护采用加密的办法对Agent的备份加以保护。Agent传送保护利用数字证书来进行用户鉴别，传送时对Agent进行加密，通过对称密钥交换来验证Agent的发送者和接收者。资源保护则建立在对Java安全模型的扩展之上。Concordia对Agent的权限限制主要是根据使用者决定，这是它和Aglet、Odyssey等系统的一个区别。

Concordia的通信利用现有的TCP/IP通信服务。在Agent发送过程中，队列管理器(queuemanager)将把它保存在本地的消息队列中直到确认被目的主机接收。

(6) TACOMA

TACOMA(Trams and Comell Moving Agents)的第1个原形系统1994年就完成了，迄今已发布了4个版本。TACOMA关注于操作系统对移动代理的支持。TACOMA系统的移动代理可以用C、Tcl/Tk、Perl、Python、VB、Scheme等编写，能运行在Windows平台(Win9x、NT、WinCE)和大多数的Unix平台上，是已知系统中支持语言种类和平台最多的系统。TACOMA提出了公文包(briefcase)、文件夹(folder)、文件柜(filecabinet)等抽象概念作为OS的扩充。和其它移动代理系统不同的是，TACOMA没有提供状态俘获功能。当Agent要迁移到新的机器中去时，Agent创建一个folder，把代码和所需的状态信息放入folder中，然后该folde移往新的机器。到达目的地后，在一个特定的入口点重新运行Agent。不提供状态俘获功能的好处是可以快速集成新的语言到TACOMA系统中，因为几乎用不着改动现成的解释器或虚拟机。但给编程人员增添了麻烦，因为编程者必须明确规定要传送的程序状态信息。

TACOMA用meet命令来实现Agent间通信。在meet过程中，客户Agent向服务Agent发送一个有请求内容的briefcase，服务Agent则回应一个有结果的folder。TACOMA的高层服务都由其它的Agent提供，如tacfirewall代理负责接收移动的folder，重新运行程序。TACOMA还有后台监视代理，在Agent意外消失时它将重启Agent。

在TACOMA已发布的版本中，它的安全机制非常脆弱。它的开发者目前正在解决系统的容错、安全性等问题。

(7) Mole

Mole是德国Stutgart大学开发的基于Java的移动代理系统，模型的主要概念是代理和场所(place)。代理分为移动代理和系统代理两种。移动代理是一个封闭的Java对象集合，不引用外部对象，这有利于简化代理的传递机制。移动代理是多线程的实体，每个代理都有一个独一无二的全局标识符。系统代理是静态代理，向移动代理提供各种服务，如

资源访问、目录服务等。Place是移动代理的驻留场所，一个节点可以有一个或多个Place。

在Mole中，代理间的可靠通信机制由所谓的COM对象实现，COM对象包括RMI对象和Messaging对象两种。通信双方事先必须建立会话(session)，COM对象和Session相联系。一般移动代理间采用RMI对象通信，移动代理和服务代理间用Messaging对象通信。但在同组的匿名Agent间的通信则采用了OMG的事件模刑(event model)。

Mole的移动代理通过和系统代理交互完成任务，移动代理只能和主机或其它代理通信而没有直接访问资源的权限。一个Place也可以限制进入它的Agent类型。值得一提的是，Mole研究人员在对如何保护Agent不受主机的恶意攻击进行研究。

(8) Voyager

Voyager是ObjectSpace公司开发的系统，被称为Agent-enhanced object request broker。它不是一个纯粹的移动代理系统，因为它除了支持移动代理外也支持传统的分布计算。在所有的基于Java的系统中，Voyager和Java结合得最紧密。

Voyager有一个虚拟对象工具(virtual object facility)用于把Java对象包装成虚拟对象。虚拟对象具有某些移动代理的特征，如可以在网络间移动、有自己的生命周期。但虚拟对象不一定要有自己的线程，它们也可以是简单的对象。

Voyager采用一个轻量级的叫Messanger的代理来传递消息，有同步、异步和将来3种消息类型。当Agent移往其它节点时，会在原地点留下叫“秘书”的对象，该对象将把发给Agent的消息传递到新的地址，也就是说可以给移动中的Agent发送消息。这是Aglet、Odyssey等系统做不到的。Voyager的Agent移动也和Aglet等系统有区别。Voyager虽然也提供了Agent服务器，但不需要每个节点都运行一个服务器。因为Agent不光在服务器间移动，它也可以移动到其它虚拟对象空间。

1.5 移动代理技术的应用和发展趋势

移动代理技术的优点和特点使其具有很大的应用价值，其应用领域包括移动计算、分布式信息检索、网络管理、电子商务、信息发布等等。

目前，国外对于移动代理技术的研究还在不断进行，并且发展迅速。已经有很多科研组织和公司开发出了较为完整的移动代理系统，并尝试运用于实际。可以预见，移动代理技术将越来越被广泛地应用，未来的网络将是移动代理主宰的网络。总结起来，移动代理有以下几大发展趋势：

(1) 进入真正配置阶段

随着移动代理标准不断完善，不同移动代理平台之间也逐渐实现兼容，最终达到统一。在未来，移动代理平台将在网络中迅速普及，就好像现在的 Java 虚拟机一样，甚至可以集成到操作系统的常用软件之中。

(2) 代理设计模式化

未来的移动代理程序设计将被完全的模式化，这些模式包括迁移模式化、任务模式化、代理实体之间的交互模式化等等。代理设计的模式化将极大加快代理程序的生产速度，使之工程化。同时，也促进了其他辅助设计工具的产生。

(3) Java 将作为平台

Java 语言天生具有平台无关性，同时还具有安全性高、动态类加载、多线程、对象能够序列化等特点。这些特点恰好正是移动代理所需要的。况且，现今 Java 语言如日中天，而移动代理正好需要一个统一的软件环境，所以 Java 作为移动代理的研发平台是发展的必然趋势。

(4) 标准化愈加完善

对于移动代理，现在已有的两大标准 MASIF 和 FIPA 都不健全，没有达到 Agent API 一级。随着移动代理技术的不断普及，标准将最终达到统一而细致。

(5) 与嵌入式设备相结合

移动代理技术着眼于任何具有计算能力的网络节点，随着嵌入式设备性能的不断提高和无线网络技术的发展，移动代理技术将势必与嵌入式设备相结合，使之成为由移动代理构成的虚拟网络中的节点。

2 移动代理的安全问题剖析及对策

2.1 移动代理面临的安全问题

早期的移动代理研究已经为我们定义了几种安全威胁，主要有三大类：

2.1.1 移动代理迁移及通信的安全

当在开放的网络中传递信息时，固有的一个缺点就是不安全性，因而当移动代理在网络中巡游时，它的代码和数据都是不安全的。同时，移动代理和异地主机平台，代理和代理之间进行通信的时候也不可避免地会受到安全问题的困扰。这些安全问题可以分成两种类型：被动攻击(passive attacks)和主动攻击(active attacks)。

(1) 被动攻击：在这种攻击模式下，攻击者在不干扰网络系统正常工作情况下，监视信息的传递过程，采取侦听、窃取、破译等手段，企图从信息流中获得有用信息。最简单的被动攻击方式是监听，它企图获取信息包的内容，会导致敏感信息从消息包中泄露。另一种被动攻击方式叫做流量分析。攻击者不必读取信息内容，只是通过对系统进行长期监视，利用统计分析方法对诸如通信频度、通讯的信息流向、通讯总量的变化等参数进行研究，从而对通讯双方进行分析，获取所需的敏感信息。

(2) 主动攻击：另一种攻击方式是主动攻击，它也是最常见的攻击方式。主动攻击以各种方式有选择地破坏信息，如修改、删除、伪造、重放、乱序、冒充等。它主要威胁信息的完整性、可用性和真实性。常见的主动攻击手段有：

① 中断：截获信息包，打断通讯双方正常的信息交换。

② 冒充：通过欺骗通讯系统(或用户)达到非法用户冒充成合法用户，或者特权小的用户冒充成特权大的用户的目的。

③ 篡改：改变信息内容，删除其中的部分内容，用一条假信息去代替原始信息，或者将某些额外信息插入其中。

④ 抵赖：这是一种来自合法用户的攻击。比如：否认自己曾经发布过某条信息、伪造一份对方来信、修改来信等。

2.1.2 恶意移动代理对执行主机或移动代理平台的攻击

恶意移动代理包括被攻击者修改过的代理和攻击者自己生成的代理，它能进行以下的攻击：

(1) 恶意代理程序可以伪装成其他代理使其有权限接触到主机的敏感信息，比如它可以在访问某个代理服务器的时候可能会设法打开包括该公司商业机密的敏感文件，并将这些信息发送给它的创建者，从而利用这些信息在商业竞争中获利。

(2) 恶意代理可以破坏执行主机或者移动代理平台的系统资料，比如它可能在访问某代理服务器的时候删除某个重要文件甚至格式化整个硬盘。

(3) 拒绝服务攻击。恶意代理程序故意用完系统资源(比如硬盘空间、内存、网络端口等)，从而使服务器无法完成与其他代理程序的正常交互。

(4) 扰乱性攻击。这种攻击对于服务器系统的破坏较小，它通常是恶作剧性质的，如不断地在服务器上打开各种应用程序的窗口或者使机器不断地发出蜂鸣声等。

2.1.3 恶意主机或移动代理平台对移动代理的攻击

恶意主机是由攻击者设置的，这样的主机看起来似乎能为移动代理提供正常的服务，但是他们是为攻击者服务的，它有以下攻击[11]：

(1) 攻击其他主机。通过滥用主机间的通信，攻击者可以使用“拒绝服务”的攻击方式来攻击其他主机，其效果就好像是在其他主机上产生了一场移动代理的洪水。一个恶意的主机还可以产生一个恶意代理来危害其他主机。

(2) 恶意主机可以仅仅破坏或者中止移动代理，从而阻止该代理的执行；或者延迟执行以破坏某些时限约束。

(3) 探测和操作代理间的交互，比如窥探电子钱包代理中的划账信息以及施加恶意的控制。

(4) 恶意主机可以读数据、代码和控制流，比如代理密钥、执行策略被泄漏，控制流被推导。

(5) 操作数据、代码和控制流，比如向代理注入木马，根据主机意图修改数据或引导代理的行为。如当一个代理负责为用户收集某种商品的最佳价格时，该主机通过篡改代理所收集的先前的服务器的报价，以欺骗用户误以为其提供的价格是最佳价格；恶意主机可以修改移动代理的代码，使其在漫游到其他主机时或者返回源主机时进行恶意攻击，在通常情况下，源主机将其发出的移动代理视为可信的，其接入本地资源的权限也就更大，因而这类代理的攻击危害也就更大。

(6) 恶意主机可以篡改代理路径，比如将一个代理导向恶意主机期望的站点。

(7) 返回错误的系统调用，主机将错误的系统调用结果返回给代理而影响其行为和控制决策过程。

(8) 冒充或伪装，主机隐藏其自身标志获得代理信任以实施攻击，比如让代理执行

某些本不应该执行的功能。

2.2 已有的移动代理安全解决方案

2.2.1 移动代理平台或执行主机的安全保护方案

恶意移动代理会给移动代理平台带来各种各样的安全问题。除了移动代理本身带有破坏意图之外，其设计缺陷、迁移途中可能遭遇到篡改攻击等都是移动代理可能攻击平台的原因。同时，移动代理平台也可能受到来自网络中其他实体的攻击。通常，攻击可以消耗系统资源、进行未授权的访问、窃取系统保护数据、破坏系统正常运行等方面进行。

传统的安全技术通常采用身份认证、进程隔离、资源的存取控制、审计等方法来保障系统的安全，针对移动代理自身的特点，还提出了一些新的方法，如：代码签名、状态鉴定、路径历史记录、携带验证代码、软件故障隔离和安全的代码解释等防范措施。

(1) 代码签名法

由于移动代理平台在为移动代理提供资源协助代理完成任务时难以真正了解代理的意图，经常会遭到代理的攻击，因此，移动代理平台通常会建立权限管理、访问控制等安全机制，确保移动代理平台的安全，而这些都是建立在身份认证的基础上的，从而使得保证代理来自于平台信任实体的技术至关重要。代码签名是其中最常用的一种。

代码签名方法SC(Signed Code)是使用数字签名对移动代理的全部或部分代码进行签名的方法。数字签名通常作为一种对代理来源、真实性和完整性的认证手段，由代理的创建者、使用者或者曾访问过代理的实体进行签名，使得代理平台通过验证签名来确定代理身份，进而结合其他安全手段保障系统安全[12-15]。

此方法适用于支持公钥基础设施的代理平台。

(2) 状态鉴定法

由于移动代理经常需要在各种异构的代理平台中迁移、运行，而各个平台之间可能存在利益冲突，从而发生篡改代理状态以影响代理在其他节点的运行结果的事件。为了能使代理对其状态进行监控，使其它平台能够检测到修改的发生，通常使用状态鉴定方法。

状态鉴定方法[15]SA(State Appraisal)是一种能够同时保护代理和主机状态安全的方法。它一方面利用移动代理携带的状态鉴定函数检测代理状态是否发生了异常变化，一方面结合代理平台上的授权机制，根据代理的身份、当前状态等信息控制代理在平台中可以拥有的权限。

该方法适用于代理平台具有完善的授权访问机制，移动代理能根据其当前状态和在站点上将要执行的任务确定其所需的权限的系统。但由于移动代理在代理平台中获取的信息是动态的，难以确认其状态的改变是由于恶意修改造成的还是由于平台信息变化引起的，因此设计难度较大，通常该方法的使用前提是状态鉴定函数能检测到对代理以后访问的站点或用户造成伤害的状态。

(3) 路径历史记录法

由于代理在完成其任务时可能访问过一些当前代理平台不信任的站点，而使得代理平台可能遇到潜在威胁，因此，代理平台有权检查代理曾经访问过的站点，因此，需要移动代理携带其路径信息，并确保其正确性、完整性。

路径历史记录法PI[16-18](Path Histories)是满足上述要求的一种方法，它要求移动代理保留其已访问过的平台记录，使得代理平台能够根据该记录决定是否运行该代理，并确定其权限。显然，路径记录将随着移动代理移动次数的增加而增加，相应的路径验证的代价也将增加，因此，对路径历史记录的验证既可通过简单察看当前路径历史记录中的当前路径记录来决定是否信任代理以前访问过的所有平台，也可以验证所有记录以进一步确认其真实性。

(4) 携带验证代码法

携带验证代码的方法PCC[19](Proof Carrying Code)是一种由代码撰写者按照代码执行者(代理的计算平台)的安全规则正式证明代理程序拥有其所要求的安全属性，并由代理携带其安全性证明同时到达代理平台，由代理平台验证其安全性的方法。由于证明代码是按照计算平台的规则撰写的，故而可以直接被验证，不需要使用额外的密码学方法或其他辅助工具。验证通过后，代理即可顺利运行，不需要进行进一步的检查，而对代码或证明码的任何篡改都会导致验证失败，因此，使用此方法的安全性较高。

该方法的理论基础是建立在逻辑理论、类型理论、形式化证明等方面，但在实际应用时还存在许多需要解决的难题，如安全的形式化标准、证明码的自动生成、证明码相对于代码的复杂度肚量、理论上可以接受的证明码的最大长度限制等等，使得该方法无法得到广泛应用。

(5) 软件故障隔离法

软件故障隔离SBFI[20](Software-Based Fault Isolation)是一种用软件方法将应用模块隔离到单独区域执行的方法。该方法为每个在平台上运行的程序建立一个独立的虚拟控件，如果该程序不具备平台的信任条件，则由该软件将其所有行为隔离在为其分配的虚地址空间中，如将其所有内存操作都转换为对其分配的虚地址空间内部代码段和数据段的访问，同时，对系统其他资源的使用也只能通过为该区域分配的一个唯一标识符向平

台申请，由平台决定是否提供该服务。因此，在使用了这种方法后，用一般的非安全程序设计语言设计的程序也能在该平台为其分配的虚地址空间中安全执行，不会造成重大损失，此方法通常也被称为沙箱操作。

