移动 Agent 系统的迁移机制研究

On the Migration Mechanism of Mobile Agent System

张冠群　陶先平　李　新　冯新宇　吕　建

（南京大学计算机软件新技术国家重点实验室　南京210093）

Abstract　Mobility is inherent in Internet, traditional message passing, RPC, or Remote Evaluation, can not meet more and more various mobility requirements. Mobile Agent implements computation migration, including data, code and control, is a good way to suit Internet. Migration mechanism is the key technology of mobile Agent. We discuss it from the perspective of both system programmer and application programmer and introduce migration mechanisms of some famous Java based mobile Agent system. At last, present the structured migration mechanism of self-designed mobile Agent system Mogent.

Keywords　Computation migration, Mobile Agent, Migration mechanism

一、引言

网络就是计算机，Internet 提供了分布的信息、软件和硬件等资源，正成为一个全球计算的基础设施。它具有资源链接的开放性和动态性，使用方式的灵活性和个性化，网络环境的不可预测性等特点。传统的客户服务器模型不能很好地适应这些新的特点，它以服务器为中心的结构使得客户的个性化能力受到了限制，同时网络协同的适应性也比较差。

移动 Agent 技术是随着 Internet 的发展而出现的一种新兴技术，它可以较好地适应 Internet 特点，有效地简化分布式系统的设计、实现和维护[2]。一般来讲，移动 Agent 是一段独立的计算机程序，它按照一定的规程，能够自主地在异构网络上移动，寻找合适的计算资源、信息资源或软件资源，利用与这些资源同处一台主机或一个局部网络的优势，处理或使用这些资源，代表用户完成特定的任务。Agent 具有下面一些特点：①自主性：自主决定在什么时候迁移到什么地方去，做什么事情；②移动性：可以跨机移动，连续计算；③协同性：其间可以通讯，进行分布协作；④安全性：对 Agent 本身及 Agent 运行环境的安全性保障；⑤智能性：可以感知和适应环境的变化。

基于移动 Agent 的应用结构灵活，开放性好，可以有效降低网络负载，较好地适应异构环境，具有较高的坚定性和容错能力，在电子商务、协同计算、服务器个性化等方面有较好的应用前景。对移动 Agent 技术的研究正成为学术界和工业界的热点之一。

移动 Agent 的关键技术包括迁移机制、通讯模型、安全体系等，其中迁移机制是移动 Agent 的核心和基础技术，它保障 Agent 可以在网络的各个节点之间迁移，进行连续计算。本文从移动技术的发展开始，讨论 Agent 迁移机制所涉及的内容，并在最后介绍我们在自主设计和实现的 Mogent 系统中的结构化迁移机制。

二、移动的理解

一个计算过程包括数据、代码及控制三部分内容，几个计算过程之间的协调要求彼此之间交换一定的内容，可能是数据、代码或者是控制。我们把发出请求一方称为客户端，接收请求一方称为服务端，而把内容的交换称为移动。开始人们通过存储媒质，如磁带、磁盘等，以人工方式完成从客户端到服务端的移动。随着计算机网络的出现，这种移动逐渐通过网络来完成，目前主要有下面几种途径：

·**Message Passing**[9]　通过 send 和 receive 两条基本原语在不同的进程之间传递消息。它移动的只是数据，客户端调用 send 后可以继续执行，再在适当的地方接收回答消息。消息传递相当灵活，功能十分强大，但是要求程序员处理很多底层的细节问题，例如如何保证消息的一一对应关系等，这使得程序员过多地陷于通讯和消息细节，影响对程序逻辑的考虑。

·**RPC(Remote Procedure Call)**[10]　允许程序员像调用本地方法一样调用远程方法，大部分 RPC 都采用 stub 过程来实现。在这种方式下，客户端的 stub 截获客户的方法调用，把数据参数打包移动到服务端，同时也把计算的控制交到服务端，调用者等待返回；服务端控制方法执行后把结果数据传回给 stub，由 stub 把结果返回给客户端，同时客户端重新获得控制继续往下执行。它把程序员从底层细节中解放出来，但它的缺点是不够灵活，客户端只能调用服务端已有的远程方

法，而且需要保持持续连接，要求安全稳定的网络环境。

·**REV(Remote Evaluation)**[1] 该方法移动的是一段代码。允许客户端把一段代码上传到服务端，进行资源的本地化和个性化处理，最后返回结果，不需要服务端提供专门的服务。这种方法能够容易让服务端满足客户端的计算要求，但缺点是移动的只是代码，不包括状态，如初始状态、全局变量等等；而且代码只能移动一次，不能进行连续计算。

上面三种常见移动技术不能很好地适应动态多变的 