(6) 安全的代码解释法

移动代理的异构性使得移动代理通常采用解释型程序设计语言或者多种平台都支持的通用语言来开发，因此，支持移动代理的系统通常具有解释代理执行的解释器，利用解释器的安全检查和限制功能自然成为确保代理平台安全性的重要关卡。例如，目前最流行的java解释器JVM[21]就具有强类型检查、沙箱机制等安全功能。

安全的代码解释 SCI(Safe Code Interpretation)的主旨是由代理解释器检测代理的代码指令安全性，对代理中有害的指令拒绝执行(如拒绝执行将任意的字符串作为程序段数据的指令)或转化为安全指令执行，从而从入口处防止代理恶意行为的方法。

2.2.2 移动代理的安全保护方案

(1) 部分结果密封(Partial Result Encapsulation)

用于检测恶意主机篡改的一种方法是对在每一访问平台上代理执行的结果进行封装以便当代理返回到源平台或者在指定中间平台上进行检查。封装可能利用不同的机制来完成，以实现不同的目的，例如：使用加密提供保密性、或者使用数字签名以保证完整性与责任性。封装的信息、一般包括代理平台上对于查询的响应或者操作。提供三种方法对部分结果进行封装：

① 代理代码中提供一种封装信息的方法；

② 依赖于代理平台的封装能力；

③ 依赖于第三方对结果加盖数字认证。

尽管三种方法都不能阻止针对移动代理的恶意行为，但是它能够检测出特定类型的篡改行为。其中第一种方案的优点是它不用考虑平台的能力而在设计时独立应用。

这种方法所采用的是DES加密算法。通常由代理收集的信息量相对于密文及包含在其中的加密关键字来说小得多。一种称为Sliding加密方法[22]允许对少量的数据进行加密并且得出高效的结果。使用变化加密方案是指在此方案中代理携带一个公共密钥，并且对从每一台访问的主机上收集的信息进行加密。最后当代理返回源平台时，信息就可以使用那儿维护的公共密钥进行解密。加密的目的是保密性，但在加密之前，也要遵守信息的完整性。

另一种代理结果信息封装策略是使用部分结果认证代码，它是使用密码而形成的加密校验和。这种技术要求代理与源平台都维护或者递增地产生用于PRAC计算[23]中的密

码列表。一旦一个密码被用于封装收集到的信息，则在跳到下一个平台之前，代理会将其毁掉，以保证向前的完整性。向前的完整性保证如果当它访问过的主机中某一台是恶意的，则在此之前的部分结果依然有效。

PRAC技术有许多局限性。最重要的是恶意平台尝试着保留源密码以及代理产生密码的函数。如果代理访问某一平台或者同伙平台，则以前的部分结果就有可能被拷贝或者认证被修改而不被发觉。由于PRAC方法是面向完整性而非保密性，则部分结果也可能被视为其访问过的主机的认证结果。

沿途的平台也可以对各自的部分结果进行封装。这两种区别不仅仅是封装机制在哪儿维护，而且涉及到责任及与此相关的可靠性。如果代理触发一个通过平台编程接口的数字签名封装，则实现时必须保证包含足够的上下文信息(如时间及查询信息)。

Karjoth设计了一种面向部分结果封装技术[24,25]的平台，这种方法是构造一个将每一结果入口与前一入口及将要访问的下一平台标识绑定的封装部分结果链表。每一平台使用其私有密钥对入口进行数字标识，并且使用安全散列函数在每一入口内将标识与结果关联起来。除了向前的完整性，这种技术还通过使用源平台的公开密钥对收集到的每一段信息进行加密，从而提供了保密性。而且向前的完整性比PRAC更强，因为平台不能够修改链表中的入口。

(2) 探测体(Detection Object)

探测体是验证数据窜改的另一类重要技术[26]。探测体是作为代理状态一部分的数据哑元或属性，并不为主机所感知。如果探测体未被修改，那么从一定程度上可以确信合法的数据未被破坏。例如，假设代理去搜索最便宜的机票，但是到达一个恶意平台时，有可能把以前的搜索纪录删除，从而使自己的机票价格成为最便宜的。当使用探测体技术时，可以在代理发出前，加入一条机票纪录，代理返回时，检验这条记录是否被修改或删除，若是没有改变，则可以认为代理返回的数据是可信的。缺点是，首先是与与具体应用相关，为一个应用设计的一个探测体不能直接用的另一个应用中。另外探测体的设计同代理的设计是相关的，探测体必须能够瞒过主机，而又不影响代理的计算。同时，代理的得出正常计算结果的同时，还要保留探测体。最后还要经常更新探测体，用以防止通过数次已经比较的结果总结出那个数据的探测体。类似的探测码是一些随机的代码片段，注入到代理的原始代码中[27]。该类方法保护数据完整性，但编码的设计必须足够好。

(3) 复制(Replication)和表决(Voting)

错误代理与恶意代理行为相似，因此，代理具有一定的容错性有助于防止其本身的恶意行为的发生。确保移动代理安全返回其源平台的技术是通过使用复制与表决。具体

做法是拷贝多个代理。尽管恶意主机可能破坏一些代理的拷贝，但足够多的拷贝可能避免这种遭遇以成功完成计算。对于计算的每一阶段代理平台通过代理携带的证书来验证代理是否完整。包含在计算的某一阶段的平台希望知道前一阶段可以接受的平台。平台只向下一计算阶段传播它根据输入认为有效的复制代理的子集。

本方法实施有三个前提假设：只有少数代理平台是恶意的或在执行时出错；代理副本之间可以独立执行；拥有信任的第三方收集整理结果。

文献[28]提出的彼此路线记录方法(Mutual Itinerary Recording)是利用两个(或以上)相互协作的对等代理来记录各自的路线，这样，代理可避免访问其协作者已访问过的主机。该技术的前提在于恶意主机是少数的，如果代理遭遇恶意主机，其对等的协作代理不太可能会同时遭遇另一恶意主机。

此方法适用于代理易于复制、代理任务可以被划分为若干阶段、任务的完成与所在平台无关的应用。该方法在满足实施前提的情况下，一方面需要消耗大量的资源进行重复计算；另一方面，还需要建立主机之间的安全通道确保计算结果正确返回。同时，信任的第三方主机的选择成为安全瓶颈。因此，通常需要结合加密、签名等传统方法和相关的密码协议以在实际应用中获得更好的安全性能。

(4) 窜改探测(Tamper Detection)

由于代理需要迁移到相关站点上完成特定任务，必须允许该站点操作其代码、改变其状态并运行，但站点是否如实的完成了代理任务则无从得知，如果遇到某个恶意平台，擅自修改代理的状态或执行结果，会导致代理的整个执行过程遭到破坏，其所携带的最终结果无法正确的反映代理的执行意图。此时，需要能够对代理获得的结果进行正确性验证。

窜改探测维护计算的完整性，使用基于密码的完整性证明或完整性线索确保计算是按照代理代码的指令进行的。执行痕迹(Execution Tracing) 是在代理的整个生命周期内记录其执行操作的日志记录[29]，当代理携带结果返回原发站点后，可根据日志记录由代理的所有者对其执行过程和结果进行校验，用以验证结果正确性的方法。但该方法有几点局限：

① 生成的痕迹可能比较大；

② 只适用于不共享存储器的单一线程的代理，否则需要追踪共享存储器的所有线程以及线程时间顺序，从而变得不太可行；

③ 代码需要是静态的，否则编译器需要将被编译模块的执行情况映射到原始代码中。

全息证据(Holographic Proof)可加快证据检测过程，使得校验器通过检查全息证据的少数位就可以以概率的方式确定证据的有效性[30]。但证据转换为全息证据是计算密集

的，且是由主机完成的，如果校验器检索的证据位被主机获知，那么就可能丧失有效性。另外，全息证据尺寸很大，不宜全部发送给校验器以避免主机窥探被检索位。为解决这一问题，文献[31]提出了一种加密的私有信息检索技术，使得全息证据可以在主机上进行校验并返回结果，并且在这一过程中主机无法获知被检索的信息位。

保护性断言(Protective Assertions)可动态确保代理的状态是正确的[32]。如果断言被旁路，那么返回到代理属主的信息就缺乏必要的部分，从而指示恶意行为的发生。该类方法保护计算完整性，但作为一类软件窜改证明方案，有受到攻击的可能，并存在应用的局限性。

(5) 窜改阻止(Tamper Proofing)

窜改阻止是使得恶意主机无法实施对计算过程的窜改行为。一种有效的方法是黑盒安全(Blackbox Security)。黑盒具有代码和数据不能被读取亦不能被修改的属性，但问题是尚无已知的算法可确保黑盒属性得到保护。移动加密技术(Mobile Cryptography)被看作是实现黑盒安全的一种可能技术[33]。但对输入程序的类型以及代理交互有限制而不能普遍适用。

有时限黑盒法是指采用各种密码学方法或程序变换方法将代理转换为功能等价但结构不同的黑盒代理，使其在一定时间内运行时不暴露原代理的执行意图，确保其数据、程序安全的方法。其特点是只在一个特定的时间段内确保代理的黑盒属性，即代理描述的代码和数据既不能被读取，也无法被修改，但当超过既定的时间段后，则无法确保其安全性，可能被攻击者采用各种攻击方法破解。有时限黑盒法通常采用修改程序属性的方法将代理明文程序转换为意图更为隐蔽的程序：如修改程序的状态，采用动态方法创建，以避免静态分析；修改数据的原始形态，根据数据本身的特性(类型、值和存放位置等)，使用动态数据类型隐藏其类型；用拆开、合并的方法隐藏数据值；用分别存放或移动的方法隐藏其存放位置等等。无论采取何种方式，其目的都是为了能隐藏代理的实际意图。有时限黑盒方法与(无时限)黑盒方法不同，其适用范围非常广泛，可以用于任何代理，且代理到达接收方后不需要解密直接执行。但算法的安全性依赖于黑盒构造方法本身的抗攻击能力和攻击者的能力，即攻击者分析黑盒，构造思维模型的正面分析能力和构造程序侧面攻击能力。因此，使用该方法存在的主要问题在于：难以确定特定算法的时限长度，即缺乏形式化的描述方法和良定义的数学模型估算时限长度。同时，有时限黑盒法在使用时需要在时间、空间方面付出代价：如黑盒程序的创建时间、传送时间、执行时间和加速时间(扰乱了原有的数据结构，使得一些固有的加速技巧无法使用)，程序额外需要的各种空间，因此，需要在安全性和效率上进行权衡。

(6) 防窜改硬件(Tamper - proof Hardware)

防窜改硬件是指以硬件的方式(这一点与其他方案显著不同)提供窜改证明或者防窜改手段。这是一类十分重要的技术，可用于实现数据和计算的完整性、私有性。采用防窜改硬件，代理属主可以信任执行环境构造者而不必直接信任主机属主[34]，因为执行环境是基于硬件防窜改的，主机属主无法进行窜改。依赖硬件构成的防窜改环境，代理可以存储与运行在一个被隔离和被保护的环境中。例如，采用智能卡作为安全内核以执行某些秘密功能和保持秘密数据，从而保护数据和计算的完整性与私有性[35]。防窜改硬件具有很强的防护能力，特别是可以杜绝各种软件攻击。目前存在一个普遍的认识，即除非采用可信的防窜改硬件，主机总会完全控制代理的执行，从而无法最终确保代理的安全。但实际问题是，构造真正防窜改的硬件不是一件容易的事情，专门的硬件攻击手段也可能破坏其防护能力。另外，这种方法要求在所有的主机中添加专门的硬件部件，从而形成一个过于严格的限定条件。

(7) 计算功能加密(Computing with Encrypted Functions)

由于一般情况下，移动代理程序采用的是明文数据、明文程序和明文消息，使得移动代理程序容易受到各种攻击，即使移动代理采取各种保护措施后，由于在平台中运行时，一般仍然需要在解除各种保护措施后，将其恢复为明文后予以执行，从而将程序意图完全暴露在执行环境中，一旦平台为恶意平台，则可以根据需要对代理进行篡改，达到自身目的，与无法了解程序意图的表面攻击(如拒绝服务攻击、对程序或输出数据的随意修改、重试攻击等)相比，具有更大的危害性和更强的隐蔽性。

加密函数计算方法是指将代理中不希望暴露其功能的函数转变为加密函数，在不暴露函数意图的情况下移动到目标平台中执行[36]。其特点在于：在将原函数转变为加密函数后，仍然要保持程序的可执行性，得到的程序必须是一组处理器或解释器能够理解的明文指令，而大多数一般的数据加密技术都不能满足要求，因为一般的数据加密技术，将原函数作为数据流输入，加密后得到的也是一个任意的数据流，无法作为程序予以执行。可见，加密函数计算方法需要个别问题个别对待。

目前，加密函数计算方法还不能应用于任意函数，只能对一部分可以求逆的多项式函数和有理函数适用。其安全性依赖于分解加密函数的可能性和复杂性，由于加密函数计算方法在适用范围内，能在不解密的情况下执行加密后的程序，确保了程序代码的秘密性和完整性，因此，具有很强的安全性和实用性。

(8) 环境密钥生成(Environmental Key Generation)

代理为了保证传输过程中关键信息的安全，将关键信息加密，并将密钥保存在程序中，此时，代理虽然能够抵抗网络中其他攻击者的窃听、欺诈行为，但是，当代理到达指定平台时，或到达其他平台伪装的指定平台。则代理将会利用该密钥解密得到关键信

息，此时，其携带的所有信息都将暴露在该平台面前，可能导致用户关键信息的泄漏。

环境密钥生成是指代理在预定义环境条件为真时，生成相应密钥以解密某些被加密了的代码[37]，所需环境条件通过加密技术来进行隐藏。该技术使得攻击者无法通过直接读取代码来理解解密条件和相应的行为，在代理解密其代码后，已经没有什么时间可供恶意主机分析代码并实施攻击。这一技术存在的弱点是主机可以简单地让代理输出解密的代码而不是启动执行，另外主机平台通常对动态创建的可执行代码加以限制。

由于该方法利用环境信息构造解密密钥，因此其加解密算法可以选择速度快、效率高的对称加密算法，而不需要使用速度相对较慢的公钥加密算法进行保护，具有实现简单、效率高等特点，适合加解密大量的数据或代码。

(9) 嵌套的执行环境(Nested Environments)

嵌套的执行环境让代理自己提供自身的运行环境，而不是直接在主机上执行[38]。代理包括三个组成部分：代理本身、代理载体和代理解释器。其中，代理本身是用不能被主机所理解的语言书写的，是以数据形态存在的被动实体；代理载体携带代理本身到达目的主机，将其分发给代理解释器执行。这一方案的安全性表现在如下方面：代理载体和代理解释器形式上是两个独立的代理，可以单独分发并相互验证；代理本身无法直接静态分析，除非完全理解新语言和解释器功能；代理的动态执行路径同解释器执行路径相混杂，从而加深了分析难度。另外，由于保护机制同代理功能分离，从而对任意代理都可适用，可与其他保护机制直接集成在一起。这类技术可用于保护计算的私有性，但代价较高，因为要设计一种新语言(这种语言不能为其他人知晓，还要为了对付可能的破解而不断更新)，要设计解释器。另外，计算资源、带宽等消耗也变大，非直接执行还将导致速度变慢。

2.3 总结

在剖析了移动代理所面临的安全威胁后，我们总结出安全问题其实可以分为以下几个子问题[39]：

(1) 移动代理的代码与数据在传输过程中不能被第三方读到；

(2) 在传输过程中，协议信息、移动代理的代码或者数据的修改必须能被发现；

(3) 第三方不能够私自创建一个代理或者伪造一个代理并将其发送到执行主机上；

(4) 移动代理不能够将自己的信息暴露在未经认可的实体面前，也不能接受未经认可的实体发出的信息，更不能被未经认可的实体控制；

(5) 移动代理必须被保护起来免受恶意主机的侵害或者被恶意主机滥用；

(6) 主机不能允许未经认可的移动代理访问它不该访问的区域。

解决前三个问题便能保证移动代理的数据在传输过程中的安全，解决第四第五两个问题能保证移动代理的代码和数据的安全，解决第六个问题则能保护主机使之免受恶意代理的侵害。

现在已经证明使用基于PKI(public-key infrastructure)的安全机制能够有效地解决上述大部分问题，除了第五个问题。防护恶意主机侵害移动代理的问题目前是移动代理安全问题的热点也是难点。

3 嵌入式移动代理平台 μAglet 简介

3.1 μAglet 的整体架构

3.1.1 μAglet 的构思

从应用角度来说，移动代理平台应该具有平台无关性，而 Java 程序设计语言恰好具备这种特点，因此如何在嵌入式设备上运行 Java 程序是一个需要解决的问题，而解决这个问题的关键在于如何将 Java 虚拟机移植到嵌入式操作系统中。

SUN 公司为了将 Java 进一步普及，使之能够运行于嵌入式设备上，在 JVM 的基础上，对其进行了一系列的小型化工作，成果就是 KVM，一种千字节虚拟机。KVM 能够在内存相对较少的嵌入式设备上运行，而且将与底层操作系统相关和无关的代码分开，具有高移植性，还能够尽量完整而快速地运行 Java 程序，因而是嵌入式设备上 Java 虚拟机的一个很好的选择。

Aglet 是日本 IBM 公司完全用 Java 开发的移动代理技术，并开发了使用的平台 Aglet Workbench，供人们开发或执行移动代理系统。可视化环境 Aglet Workbench 提供 ATP 传输协议传输 Aglet，提供管理界面 Tahiti 监视和控制 Aglet 的运行。它的主要特点是：提供了一个简单而全面的编程模型，为代理间提供了动态和有效的通信机制，同时具有易用的安全机制。Aglet 同时传送代码及其状态，以线程的形式驻留在计算机上，可随时暂停正在执行的工作，并允许把整个 Aglet 分派到一台计算机上，再重新启动执行任务。由于 Aglet 是线程，因此不会消耗太多的系统资源。Aglet 是一个相对比较成熟的移动代理平台，应用也比较广泛。

3.1.2 Aglet 与 KVM 的兼容性分析

但是 Aglet 运行于标准的 JVM 平台上，而 KVM 的功能较之 JVM 弱，因此 Aglet 是无法直接在 KVM 上运行的，也就是两者不兼容。这些不兼容主要表现在如下方面：

(1) 网络功能

目前版本的 J2ME CLDC 并没有定义一个标准协议，也没有提供 java.net 的 Java API，而是提供了通用联网框架(Generic Connection Framework)的服务。通用联网框架的结构如图 3.1 所示。

最上层的接口是 Connection，其他的接口都从那里继承。通常使用的是分组数据交换和电路交换，因此在联网框架中相应的定义了 DatagramConnection 和 StreamConnection。由于在基于流传输中我们需要对输入流和输出流通是具有操作的能

力，因此 StreamConnection 扩展了 InputConnection 和 OutputConneciton，我们经常使用的 Conn.openInputStream()、conn.openOutputStream()方法都是在这两个重要的接口中定义的。

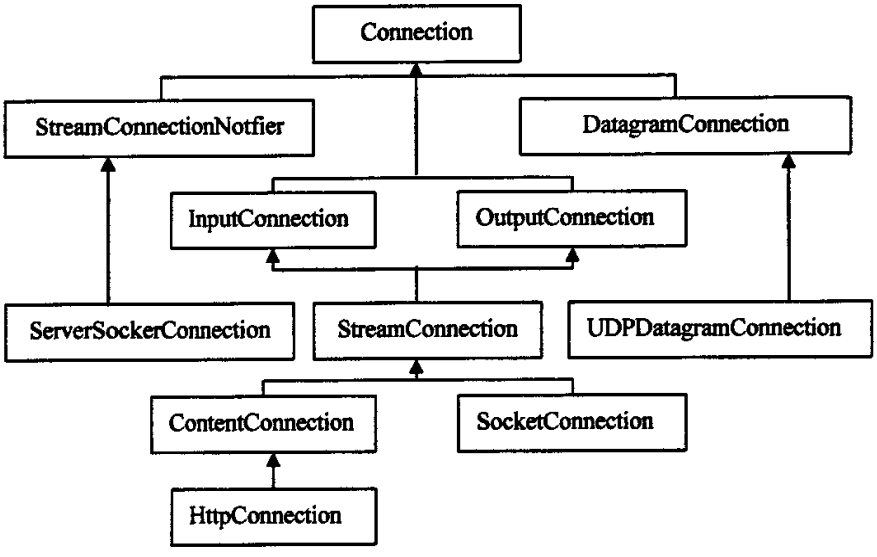

图 3.1 通用联网框架

Fig. 3.1 Generic connection framework

StreamConnectionNotifier 接口定义了连接监听器应该具备的能力，它的方法 acceptAndOpen()方法返回一个 StreamConnection 类型的连接。其余的接口和类如 ServerSocketConnection 、SocketConnection 、ServerSocketConnection 、UDPDatagramConnection、HttpConnection 等在 CLDC 中并没有实现，而是由 MIDP2.0 中扩展出来的。

而 Aglet 通过 Java.net 提供的大量 API 接口函数来实现其传输协议 ATP。因此，对于 μAglet 平台，其底层的网络通信协议必须由基于上述的通用联网框架来实现。

(2) 对象的移动

通过第三章的分析，我们已经知道，Aglet 采用标准的 Java 序列化方法传输 aglet 对象，但是在 CLDC 中并没有对序列化机制进行支持。因此，在进行对象移动时，必须要实现自己的 serialize()和 deserialize()方法。

(3) 图形化支持

Aglet 采用了 GUI 工具 Tahiti 进行管理和操作 aglet 实体。在本例中，由于硬件上不支持图形化(没有 LCD 显示屏)，而且移植的 KVM 也没有包括对图形化的支持。因此，

在实现 μAglet 时，所有与图形化有关的模块都应该去掉。与此同时还要提供一个较为强大的命令行接口，以及在文本界面下，较好的系统信息显示。

(4) 异常处理

在 Aglet 中，定义了大量的异常，并相应的对这些异常进行处理。有些异常继承自标准版 Java 中定义的异常类，而这些类却在 CLDC 中没有定义。因此，对于这些异常，需要重新定义，相应的处理也要做调整。

(5) 事件处理

在 Aglet 中，部分用到了事件处理系统，如 aglet 迁移到新的运行环境时执行的操作，aglet 刚刚被创建时执行的操作，aglet 被删除时执行的操作等等。一些接口和类继承自标准 Java 中的 java.util.EventListener 和 java.util.EventObject。但是在 KVM 中，对事件的处理，API 并没有提供和 J2SE 一样的事件处理接口。取而代之地，KVM 采用了自己独特的事件处理方式，主要分为以下 4 种：

① 同步通知(阻塞方式)

这种方式适合于 KVM 直接从虚拟机内部调用本地的 I/O 事件系统函数的情况。由于在虚拟机内部，KVM 只有一个负责控制的物理线程。当本地函数被执行时，其他的 Java 线程不能被处理。但是仍有许多情况，这种解决方法也是可以被接受的，因为可以精心设计本地函数，保证它们足够短小和高效。

② 在 Java 代码中轮询

通常的事件处理能够通过本地代码和 Java 代码结合的方式实现。这是一种在等待事件完成期间，允许其他 Java 线程执行的简单方式。当使用这种方式时，一个轮询的 Java 循环通常被放到 Java 运行时库的某个地方，因此对于应用程序来说，这个循环是隐藏的。普通的过程是，运行时库初始化一个短小的本地 I/O 操作，然后重复的查询 I/O 操作的状态直到完成。Java 代码的轮询循环总是应该包含一个 Thread.yield 调用，这样其他的 Java 线程能够被允许有效的执行。

③ 在字节码解释器中轮询

第三种实现事件处理的方法是使用字节码解释器周期性地调用本地事件处理操作。这种方法是上述同步通知的一个变种。这种方法原来被广泛地在 KVM 中使用，如实现 Palm 平台的 GUI 事件处理。

在这种方法中，一个本地的事件处理函数在解释器循环中被周期性地调用。由于性能方面的原因，通常不能在每条字节码之前执行完，而是几百条字节码左右。这种方式下，事件处理的代价被分化。通过改变调用事件处理代码之前的字节码的数量，KVM

设计者能够控制在事件处理延时和 CPU 寻找下一个新的事件所耗费的时间之间取得均衡。

这种方法的优点是执行上的代价要少于在 Java 中轮询，并且事件通知的延时要更精确和更易控制。

④ 异步通知方式

原始的 KVM 实现只支持上述三种事件处理方式。然而，为了支持异步事件处理，必须引入新的机制。

异步通知适合于事件处理并行发生，而虚拟机继续它的执行的情况。这通常是最有效的事件处理方式，通常延时也很少。但是，这种方式通常需要底层的操作系统提供实现异步事件处理的机制。这种机制不是在所有的操作系统下都有。并且，这种方法实现起来有点复杂，因为虚拟机的实现者必须意识到同步和死锁等问题。

有两种方法实现异步通知：一种是利用本地(操作系统)线程，一种是利用软件中断、回调函数或者轮询函数。

在 KVM 中，事件处理主要有两个层次，如下所述。

在解释器的顶部，是下面的代码：

```
if (isTimeToReschedule())
reschedule();
```

标准的 reschedule 代码应完成以下的操作：

① 检查是否还有存活的线程，如果没有则停止虚拟机。

② 检查是否已经经过足够的时间来允许一个等待特定时间到达的线程重新运行。如果有这样的线程，则自动重新运行。

③ 检查是否有 I/O 事件出现，并且它允许相关的线程在适当的位置竞争 CPU。

④ 尝试切换到其他线程。

由于性能原因，上面的操作用宏来实现，定义在 VmCommon/h/events.h 中。与设备相关的事件处理代码可以放在这里。

事件处理的第二个层次是下面的函数：

```
GetAndStoreNextKVMEvent(bool_t forever, ulong64 waitUntil)
```

如果在宿主机的操作系统出现新的事件，这个函数必须掉用一个特定的函数 StoreKVMEvent 来使事件对于 KVM 可用。如果没有新的事件，这个函数只是简单地返回。如果 forever 参数是 true 的话，函数将一直等到事件出现；如果 forever 参数是 FALSE 的话，函数将至多等 WaitUtil 这么长时间，直到事件出现。

事件处理比较复杂，在本例中，采用了在 Java 代码中轮询的方式，即用 Java 线程来循环查询事件的到来。这样做虽然只能处理同步事件，但实现起来较为简单。

3.1.3 μAglet 整体架构的定制

基于以上的分析，对 Aglet 进行裁减，只留下最核心的部分作为嵌入式移动代理平台 μAglet 的原型。主要进行修改和裁减的部件如下：

(1) 删除所有的与安全问题相关部分。

(2) 不再使用 ATP 协议和 Java RMI 进行对象迁移，取而代之地，用 GCF 框架提供的网络通信方法编写自己的通信协议。

(3) 所有与事件处理相关的接口和类都改为继承自 Thread 类，并加入轮询代码。

(4) 去掉所有与图形界面相关的部分，增加命令行解释模块。

(5) 取消模式支持，简化路线类，以及消息类型。

(6) 简化异常处理机制。

其中，主要的包、类和接口的定义如下：

(1) com.ibm.aglet 包

com.ibm.aglet 包含了 3 个接口和 6 个类，是 μAglet 中最重要的包。

接口：

AgletContext：AgletContext 是 aglet 用来取得环境信息并向环境与当前其他在该环境中活跃的 aglet 发送消息的接口。它负责维护和管理主机系统环境中运行的 aglet。AgletContext 接口提供了一些重要的函数，如 createAglet()、getAgletProxy()、getHostingURL()、retractAglet()等等。

AgletProxy：AgletProxy 接口是 aglet 的外壳。该接口用来提供控制和限制对 aglet 的直接访问的机制。AgletProxy 接口中的一些重要函数，如 activate() 和 deactivate()、clone()、dispatch()、dispose()等等。

MessageManager：MessageManager 接口控制同时到达的消息。每种消息都有一个优先级，并根据优先级，被放到消息队列中的相应位置。MessageManager 接口中的重要函数，如 notifyMessage()、exitMonitor()、setPriority()等等。

类：

Aglet：Aglet 类是 aglets 的基础抽象类，直接继承自 java.lang.Object。Aglet 类是 μAglet 系统中最核心的类，其中定义了大量的方法。这些方法总结起来可以分为几大类：

与事件处理相关的方法，如

① 实现信息获取的方法，如 getAgletID()、getAgletInfo()等；

② 设置属性的方法，如 setStub()、setText()等；

③ 生存周期控制方法，如 dispatch()、clone()、deactivate()、run()等；

④ 消息处理相关的方法，如 notifyMessage()，waitMessage()，handleMessage()等。

AgletID：AgletID 类代表了 aglet 独一无二的标识符，直接继承自 java.lang.Object。

AgletInfo：AgletInfo 类是保存 aglet 相关信息的对象，直接继承自 java.lang.Object。它提供了提取这些与 aglet 相关信息的方法，如取得地址，取得类名，取得 ID 等等。

AgletStub：AgletStub 抽象类用来实现 aglet 的动作，一般不提供给程序员。

Message：Message 类是保存传递给接收者参数与类型的对象。在 Aglet 类中的 handleMessage 方法中，如果有请求，将设置回复。在 Message 类定义的方法中，主要包括了对参数的设置和提取、消息类型的判断和提取、对请求的回复等等。

Ticket：Ticket 类是 aglet 所持有的“车票”。一个 aglet 想要移动到某个地方，它必须拥有“票”。“票”包含了目的信息和到达目的的路线信息。

(2) com.ibm.aglet.event 包

com.ibm.aglet.event 包含了 3 个接口和 8 个类，定义了与事件处理相关的属性和操作。在 Aglet 中，这些与事件处理相关的类和接口主要继承自 J2SE 中的 java.util.EventListener 和 java.util.EventObject，但在 μAglet 中，这些类和接口只能继承 java.lang.Thread。

接口：

CloneListener：接收克隆事件的监听器。该接口提供了 3 个函数为 onCloning()、onClone()、onCloned()。

MobilityListener：接收移动事件的监听器。该接口主要提供了 3 个函数为 onDispatching()、onReverting()、onArrival()。

PersistencyListener：接收持续性事件的监听器。该接口提供 2 个函数为 onActivation()、onDeactivating()。

类：

AgletEvent：所有 μAglet 事件的父类。

AgletEventListener：所有与 μAglet 事件有关的监听器的容器类，实现了上述的三个接口，即 CloneListener、MobilityListener 和 PersistencyListener，是最经常被编程者使用的类。

CloneAdapter、MobilityAdapter 和 PersistencyAdapter：这三种适配器类分别接收克隆事件、移动事件和持续性事件。这些类只是为了通过继承它们，方便地创建监听，并只重载感兴趣的方法。

CloneEvent、MobilityEvent 和 PersistencyEvent：对应三种类型事件的类。

(3) com.ibm.aglet.system 包

com.ibm.aglet.system 包提供了 1 个接口和 4 个类，定义了与 μAglet 运行环境相关的属性和操作。

接口：

ContextListener：提供了一系列接收运行环境事件的方法，如 aglet 的激活、克隆、创建、派送、销毁、继续、状态改变等等，以及运行环境的启动与停止。

类：

AgletRuntime：提供访问本地或者远程的运行环境的方法。应用程序和 aglet 都不能创建它自己的运行时类的实例。

Aglets：该类为没有 AgletContext 和精灵线程的客户端接收进来的 aglet 定义了一系列方便的函数。

ContextAdapter：该适配器接收 μAlget 的运行环境事件。该类只是为了通过继承它，方便地创建监听，并只重载感兴趣的方法。

ContextEvent：运行环境的事件。

(4)ibm.agletx.util 包

ibm.agletx.util 包主要提供与 aglet 移动的路线和执行的任务相关的类。

类：

SeqItinerary：为 aglet 的路线定义一个抽象的接口。一个路线是一个[*Host*,*Task*]对的集合，当 aglet 到达 Host 时，执行相应的 task。该抽象类继承自 MobilityAdapter。

SimpleItinerary：是一个路线对象，能够指定一个目的地和一条消息，该消息能够在它到达目的地时发送给所有者的 aglet。

Task：在路线中定义的被执行的任务。

3.2 μAglet 的工作机理

μAglet 除了不提供安全机制，其他部件组成和相互关系与 Aglet 相同。但就具体实现来讲，在传输协议、序列化和反序列化以及事件处理方面有很大不同。

3.2.1 传输协议

在 Aglet 中，对象和消息的传输使用 ATP 协议。ATP 协议实际上是对 HTTP 协议的简单封装，基于请求/应答方式。ATP 为代理服务定义了四种标准的请求方法：派送、回收、获取、消息。

在 μAglet 中，采用 J2ME 提供的通用联网框架实现 ATP 协议。通用联网框架的设想是以 Java 接口的形式定义一些能够覆盖联网和文件 I/O 的通用方面的抽象。这个体系结构广泛支持各种手持设备，而将这些接口的实际实现留给了编程者根据其设备的实际功能选择要实现哪个接口。由 Java 接口定义的通用方面分为以下几种形式的基本通信类型：

(1) 基本串行输入(由 javax.microedition.io.InputConnection 定义)

(2) 基本串行输出(由 javax.microedition.io.OutputConnection 定义)

(3) 数据报通信(由 javax.microedition.io.DatagramConnection 定义)

(4) 用于客户机/服务器(client/server)通信的套接字通信通知机制(由 javax.microedition.io.StreamConnectionNotifier 定义)

(5) 与 Web 服务器进行的基本 HTTP 通信(由 javax.microedition.io.HttpConnection 定义，MIDP 扩展)

J2ME 定义了 javax.microedition.io.Connector 类，这个类包含了用于创建所有连接对象的各个静态方法。这一任务是通过根据平台名称和所请求连接的协议动态地查找一个类来完成的。在 URL 处理中，Connector.open()用来打开 URL。Connector.open()方法的字符串(string)参数是一个有效的 URL。URL 字符串由于通信协议的不同而不同。无论使用哪一种类型 URL，调用 Connector.open()都会打开一个从 Connection 到 java.io.InputStream 的字节输入流。

3.2.2 序列化和反序列化

在 J2SE 中，可以通过 ObjectInputStream 和 ObjectOutputStream 的帮助，轻松实现对象的序列化。只要定义的类实现了 java.io.Serializable 接口，就可以利用 ObjectOutputStream 中的 writeObject()方法将一个对象序列化；利用 ObjectInputStream 中的 readObject()方法，可以返回读出的对象。Serializable 接口不需要实现任何方法。

在 CLDC 中定义的 Java IO 也提供了足够的类来完成相应的 IO 操作。主要使用的类是 ByteArrayInputStream、ByteArrayOutputStream、DataInputStream 和 DataOutputStream。

3.3.3 事件处理

在μAglet 中，用在 Java 线程中定时查询的方法检查事件的发生，只能处理同步事件。

3.3 μAglet 与 Aglet 的对比

与 Aglet 系统相比，μAglet 原型系统做了大量的裁减工作。同时为了适应 J2ME CLDC 提供的 API，做了一些修改。表 3.1 列出了二者之间的主要区别。

表 3.1 Aglet 与 μAglet 的对比

Tab. 3.1 Contrast between Aglet and μAglet

	Aglet	μAglet
对象迁移协议	采用 ATP 协议或 Java RMI 进行 aglet 的迁移，使用标准 Java 平台的 java.net 包提供的通信机制。	利用 CLDC 定义的 GCF 网络框架，实现更为底层的通信机制。
对象移动方式	利用标准平台的 Java2 中提供的序列化和反序列化方法，将对象序列化后通过通信层传输。	通过 ByteArrayInputStream、ByteArrayOutputStream、DataInputStream 和 DataOutputStream 将对象序列化后通过通信层传输。
操作接口	图形化管理工具 Tahiti 或命令行方式	只支持命令行方式
消息类型	Now-type、Future-type、Oneway-type、Delegation-type	Now-type、Oneway-type
模式支持	会议模式、消息员模式、网络工具模式、信使模式、从模式	不支持
安全机制	主要通过公共密钥进行域认证和 aglet 对象认证，还有一部分完整性检查机制。	不支持
路线和任务种类	AlternateItinerary、MeetingsItinerary、MessengerItinerary、SeqPlanItinerary、SimpleItinerary、SlaveItinerary 以及 Task 与 MeetingTask。	仅支持 SimpleItinerary 和 Task。
异常处理	比较完善	只定义了几个最重要的异常并做了处理
事件处理	利用标准平台的 Java2 中提供的事件处理方式，通过继承 java.util.EventListener 和 java.util.EventObject 实现事件机制。支持异步事件。	利用 Java 代码级的轮询，在线程中定时查询事件是否发生，并产生相应线程进行处理。不支持异步事件。

4 一种嵌入式移动代理安全策略

4.1 概述

移动代理在嵌入式设备上的应用主要是一些小型应用，需要的处理资源和存储资源相对较少。不同的代理可以依照对环境的感知而一起协同地完成一个小的、特定的任务。为了更有效地使用资源，一个移动代理可以挂起当前的工作然后移动到一个位于固定网络的移动代理平台上，这样不仅可以降低对嵌入式设备网络连接的昂贵的开销，而且可以充分利用固定网络上的计算资源。一旦移动代理迁移到固定网络平台，嵌入式设备的网络连接可以暂时断开。当连接重新建立起来后，代理就可以携带需要的结果返回到嵌入式设备的移动代理平台上。

嵌入式设备相对于PC机或者服务器来说，其存储资源和计算能力都有限得多。这就使得一些复杂应用无法在嵌入式设备上执行，因此必须要将复杂的应用程序迁移到计算能力较强的平台上运行，即需要一个第三方的服务器。

移动代理的巡游分为“单跳”和“多跳”两种，前者跟C/S模式很像，即每次访问目的节点后都要返回原始主机；后者是指一个移动代理巡游所有目标主机后再返回原始主机。从效率上看，多跳无疑更高，但是由于嵌入式移动代理必须要经过第三方平台，因此嵌入式设备之间的移动代理巡游不得不采用单跳模式。而嵌入式设备与固定网络节点间的移动代理巡游则采用多跳方式。

移动代理的安全策略应当是可以动态改变的。网络服务日益多样化，不同的业务对于不同场合、不同用户有不同的安全需求，使用单一的安全机制无法适应多业务的需求。而安全等级化满足了用户对安全差异化的需求。

目前尚未有安全问题的严格分级标准，由于本课题主要是研究安全策略，因此为了简化问题，将安全等级分为I和II两个等级，区分的标准就是：

(1) 安全等级I：适用在嵌入式设备之间的移动代理活动。

(2) 安全等级II：适用在嵌入式设备与固定的PC机之间的移动代理活动。

4.2 第三方可信赖平台及域名服务器

嵌入式设备上无法运行复杂的计算，因此要派出移动代理将计算迁移到第三方平台上进行，这就要求第三方平台有强大的计算能力。另外，出于安全需要，第三方平台还要是可以信赖的，可信赖的意思就是不会被攻击。

整个由移动代理平台构成的网络分成若干个区域，每一个区域有一个可信赖的第三方平台，分区域的目的是为了使嵌入式移动代理移动到最近的第三方平台。每一次嵌入式设备联网时会自动寻找距离最近的第三方平台。

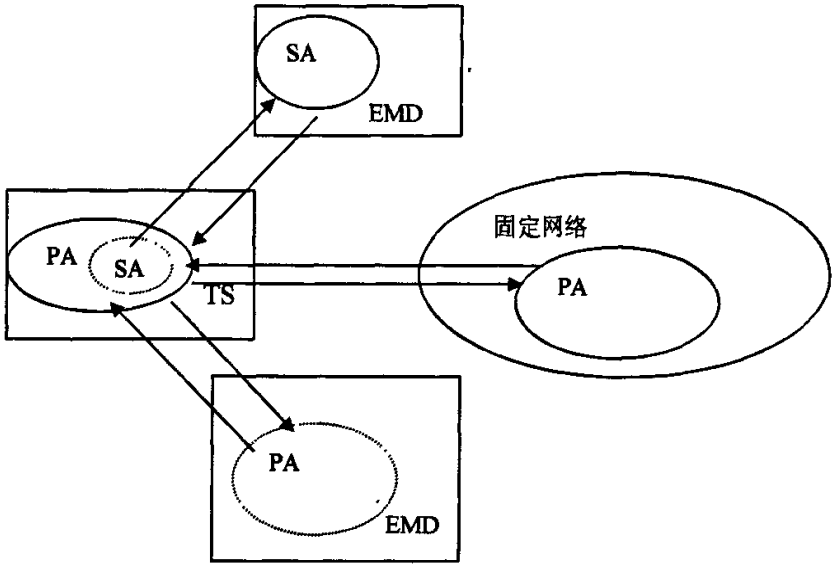

图 4.1 整体应用框架图

Fig.4.1 Overall application framework plan

EMD(Embedded Mobile Device)是嵌入式设备，*TS*(Third Server)是第三方服务器。

PA(Proxy Agent)是嵌入式设备发送到第三方上的移动代理，在嵌入式设备断网期间，它负责代表用户处理各种请求、收集信息或者进行计算，并在用户重新连接后，将结果返回给用户。

PA 可以派出子代理 *SA*(Sub Agent)来协助完成任务。*SA* 的功能通常与具体的应用有关，*PA* 以异步方式发送 *SA*，若 *SA* 没有结果返回，*PA* 通知用户，或是按照应用的要求再创建一个 *SA* 发送出去。

我们还需要一个域名服务器，因为在嵌入式设备的无线接入网络方式中，*IP* 地址是动态改变的，因此不能以 *IP* 作为移动代理的巡游路线标识，域名可以很好的解决这个问题，通过一张反映域名与 *IP* 地址的对应关系的表，可以将域名转化为实际的 *IP* 地址。

4.3 移动代理的加密及传输

对于传输中的移动代理的保护，我们使用密码学技术，即加密移动代理。

4.3.1 密码学理论概述

(1) 对称密钥密码体制

密码技术的基本思想是伪装信息，使得未被授权者不能理解信息的真实含义。伪装前的原始信息称为明文。伪装后的信息称为密文。伪装的过程称为加密。加密算法就是在加密密钥的控制下进行用于对数据加密的一组数学变换。发送方将明文数据加密为密文，授权方在接收方收到密文后施行与加密变换相逆的变换，将密文恢复成为明文，这一过程称为解密。解密在解密密钥的控制下进行，用于解密的一组数学变换称为解密算法。因为数据以密文的形式传输即使数据被非法窃取或因系统故障而泄露，未授权者也不能理解它的真正含义，从而达到数据保密的目的。同样未授权者也不能伪造合理的密文因而不能篡改数据，从而达到确保数据真实性的目的。

(2) 非对称密钥密码体制

公开密钥密码体制是几千年来在加密领域中第一次出现的真正的革命性进展，公开密钥算法是基于数学函数的而不是基于简单的比特位操作。但是更重要的是公开密钥的密码技术是非对称的，它涉及到两个独立的密钥的使用。公钥体制与传统密码体制不同，从加密密钥求解解密密钥是非常困难的。

因此加密密钥可以公开登记在网络的密钥数据库中，任何人要与某用户通信只要在密钥数据库中查得其加密密钥，用之加密的明文，只有该用户才能对得到的密文解密。在公开密钥密码体制中比较常用的是RSA。

(3) 散列函数和消息摘要

散列函数是将任意有限长度的二进制串映射为固定长度的二进制串的函数。这个固定的二进制串叫做散列值。散列函数应该是容易计算的同时也是公开的，处理过程不用保密。密码散列函数的安全性基于它的单向性。密码散列函数在现代密码学上有着广泛的应用。在签名方案中，散列函数可以将任意长度的二进制串映射为签名算法所要求的长度。此外单向散列函数还可以用在需要提供预先承诺的密码协议中。在实现的散列处理算法中应用最多的是MD5和SHA-1。MD5是RonRivest设计的系列散列算法中的第5版。它用复杂的方法对位流进行分段组合使得输入的每一位都对输出产生影响。简单地讲MD5算法的输入为任意长度的消息，产生输出为128位的摘要值。MD5算法在32位机器上运行很快而且算法实现不需要大的替换表，因而代码紧凑，在现实情况下的安全性也很高。另一个应用较广的算法是由National Institute of Standards and Technology(USA)提出的SHA-1。它与MD5算法不同之处在于生成的摘要值是160位的，实现过程中也不需要替换表，但对每个输入块都要计算80次。虽然SHA-1的摘要值仅比MD值多32位，它的安全性也因此提高了232倍。

(4) 数字签名和数字证书

数字签名用来证明消息的发送者是谁，并且发送者不能抵赖对消息的签名。消息的接收者可以识别和验证消息确实源自该发送者，但是不能伪造发送者对消息的签名。数字签名一个消息并不修改原始的消息只是简单地生成一个附加的数字。签名信息可以捆绑在消息上一起发送也可以分开发送。数字签名一般基于公钥加密密码体制和消息摘要算法。消息发送者首先应用消息摘要算法计算其消息的摘要值，然后用自己的私钥加密该消息摘要得到消息的数字签名消息。接收者使用相应的消息摘要算法对接收到的消息计算摘要后，使用接收到的数字签名和消息发送者的公钥来验证该消息摘要。如果这个摘要值与数字签名匹配，那么这将确认消息原文是发送者签名的并且消息在传送过程没有被改动。由于发送者使用自己的私钥进行签名所以事后不能抵赖对该消息数字签名，因为只有他拥有该私钥并可以生成该签名信息。接收者也不能伪造发送者的签名，因为发送者的私钥并不公开，接收者不能得到该私钥。目前较为常用的数字签名算法组合有MD5+RSA、SHA+RSA等。消息接收者需要得到发送者的公钥才能验证发送者对消息的数字签名。目前往往用数字证书(Certificate)的方式来公布，公钥由数字证书中心(CA)来管理和发布。数字证书通常包括用户信息、公开密钥证书有效期等内容。每个用户的私钥由用户自己保存，同时用户往往还保存自己的数字证书和CA的公钥以便和CA联系获得其他用户的公钥数字证书。

4.3.2 移动代理传输安全

(1) 认证过程

本过程有两个步骤：主机间相互认证以及移动代理传输加密。

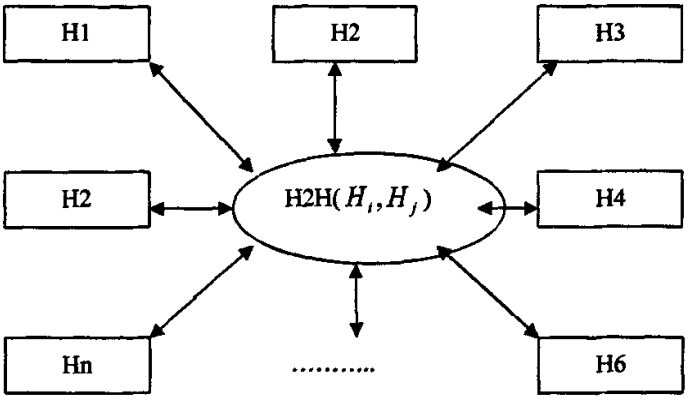

图 4.2 主机间相互认证

Fig. 4.2 Host mutual authentication

图 4.2 描述了主机间相互认证的形式。也就是从理论上说在整个移动代理网中的主机都需要互相认证。

但是由于嵌入式设备间的移动代理的运行是在第三方平台上运行的，因此嵌入式设备不需要与其他主机相互认证，而只需要和第三方平台相互认证即可。这样一来，嵌入式设备进行认证的次数大大减小。如图 4.3 所示。

嵌入式设备联网时会自动寻找最近的第三方平台，然后再检查两者是否已经验证过，如果没有则开始验证，并且将嵌入式设备原先已有的认证信息覆盖掉。

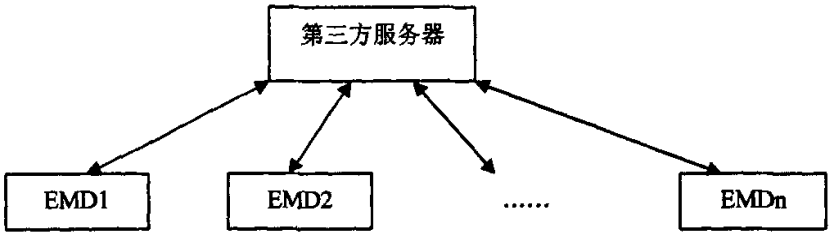

图 4. 3 嵌入式设备与第三方可信赖服务器的认证

Fig. 4.3 Authentication between embedded mobile equipment and credible third-party server

图 4.4 描述了两台主机间认证的具体过程。

任意两台机器都需要有一个对称密钥 K_{H1_H2}，这个密钥的分配是由密钥分配服务器来分配的。

主机*H1*产生一个随机数据 N_{H1}，然后用*H1*和*H2*共有的对称密钥 K_{H1_H2} 加密 N_{H1} 得到 $\sim N_{H1}$，再将 $\sim N_{H1}$ 传送到主机*H2*，主机*H2*用 K_{H1_H2} 解密 $\sim N_{H1}$ 得到 N_{H1}，然后它再生成一个随机数据 N_{H1}，并且用 K_{H1_H2} 将 N_{H1} 和 N_{H2} 一起加密，传送回主机*H1*，主机解密后比较原始 N_{H1} 和主机*H2*传送过来的 N_{H1}，如果不同，则认证失败，表明主机*H2*是不可信赖的，不能加入到移动代理平台网中。如果相同，则主机*H1*加密 N_{H2}，然后传回主机*H2*，*H2*解密后也进行原始 N_{H2} 和接受到的 N_{H2} 的比较，不同则认证失败，相同则认证成功。

认证结束后，两主机便用同一个hash函数产成一个共享密钥*SC*。在后面，共享密钥*SC*和 N_{H1} 及 N_{H2} 就用来对移动代理数据进行加密和解密。

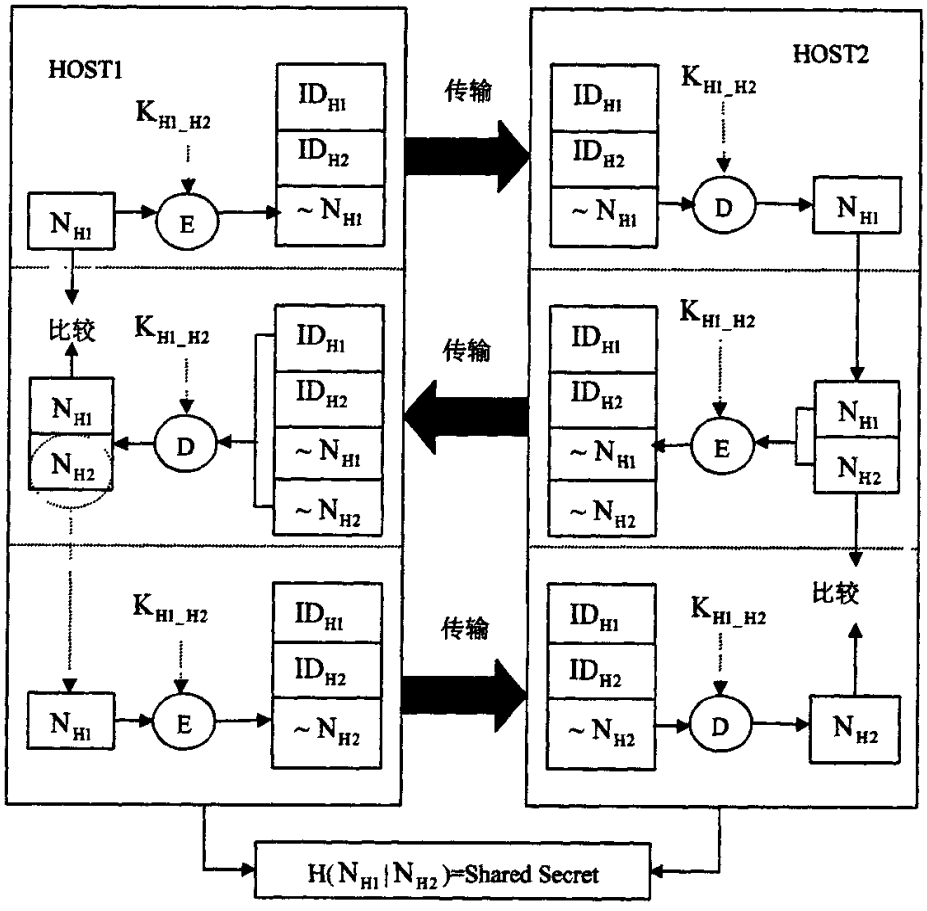

图 4.4 两主机的认证过程

Fig.4.4 Authentication between two hosts

K_{H1_H2} 是一个对称密钥；

N_{H1} 和 N_{H2} 是 *H1* 和 *H2* 分别产生的随机数；

$\sim N_{H1}$ 和 $\sim N_{H2}$ 是 N_{H1} 和 N_{H2} 的加密值；

ID_{H1} 和 ID_{H2} 是 *H1* 和 *H2* 的标识。

(2) 加密移动代理

将移动代理要完成的任务以及所涉及到的所有数据封装成一个类，这样这个类的实例与加密整个移动代理是等效的，但是从编程的角度来说，前者更有效率更容易实现，同时也减少了执行加解密算法所需要的时间开销。

传统的对称加密在运算速度上比非对称加密要快出100-1000倍，因此对于嵌入式设

备这种资源受限的设施来说，是个合适的选择。

DES(Data Encryption Standard)满足了国家标准局欲达到的 4 个目的：提供高质量的数据保护，防止数据未经授权的泄露和未被察觉的修改；具有相当高的复杂性，使得破译的开销超过可能获得的利益，同时又要便于理解和掌握。

DES 算法把 64 位的明文输入块变为 64 位的密文输出块，它所使用的密钥也是 64 位，首先，DES 把输入的 64 位数据块按位重新组合，并把输出分为 *L0*、*R0* 两部分，每部分各长 32 位，并进行前后置换(输入的第 58 位换到第一位，第 50 位换到第 2 位，依此类推，最后一位是原来的第 7 位)，最终由 *L0* 输出左 32 位，*R0* 输出右 32 位，根据这个法则经过 16 次迭代运算后，得到 *L16*、*R16*，将此作为输入，进行与初始置换相反的逆置换，即得到密文输出。

DES 算法的入口参数有三个：*Key*、*Data*、*Mode*。其中 *Key* 为 8 个字节共 64 位，是 DES 算法的工作密钥；*Data* 也为 8 个字节 64 位，是要被加密或被解密的数据；*Mode* 为 DES 的工作方式，有两种：加密或解密，如果 *Mode* 为加密，则用 *Key* 去把数据 *Data* 进行加密，生成 *Data* 的密码形式作为 DES 的输出结果；如 *Mode* 为解密，则用 *Key* 去把密码形式的数据 *Data* 解密，还原为 *Data* 的明码形式作为 DES 的输出结果。在使用 DES 时，双方预先约定使用的密码即 *Key*，然后用 *Key* 去加密数据；接收方得到密文后使用同样的 *Key* 解密得到原数据，这样便实现了安全性较高的数据传输。

(3) 移动代理的完整性验证

DES 这种传统的加密算法有一个缺点就是无法保证密文的完整性，因此需要增加一个完整性的校验。图 4.5 描述了验证移动代理完整性过程。

图 4.5 中 *HA* 的 *MAC* 是 HMAC 加密函数产生的，它将与移动代理一同发送到主机 *HB*。主机 *HB* 接受到代理后，用与主机 *HA* 相同的加密函数产生一个 *MAC*，其中主机 *HB* 将本地的公共密钥以及 N_{H1} 和 N_{H2} 作为输入参数。然后比较本地产生的 *MAC* 与接受到的 *MAC*，如果相同则说明在传输过程中没有受到恶意攻击，反之则表明受到攻击。

这一步骤的处理顺序如下：

① 将移动代理中封装了关键代码和数据的类作为 hash 函数的输入。这个 hash 函数会使用在认证中获得的公共密钥 *SC* 和 N_{H1}、N_{H2} 对此类进行加密，得到一个 *MAC*。

② 移动代理从主机 *HA* 迁移到主机 *HB*。

③ 在主机 *HB* 上也产生一个 *MAC*。主机 *HB* 使用跟主机 *HA* 同样的 Hash 函数，同样的输入，产生一个 *MAC*。

④ 比较两个 *MAC*。相同则没有收到攻击，否则被攻击。

(4) 安全性分析

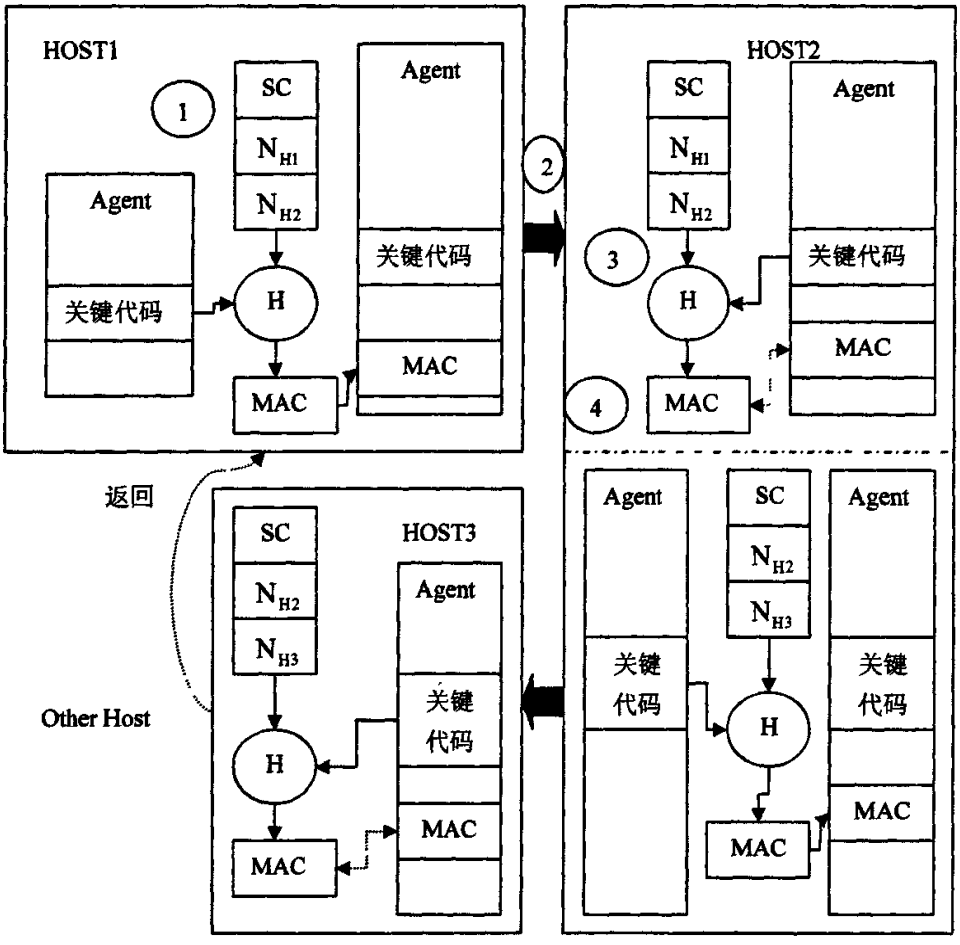

图 4.5 验证移动代理完整性

Fig. 4.5 Verify the integrity of mobile agents

任意两台主机使用的私有密钥和两个随机数是只有这两台主机才知道的，第三方不可能知道。因此，即便第三方非法截获信息，也无法推算出私有密钥，也就无法仿造出一致的响应。

另外，由于私有密钥是两台通信主机独有，因此通信双方的身份可以得到验证，即将信息发往其他主机不会被承认。

传输过程中采用了 DES 加密，因此即使报文被截获，也无法被破译。

4.4 安全等级 I 策略

由于嵌入式设备的有限性，使得某些危害很大的恶意攻击无法在嵌入式设备上实现，比如不会有人用一个手机作为电子商务的服务器，自然也就不会出现恶意窃取电子货币的情况，也不会有人用 PDA 作为电子售票系统，同样就不会出现银行卡密码被盗等情形。

但是，尽管嵌入式设备上的应用大多是小型应用，安全仍然是必须的。嵌入式设备无法运行太复杂的安全机制，因此可以把一些较为关键的处理派遣到第三方可信赖平台上运行。也就是某个嵌入式设备派出移动代理到第三方平台，由第三方平台根据巡游路线逐一派遣子代理，到目的嵌入式设备上去，并获取要使用的数据后返回到第三方平台，由其负责处理这些数据。如图 4.6 所示。

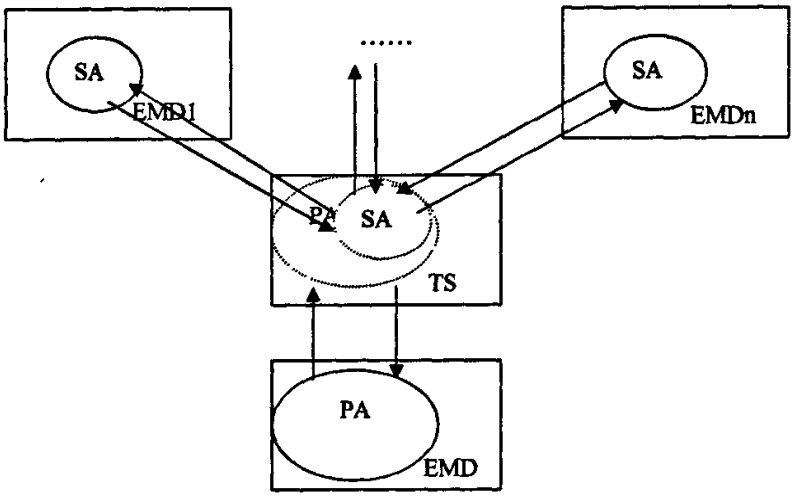

图 4.6 安全等级 I 策略的整体描述

Fig. 4.6 Describe the overall of security level I strategy

为了增强子代理 *SA* 的安全性，在第三方平台上增加一个测时功能。也就是通过代理在嵌入式设备上执行的时间与规定的时间的比较来检验代理是否收到过攻击。

首先第三方平台有一个代理 *SA* 派送出去的时间 T_{send}，以及 *SA* 返回的时间 T_{back}，还有一个网络延迟时间 T_{delay}。这个网络延迟时间是一个参考值，它应该能根据网络状态动态变化，也就是每次使用它的时候它的值可能会不相同。

第三方平台还有一个变量 T_{out}，即允许代理外派的最大时间(加解密时间+执行时间)，由于代码的执行速度与平台的硬件相关，因此没有一个数学公式能确定一个代码

在任意平台上被破解所需的最短时间，这也正是限时法无法在实践中获得应用的根本原因，因此我们所定义的T_{out}是一个经验值。

一个正常的没有受到过攻击的 SA 应该满足$T_{back}-T_{send}\le T_{out}+T_{delay}$。如果外派时间大于$T_{out}$甚至超过了$T_{out}+T_{delay}$，那么它所访问的目的平台就是可疑主机。

4.5 安全等级 II 策略

4.5.1 策略概述

嵌入式设备与固定网络节点之间的移动代理的安全问题其实就等同于普通的移动代理安全问题，因为从嵌入式设备发出的移动代理是要移动到第三方可信赖平台上，由第三方平台接管并负责的，这就将嵌入式设备从问题里剔除掉了，问题便转化为移动代理的安全问题了。

移动代理的传输安全问题及解决方案在前面已经有阐述，这里就不赘述了。这里主要讨论的是如何保护移动代理本身。

由于我们的主要目的是保护移动代理，所以我们将集中定义几种对移动代理的主要攻击。对这些攻击的防范或者检测是评估安全策略的标准。

(1) 伪装。一个平台能把自己伪装成另一个平台，或者将代理发送到另外的目的地而不是这个代理想去的目的地。

(2) 拒绝服务。平台能忽略移动代理的服务请求，对关键任务产生不可接受的延迟，拒绝代理的执行或者没有任何通知的情况下简单地结束代理。

(3) 窃听。在代理移动过程中，它暴露在平台或其他实体前，这些实体能非法接触到关键信息或者侦听到代理间的通信。

(4) 修改。当一个代理到达一个平台后，它的代码、状态、数据都将暴露在平台面前，使得后者可以修改前者。

预防机制试图使得恶意主机不能或者很难修改代理的代码或者状态。预防的等级可以强也可以弱。一个弱预防的例子就是限时黑箱，但是它缺少一个可以在实际中应用的算法。强预防的例子有使用修改检验设备，也就是移动代理是在一个物理上封闭的环境中运行的。这对大规模的应用来说是一个非常昂贵的方法。

检测机制主要目标是检测非法的代码、状态和移动代理执行流的修改。检测同样可以有强有弱。执行跟踪是一个弱检测的例子，因为跟踪的大小和数量会随着所访问的主机的增加而增加，而检测过程却只在代理返回母机时执行。

在我们的方法中，我们想使用一种机制能克服这些缺点并且能及时探测到几乎所有的攻击，因此是一个强探测机制。因此，我们调整了引用执行的概念取而代之使用移动代理的复制体并且假设存在一个可信赖的服务器[40]，后面将有详细的论述。

使用复制体概念，我们的协议会确保一个移动代理不被恶意主机攻击，为了达到这个目的，它的复制体会在可信赖服务器上运行，这样做的目的是为了验证它的执行是否正确。

为了优化代理和其复制体之间的通信量，并且两者都包含在网络的一部分中，我们假定网络被分成几个域，每个域都有一个可信赖服务器，这个假想是建立于正在研究的一种商业模型，这个模型把网络看成是许多电子市场，而模型本身则看成电子商店。

在这个协议中主要有 5 种实体：源平台(P_0)，目的平台(P_{ij})，可信赖服务器(TS_j)，移动代理(MA)，以及它的复制体(CA)。下面简单地描述一下：

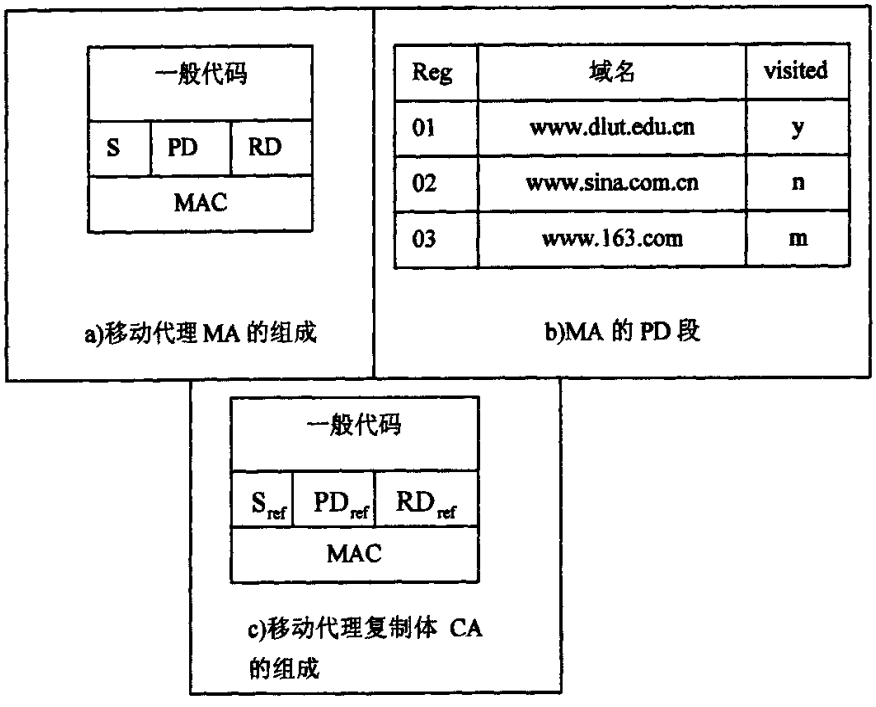

图 4.7 移动代理及其克隆体的构成

Fig. 4.7 The constitute of MA and CA

源平台用 P_0 表示，它负责创建移动代理(MA)，并供给给移动代理一个初始化的路线。随后它会产生一个代理的复制体(CA)，复制体跟本体除了 ID 不同外完全相同。最后，

MA 移动到路线中的第一个平台 P_{11}，而 *CA* 则移动到相应区域的可信赖服务器上(TS_1)，*MA* 会接着从平台 P_{ij} 移动到另一个平台去搜索用户需要的信息，而 *CA* 只有在 *MA* 跨区域移动时才会随之迁移到相应区域的可信赖服务器上。最后 *MA* 和 *CA* 携带结果返回源平台进行最后的比较。

在我们的协议中，移动代理是由三部分组成的：*MAC*、关键 agent 部分及一般 agent 部分。如图 4.7 所示

MAC 部分已经在前面的章节有所描述，不再赘述。

MA 的关键 agent 部分包含状态 *S*，它包含了移动代理在下一个平台上从迁移点处继续运行所需要的信息。关键 agent 部分还包含数据，它是由协议数据 *PD* 和结果数据 *RD* 组成的。*PD* 部分包含了路线信息，比如以前被提到的平台，每一个平台所属的区域和其状态(访问、未访问、危害)。这部分最初状态是包含一定数量的平台，这些平台是源主机所提议的，随后此部分会随着访问的平台而升级。

克隆体 *CA* 是 *MA* 的一个拷贝，所以它与 *MA* 有相同的组成部分和状态。唯一不同的是 *CA* 只在可信赖服务器上运行，因此它所携带的信息被认为是权威参考，这些信息便被加上 *ref* 的下标。*CA* 被用于通过并行运行来校验 *MA* 的执行。

目的平台 P_{ij} 是 *MA* 为了执行用户的任务而要访问的移动代理平台。每一个平台上都有一个固定代理负责与 *MA* 或者其他的移动代理进行交互。

可信赖服务器 TS_j 是能给移动代理提供执行服务并且被用户所信任的平台。前面提到，每一个域内都会有一个这样的平台，它们的任务就是使用从 P_{ij} 发过来的信息执行 *CA*，从而校验 *MA* 的执行。TS_j 同样也有一个固定服务代理负责执行 *CA* 以及和其他的移动代理进行交互。

4.5.2 策略详解

本部分会提供详细的步骤，如图 4.8 所示。

首先，母平台 P_0 的固定代理会创建一个 *MA*，然后克隆体 *CA* 产生，*MA* 迁移到第一个目的平台 P_1，而 *CA* 迁移到 P_1 所在域的可信赖服务器 TS_1。

接下来，*MA* 会按照它的巡游路线，从一个平台 P_{ij} 迁移到另一个目的平台，而 *CA* 只有在跨域的时候才迁移到相应的区域的可信赖服务器上。*MA* 和 *CA* 的执行如图 4.8 所示。

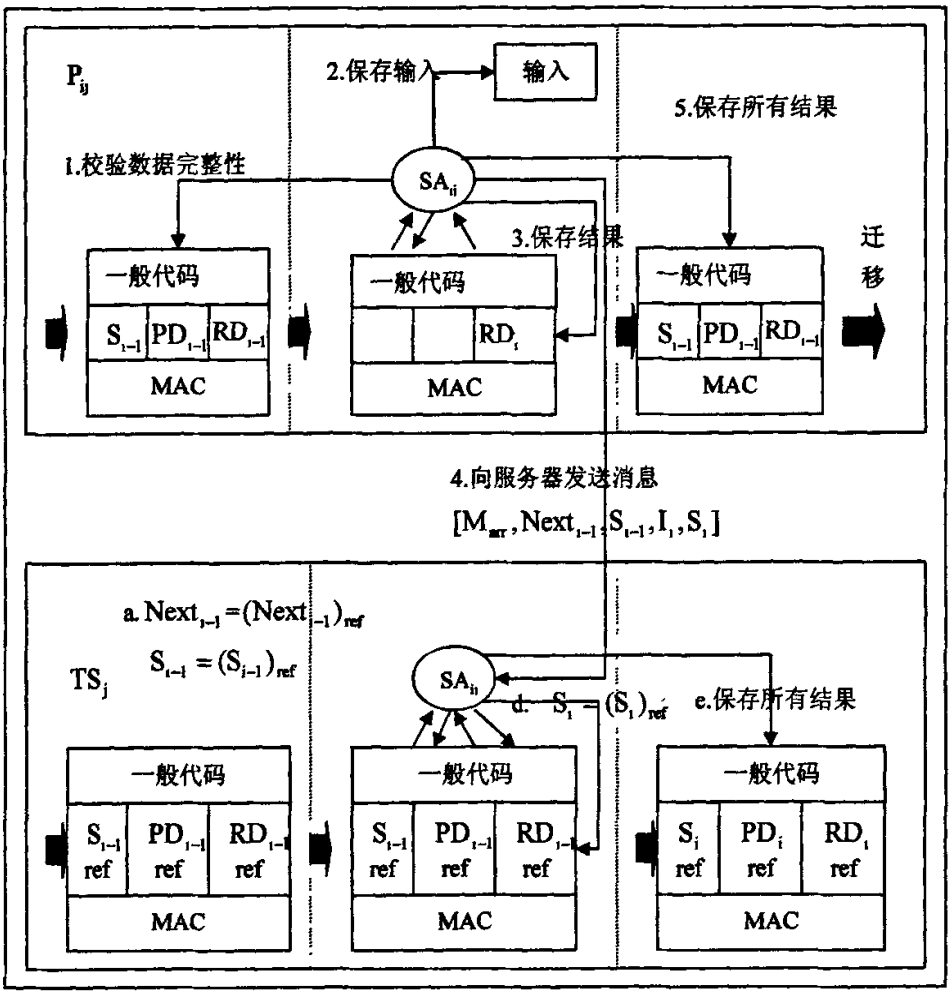

图 4.8 安全协议的详细步骤

Fig. 4.8 Detaled steps of security protocol

当 *MA* 到达目的平台 P_{ij} 并解密后，平台上的固定代理 SA_{ij} 会检验 *MA* 的合法性，然后再通过 *MAC* 检验代码的完整性，即代理在传输过程中是否收到攻击。如果校验通过，则进入到执行步骤。

在执行期，SA_{ij} 会保存所有要用到的输入信息，比如和巡游路线相关的信息 Ih_i。执行结果会保存在*MA*的 RD_i 区。

*MA*会将信息 $[M_{arr}, Next_{i-1}, S_{i-1}, I_i, S_i]$ 送到相应的可信赖服务器上。M_{arr} 是表征*MA*已经到达 P_{ij}，并让验证结果。S_{i-1} 是前一个平台执行后的状态。I_i 是执行*MA*所需要的信息，

而S_i是指MA在P_{ij}上的执行后的状态。

最后，如果 CA 的校验通过，那么巡游路线信息和最新的结果信息都会保存在 MA 中并且迁移到下一个目的平台。

当可信赖服务器上的固定代理SA_j接受到 CA 后，它首先检验 MA 是否是从正确平台迁移过来的，这个是通过校验$Next_{i-1}$和$(Next_{i-1})_{ref}$是否等于P_{ij}作为判断依据。然后，SA_j会验证S_{i-1}和$(S_{i-1})_{ref}$是否相等，如果相等说明在前一个平台上 MA 的执行是正确的。接下来，CA 会使用接受到的输入信息I_i执行，执行结果将保存在 CA 中。在 CA 执行后，SA_j会检验 MA 在P_{ij}上的执行是否正确，这个是通过验证S_i和$(S_i)_{ref}$是否相等来判断的。最后，如果 MA 的下一个迁移地点跟P_{ij}不在同一个域，则 CA 要迁移到相应域的可信赖服务器上区，否则，它将重复上述过程。

4.5.3 安全分析

这部分主要是分析这个安全策略如何探测恶意攻击。因此我们会对前面已经给出的几种攻击进行逐一分析。

(1) 伪装检测：一个平台P_{ij}无法伪装成另一个平台，因为它没有另一个平台与目的主机的共享私有密钥，因此代理无法正确加密，目的平台便无法正确解密接受到的代理信息，这样P_{ij}便会被指认为恶意主机。如果 MA 在执行后迁移到它不应该去的平台，TS_j同样能根据来自P_{ij}的$Next_{i-1}$信息中的判断出来。

(2) 拒绝服务。如果平台拒绝让移动代理运行，则可信赖服务器就无法接受到来自P_{ij}的信息，自然无法执行 CA，也就不可能通过探测了。如果P_{ij}简单地结束了代理，同样会被TS_j检测出来，因为后者是了解 MA 的行程的。

(3) 窃听。一个平台不能窃取或者接触其他平台上收集到的数据，因为每一个平台都对数据进行了加密。任何修改都会在移动代理返回母平台时被探测出来，因为 CA 和 MA 的数据不一致。

(4) 修改。如果平台P_{ij}只修改了关键代码部分而没有修改 MAC，则迁移到下一个平台时就会因为 MAC 的不匹配而被探测出来。代理执行后，如果P_{ij}在发送给 CA 的消息中加入错误信息，比如如果它发送错误的S_{i-1}，那么恶意主机不是$P_{(i-1)j}$就是P_{ij}，这一定会被TS_j检测出来的。如果P_{ij}发送错误的I_i和正确的结果状态，TS_j也会检测出来，因为 CA 用错误的I_i是不会产生正确的结果状态的。如果P_{ij}发送错误的I_i以及根据错误I_i

执行出来的相对正确结果状态(实际上还是错误的)，虽然在TS_j上不能检测出来，但是当迁移到下一平台时，*MA* 会将上个平台的结果状态发送给TS_j进行校验，这时就可以探测出P_{ij}是恶意主机。

4.6 应用情景描述

4.6.1 情景 1

某外出旅行的用户希望能及时了解股市行情，并在某支股票到达期望值时进行买卖交易。用户随身携带的 PDA 可以通过无线上网，但是无线网络连接带宽低且费用昂贵，因此用户连接的时间比较短，大部分时间 PDA 处于断线状态。但是用户的 PDA 上安装有嵌入式移动代理系统，可以创建、接受和发送移动代理。而在用户所处区域中有强大的可信赖的第三方服务器，可以接管用户发出的移动代理，也就是 PDA 可以通过这个第三方服务器与固定网络连接。而固定网络可以与股票公司的服务器(也安装有移动代理系统)实时通信，进行网上股票交易。

用户首先在 PDA 上创建一个委托代理 PA，并告知参数(比如股票代码、价格和交易数额等)，然后将 *PA* 派送到相应的可信赖服务器上去，*PA* 就相当于一个派遣移动代理的主机，往各个目的地派遣子移动代理 *SA*，用户指定 *SA* 加载安全等级 II。*SA* 的作用就是根据用户提供的参数和规则，来实时监控股票行情，一旦符合条件，就立刻代用户进行交易。将交易结果返回给 *PA*，*PA* 会将结果缓存起来，等到用户再次联网时发送给用户，或者第三方服务器发送短信提示用户上网，然后将结果返回。

4.6.2 情景 2

某外出旅行的用户希望能及时了解家中的动态，比如各种家用电器是否停用，各种物品是否遭窃，水龙头煤气开关等是否关好。用户可以通过随身携带的 PDA 无线上网，发送移动代理到第三方服务器上，由服务器负责具体任务。第三方服务器与用户的家庭网关联系，并派出子代理收集信息(指定安全等级 I)，收集到的信息都返回到第三方服务器，由其按照用户的要求进行处理，最后将结果通知用户。

5 安全策略在实际项目中的应用

5.1 项目需求分析

东莞市有大小数千个工厂，每天的排污量十分惊人，为了保护环境，必须对工厂的排污量进行监控，对污水的成分进行监控，还要根据污水排放量进行收费。但是由于工厂数目巨大，没有时间和人力对工厂的排污进行实地检查。因此，环保局为每个工厂提供一个智能监控设备，它能够实时监控排污系统，提供环保局所需要的一切数据。在环保局监测中心，有若干台服务器，它负责接受工厂的智能监控设备发送来的数据。智能监控设备是通过短信的方式把数据发送到监测中心的服务器上的。

但是这种监控方式有两个不足：

(1) 短信延迟严重。由于工厂有上千之多，因此，短信序列也有上千条之长，一条一条接受，排在后面的短信的时效性便非常低。

(2) 只能在服务器上看到监测数据，使得工作方式死板。

因此这个监控系统需要升级更新。

5.2 解决方案

由于第一个问题的解决与本论文的主题无关，因此不加阐述。

对于第二个问题的解决，我们采用了嵌入式移动代理的方案。也就是为环保局工作人员分配一个嵌入式设备，上面安装有嵌入式移动代理系统，当工作人员需要某个工厂或某几个工厂的数据的时候，可以派送一个移动代理，让它到环保局监测中心的服务器上，与当地数据库进行交互，把数据取出然后返回到嵌入式设备上。由于这些排污数据对环保局来说很重要，因此移动代理的安全一定要得到保证，故而采用了本论文所描述的安全机制。

5.3 具体实施

5.3.1 硬件设备说明

目前该项目尚未最终完成，加入嵌入式移动代理的方案也只是处于实验室内的实验阶段。

嵌入式设备 *ED* 采用了优龙科技的 FS44B0II 型 ARM 开发板，上面安装有本实验室所开发的 μAglet 嵌入式移动代理系统，由于硬件条件所限，开发板没有采用无线上网，而是采用以太网接入方式。

用两台 PC 机作为数据服务器 *DS*，这两台服务器组成一个局域网，另有一台机器作为该域的可信赖服务器 *TS*。

另有一台 PC 机作为第三方可信赖平台 *TSP*。下位机是基于 W77e58 的单片机 *SP*，它的一个串口与一台 PC 机相接，这台 PC 机每隔 4 秒往串口发送一次数据；另一个串口则与明基的 M22 型 GPRS 模块相连实现无线上网。

整个实验的框架图如下。

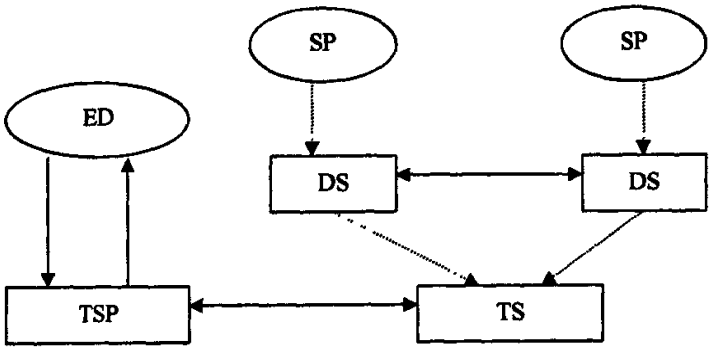

图 5.1 实验过程结构图

Fig. 5.1The course chart of the experiment

DS 负责存储 *SP* 所发送的数据信息，*TS* 是安全等级 II 中的域内可信赖服务器，是用来运行移动代理的复制体的。*TSP* 是第三方可信赖平台，它负责接管 *ED* 发送的移动代理。*ED* 和 *TSP*、*SP* 和 *DS* 都是通过 GPRS 无线网络通信，而 *TSP*、*TS* 以及 *DS* 间则是有线局域网。

5.3.2 主要软件设计说明

(1) 域名服务器

Boolean RegionRegister(URL ip,String urlname)：注册域名，建立域名与 IP 的对应关系；

Boolean UnRegionRegister(String urlname)：注销域名；

URL getIP(String urlname)：根据域名查找 IP；

String getRegion(URL ip)：根据 IP 查找所在域；

Boolean update(URL ip,String urlname)：更新域和 IP 的对应关系；

(2) 认证模块

Boolean Authentication(String urlname1,String urlname2)：两主机间的认证；

Int CreateSC(int NH1,int NH2)：生成两主机共享的私有密钥；

(3) 移动代理

Void Migrate(Vector itinerary)：迁移移动代理；

Object MessageToSA(msg message,Object arg)：与 SA 间的消息通信；

Vector ObtainResult()：从关键类中获取结果信息；

Vector RegionToIP(Vector itinerary)：将路线中主机的域名转换成 IP 地址；

(4) 安全

Void isClone(Aglets MA)：生成移动代理的复制体；

Void Decryption()：解密关键类；

Void Encryption()：加密关键类；

String CreateSHA(KeyCode obj)：生成关键类的报文摘要

(5) 固定代理

Booelan ConfirmReq(KeyCode obj)：验证移动代理的完整性；

Object QueryDB()：查找本地数据库；

Vector SendResult(Vector result)：将查询结果传递给移动代理；

Object SAToSA(msg message,Object arg)：固定代理间的消息通信；

Boolean CompareResult()：将移动代理的结果和复制体的结果进行比较；

5.3.3 系统运行

首先在各个节点启动 μAglet 系统平台的服务，uClinux 平台的 μAglet 服务程序可以通过上位机的终端程序启动，如图 5.2 所示。

后图中的 aglets.props 是一个配置文件，主要指定了服务的端口、类路径等，将参数传给 main. class 并由 console. class 负责解析。Console 程序提供了一系列可供用户操作代理实体的命令，如 Create、remove、dispatch、clone、msg、shutdown 等。

然后创建 *MA*，*MA* 会被派送到第三方可信赖平台上，由其接管。第三方可信赖平台上的固定服务代理 *SA* 会接受到 *MA*。*SA* 产生 *MA* 的复制体 *CA*，并且将 *MA* 发送到目的平台，而将 *CA* 发送到目的平台所在域的可信赖服务器上。两方面同时运行，收集信息，并比较信息的一致性。最终将数据返回，如图 5.3 所示。

由于是嵌入式设备和固定 PC 机之间的移动代理活动，所以选择安全等级 II 策略对移动代理进行保护。

图 5.4、图 5.5、图 5.6 分别是第三方可信赖平台、数据服务器、域内可信赖服务器的工作过程显示画面。

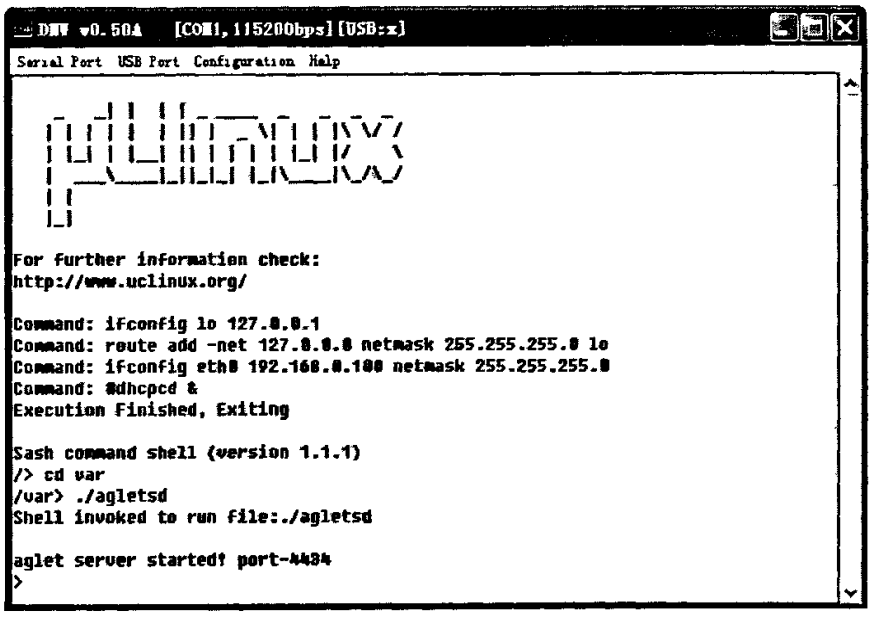

图 5.2 启动 μAglet 平台系统

Fig. 5.2 Laungh μAglet system

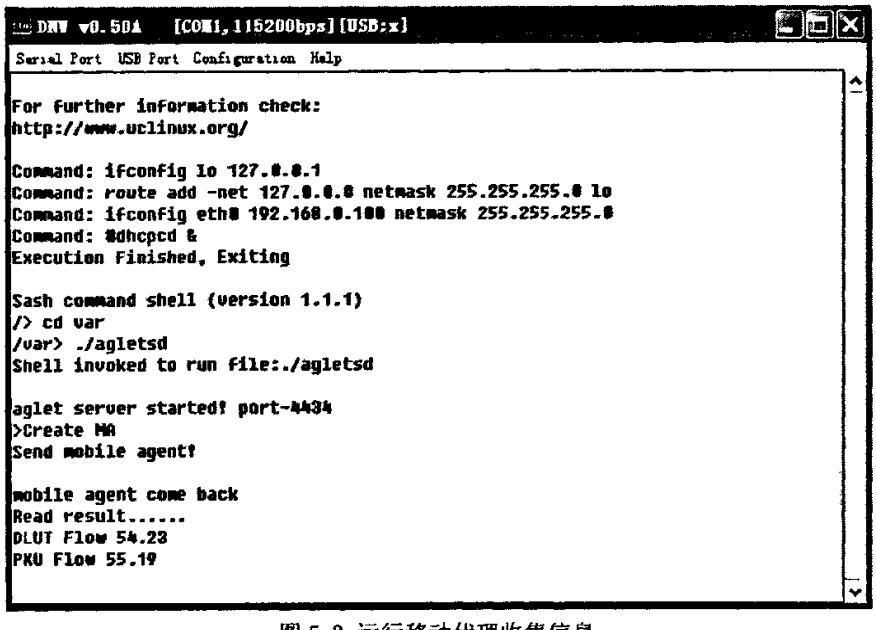

图 5.3 运行移动代理收集信息

Fig. 5.3 Collect information by running mobile agent

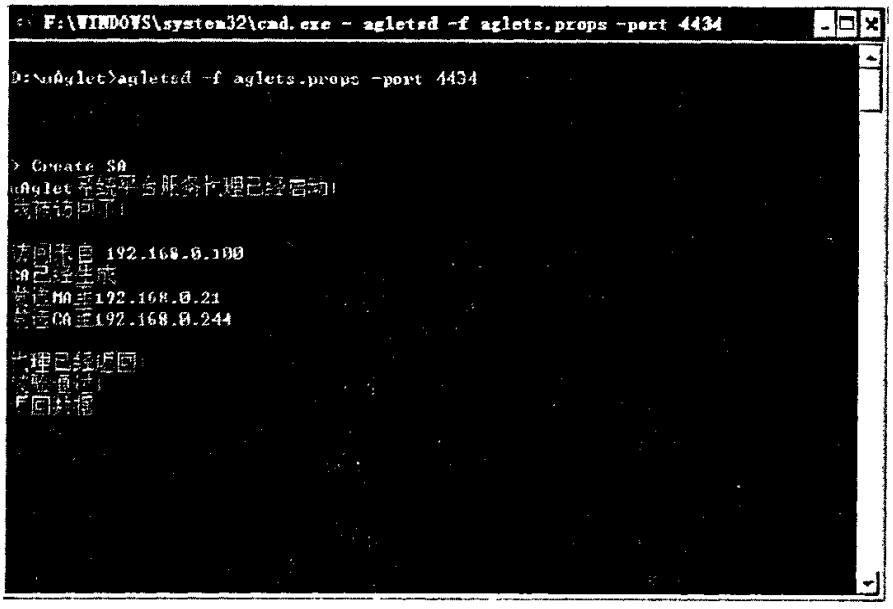

图 5.4 第三方可信赖服务器

Fig. 5.4 The third trust server

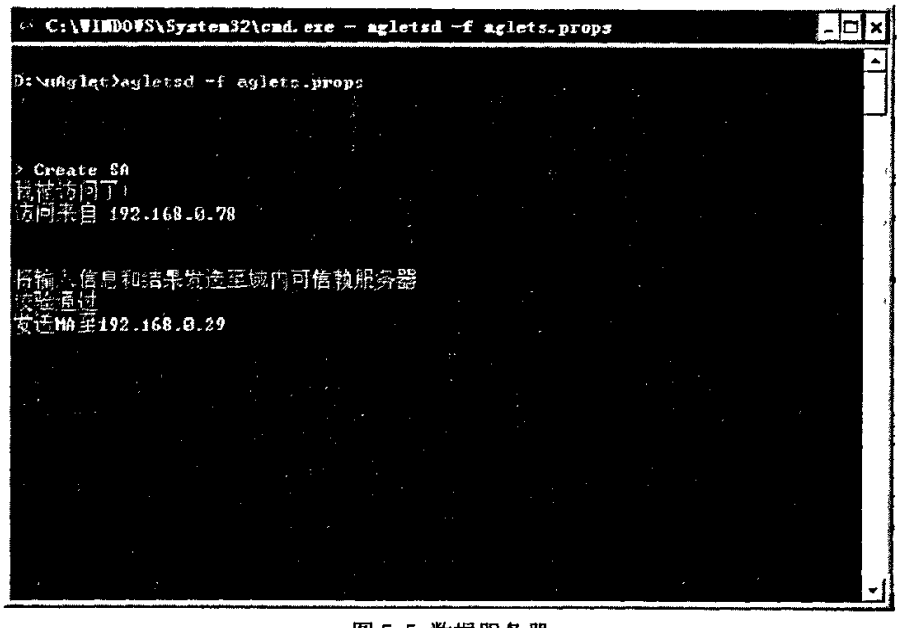

图 5.5 数据服务器

Fig. 5.5 The data server

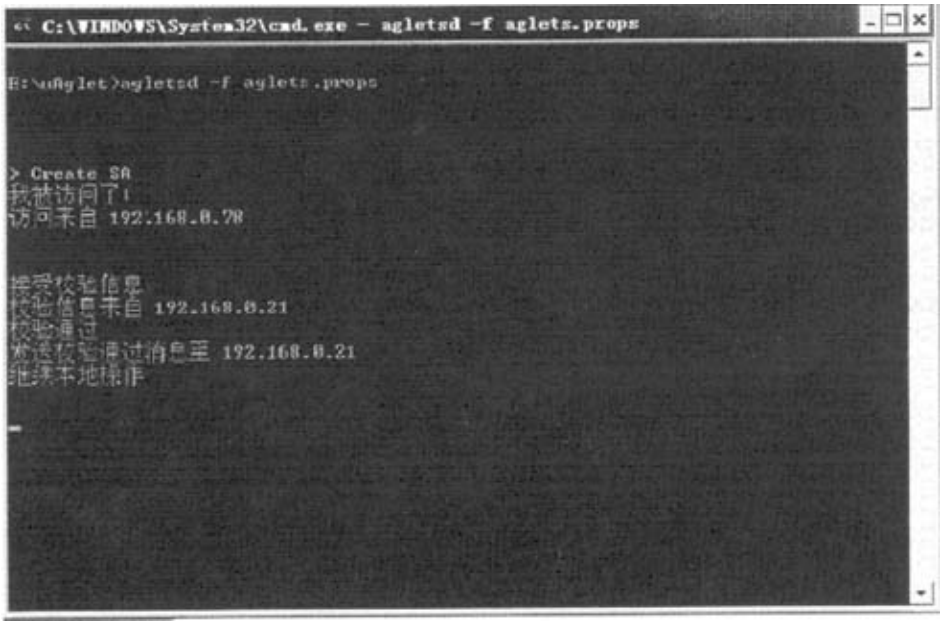

图 5.6 域内可信赖服务器

Fig. 5.6 The trust server in region

5.3.4 测试结果

主要是对可能发生的窃听、伪造、中断攻击进行了测试。

(1) 使用一台 PC 机伪装成正常的执行主机(比如修改成同样的 IP 地址)，虽然截获了移动代理，但是由于没有对称密钥，无法解密密文，窃听攻击测试通过。

(2) 将代理派送到不在巡游路线的主机上(模仿伪造攻击)，结果由于巡游路线和复制体的路线不同而被检测出来。

(3) 接受到代理后私自将代理停止。结果由于复制体无法接受到输入信息而被检测出来。

5.4 性能分析

安全策略提高了嵌入式移动代理的安全性，但是同时也增加了系统开销。本文就模拟东莞环保局项目的实验的实际运行结果进行分析。

实验实现了两种情况：第一个是移动代理在没有任何安全机制的情况下运行；第二种情况是移动代理采用安全等级 II 策略运行。通过对比这两种情况的流量、运行时间，可以看出该安全策略的性能。

5.4.1 流量分析

流量包括移动代理自己产生的流量、复制体产生的流量以及代理和复制体之间的通信流量。

流量统计是通过对每一台参与实验的设备的 4434 端口进行监控得到的。4434 端口是 μAglet 系统启动时默认的占用端口。

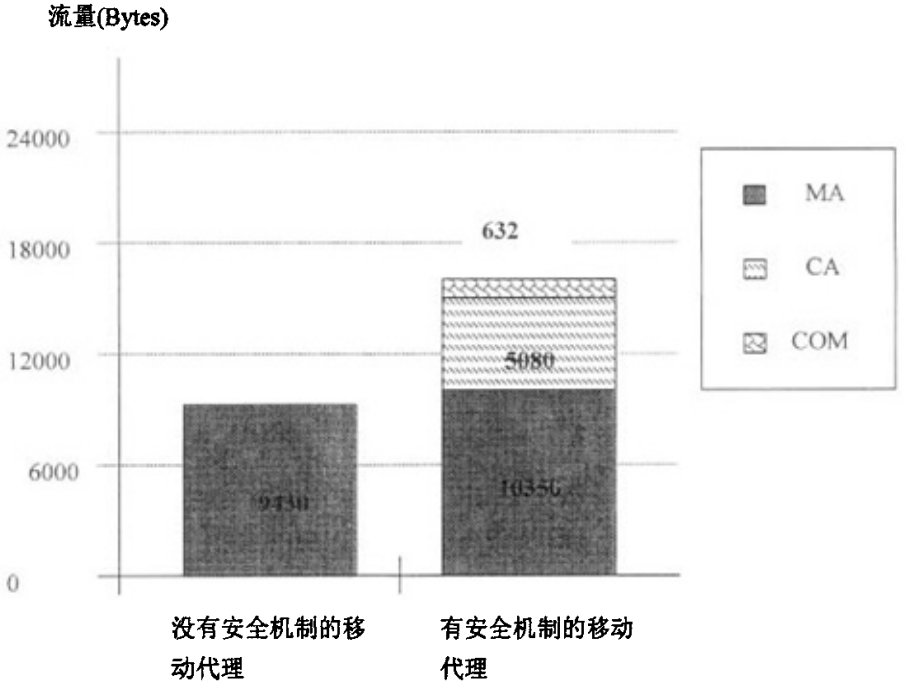

图 5.7 两种情况下的流量对比

Fig. 5.7 Contrast of flow in two cases

图 5.7 表明没有安全机制的移动代理的流量完全是由移动代理自己产生的。而有安全机制的移动代理运行时，复制体产生的流量大概是移动代理的一半，这是因为在试验中移动代理从第三方可信赖平台出发至返回迁移了三次，而复制体只迁移了两次。从数据上来说，复制体带来的流量负担并不大。

在最坏情况，也就是一个域内只有一台目的主机和一台可信赖服务器，那么就相当于复制体和移动代理的迁移次数是相同的，那么复制体造成的流量与移动代理的基本相当。

由此可见如何划分域将对整个流量大小有很大的影响。

由图 5.7 可知通信流量与代理迁移流量相比是非常小的，因此不会对整个系统的开销有太大的影响。

5.4.2 执行时间分析

第一种情况的执行时间是指第三方可信赖平台派出移动代理直到移动代理返回所需时间。

第二种情况的执行时间是指第三方可信赖平台派出移动代理和复制体直到移动代理和复制体返回第三方可信赖平台并完成最后的校验所需时间。

执行时间的测量是通过在移动代理代码中加入系统时间变量来完成的。

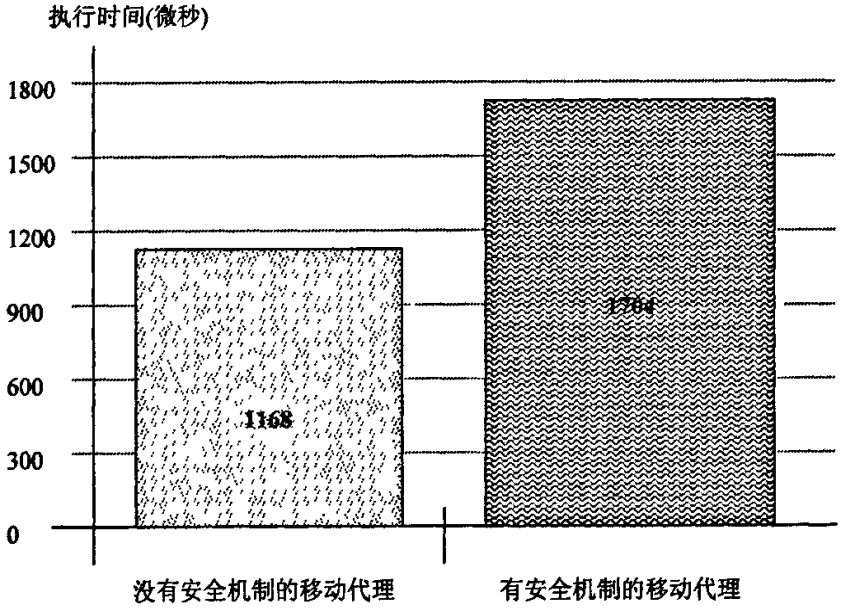

图 5.8 两种情况的执行时间对比

Fig. 5.8 Contrast of execute time of two cases

和没有安全机制的移动代理的执行时间相比，有安全机制的移动代理多需要 46%的时间，这些额外的时间开销，主要是因为要给移动代理加密解密、复制体与移动代理要通信以及移动代理等待复制体在可信赖服务器上运行所致。

结　论

我们首先介绍了移动代理的概念，然后讨论了移动代理的安全问题，进而引出嵌入式移动代理的安全问题。

嵌入式移动代理的安全问题有其特别的一面，嵌入式设备的有限性以及无线接入网络的缺陷，使得嵌入式移动代理的安全更为复杂。

本文首先分析了嵌入式设备的有限性，再分析了无线接入网络的缺陷，在前人工作基础上，将现有的一些移动代理安全解决方案进行了改造，使其能够满足嵌入式移动代理安全的需要。主要工作有：

(1) 主机间认证机制。嵌入式设备与第三方平台间的认证以及普通 PC 机两两间相互认证，保证了移动代理在传输过程中不会受到攻击，也保证了移动代理的完整性；

(2) 安全等级 I 策略。该策略应用于嵌入式设备间的移动代理活动。由于嵌入式设备的有限性，该策略的要旨就是简单又安全，简单地说就是将关键处理放置在第三方可信赖平台上运行，嵌入式设备本地只运行最简单的操作。同时增加简单的限时检测来增加移动代理的安全性。

(3) 安全等级 II 策略。该策略应用于嵌入式设备与普通 PC 机间的移动代理活动。它让复制体在可信赖服务器上、本体在目的平台上同时运行，比较复制体与本体运行后的状态来判断本体是否收到攻击。属于一种强检测方法。

从模拟实验的结果来看，这个安全方案是可行有效的。

但是，安全策略虽然提高了移动代理的安全，同时也增加了系统花费和时间开销：

(1) 主机间的认证随着参与的主机数量的增加而复杂化，这必然不利于大规模的实际应用。

(2) 安全等级 I 策略由于采取了单跳的模式，使得一个代理巡游整条路线的时间大大增加，当路线非常长的时候，这个时间将无法忍受。

(3) 根据实验结果来看，安全等级 II 策略中由于采用了复制体同时运行并比较结果的方法，同样使得整个移动代理系统的效率降低。

因此如何进一步完善安全策略，并且在安全和系统性能间取得一个合理的平衡，将是本课题以后的重要工作。

参 考 文 献

[1] Greenberg M S. Mobile agents and security. IEEE Communications Magazine. 1998, 36(6): 1-10.
[2] 周冲, 孙勇强. PTIR一个用于检测恶意主机攻击的记录协议. 计算机研究与发展. 2002, 39(3): 295-300.
[3] Michael S. Mobile agents and security. IEEE Communications Magazine. 1998, 36(7): 76-85.
[4] WJansen. Countermeasures for mobile agent security. Computer Communications. 2000, 23(17): 1667-1676.
[5] 程臻, 郝智泉, 舒玉华. 嵌入式实时系统中跨平台通信的实现. 电子设计应用. 2003, (4): 53-56.
[6] 莫秀刚. 在国产嵌入式操作系统上构建 J2ME 平台 DeltaKVM: (硕士学位论文). 北京: 北京理工大学, 2002.
[7] 梁合庆. 嵌入式系统用 Java 语言. 电子产品世界. 2001, (4): 24-25.
[8] 刘大有, 杨鲲, 陈建中. Agent 研究现状和发展趋势. 软件学报. 2000, 11(3): 315-321.
[9] Antonio C. Mobile agent technology from first proposals to current evolutions. Microprocessors and Microsystems. 2001, 25(2): 63-64.
[10] 周勇刚.移动 Agent 系统安全技术研究: (硕士学位论文). 四川成都: 西南石油大学, 2002. 04.
[11] Hohl F. A model of attacks of malicious hosts against mobile agents. 4th Workshop on Mobile Object Systems, France, 1998: 105-120.
[12] Tardo J, Valente L. Mobile agent security and Telescript. Proceedings of IEEE COMPCON'96, Santa Clara, California, 1996: 58-63.
[13] Robert S, Gray. Agent tcl: A flexible and secure mobile-agent system. Proceedings of the Fourth Annual Tcl/Tk Workshop(TCL96), Monterey, 1996: 9-23.
[14] Karjoth G, Danny B, Lange et al. A security model for aglets. IEEE Internet Computing. 1997, 1(4): 68-77.
[15] Karnik N. Security in mobile agent systems: [Ph.D. Dissertation]. Minnesota: University of Minnesota, 1998.
[16] Chess D, Grosof B. Itinerant agents for mobile computing. IEEE Personal Communications. 1995, 2(5): 34-49.
[17] Joann J, Ordille. When agents roam, who can you trust?. Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, Oregon, 1996: 188-191.
[18] Gosling J, McGilton H. The java language environment: A white paper. Sun Microsystems, 1996.
[19] Necula G, Lee P. Safe kernel extensions without run-time checking. Proceedings of the 2nd Symposium on OSDI'96, Seatle, Washington, 1996: 229-243.

[20] Wahbe R, Lucco S, Anderson T. Efficient software-based fault isolation. Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review, Asheville, 1993: 203-216.
[21] Fuggetta A, Picco G P, Vigna G. Understanding code mobility. IEEE Transactions on Software Engineering. 1998, 24(5): 342-361.
[22] Wroblewski G. General method or program code obfuscation: [Ph. D. Dissertation]. Poland: Wroclaw University of Technology, 2002.
[23] Mullender S. Distributed systems (2nd edition). New York: Addison-Wesley, 1993.
[24] Colllberg C, Thomborson C, Low D. Taxonomy of obfuscating transformations. Technical Report#148, Department of Computer Science, University of Auckland, 1997.
[25] Siff M. Techniques for software renovation: [Ph. D. dissertation]. Wisconsin: Computer Sciences Department, University of Wisconsin, 1998.
[26] Meadows, Catherine. Detecting attacks on mobile agents. DARPA Workshop on Foundations for Secure Mobile Code Workshop, 1997: 64-65.
[27] Oguz, Kaan, Onbilger et al. A distributed and compromise-tolerant mobile agent protection scheme. http://www.cise.ufl.edu/~nemo/papers/IAWTIC2001.htm, 2001.
[28] Roth V. Secure recording of itineraries through cooperating agents. Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems, Secure Internet Mobile Computations, France, 1998: 147-154.
[29] Vigna G. Protecting mobile agents through tracing. 3rd Workshop on Mobile Object Systems, Finland, 1997: 12-25.
[30] Yee B S.A sanctuary for mobile agents. Secure Internet Programming, Berlin, 1999: 261-274.
[31] Biehl I. Ensuring the integrity of agent-based computations by short proofs. Proceedings of the 2nd International Workshop on Mobile Agents, Berlin, 1998: 183-194.
[32] Kassab, Lora. Agent trustworthiness. Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems Secure Internet Mobile Computations, France, 1998: 121-133.
[33] Sander T, Tschudin C. Towards mobile cryptography. Proceedings of the 1998 IEEE Symposium on Security and Privacy, IEEE Computer Society, Los Alamitos, 1998: 215-224.
[34] Palmer E. An introduction to citadel: A secure crypto coprocessor for workstations. Proceedings of the IFIP SEC' 94 Conference, Curacao, 1994: 9-14.
[35] Fünfrocken, Stefan. Protecting mobile web-commerce agents with smartcards. Proceedings of the 1st International Symposiumon Agent Systems and Applications, New York, 1999: 90-102.
[36] Loureiro S, Molva R. Function hiding based on error correcting codes. Proceedings of Cryptec' 99, International Workshop on Cryptographic Techniques and Electronic Commerce, Hong Kong, 1999: 92-98.

[37] Riordan J, Schneier B. Environmental key generation towards Clueless Agents. Mobile Agents and Security, Berlin, 1998: 15-24.
[38] Bern E Z. Protecting mobile agents in a hostile environment. Agent Technology III (MAMA' 2000), ICSC Academic Press, Canada, 2000: 186-192.
[39] Wagner A. Implementing mobile agent security in an untrusted computing environment. 8th International Conference on Telecommunications, Zagreb, Croatia, 2005: 591-594.
[40] Benachenhou L, Pierre S. A new protocol for protecting a mobile agent using a reference clone. MATA, Montreal, Canada, 2005: 364-373.

攻读硕士学位期间发表学术论文情况

[1] 王宇新，张雷，郭禾. 嵌入式移动代理的研究与实现. 南开大学学报(自然科学版)增刊，2005，38:41-44. 在学位论文中的章节：第三章.

致　谢

在研究生阶段的学习和生活即将结束之际，我要向我的导师、同学和家人致以诚挚的谢意。

本论文的研究与撰写是在导师郭禾老师、王宇新老师和陈锋老师的悉心指导下完成的。

我非常荣幸能在郭禾老师的指导下攻读硕士学位。郭老师知识渊博，考虑问题很有深度，总能指出问题的本质所在。在接近三年的学习里，郭老师给嵌入式组提供了大量的硬件设备，使得我们能够通过大量的实践来印证我们的研究，增强了我们的动手能力，也积累了大量的经验。生活上，郭老师平易近人，在他脸上永远绽放着微笑，散发着无尽的亲和力。对于学生来说，他不仅是一位明师，更是一个人生中的榜样和学习的楷模。在这里我再次向郭老师表示由衷的感谢和敬意。

陈锋老师理论基础扎实，思维极其活跃，总是能提出一些新想法新理念新概念，让你感觉到一盏学术上的明灯总是在你面前闪耀。王宇新老师在技术上有着独特的嗅觉和敏感，不管多么困难的技术问题，他总能给出一些关键的提示，使得我能够豁然开朗。他们都是值得我学习的榜样。我在这里对他们表示由衷的感谢和敬意，特别地，祝愿远在英伦的陈锋老师能够圆满地获得博士学位。

如果没有同学们的帮助，我一个人是无法走到今天的。我要感谢仙云森、赵振兴和张海生等嵌入式组成员，他们在学术上给我了很大的启示，而且在撰写毕业论文的过程中给我了很大的帮助。我还要感谢韩东、孙玉成、郭东卿等，他们给我的生活带来了很多快乐。

我很高兴能与师弟师妹们在和风实验室相逢，他们不仅是我的弟弟妹妹更是我的朋友，我会永远怀念与他们一同走过的日子。

同时，我要感谢我的父母，他们含辛茹苦地把我养大，供我读书，我长大了成才了，他们却老了，没有他们二十多年来的含辛茹苦，就不会有我的今天。

最后，我要感谢所有给予过我关心、支持和帮助的人们！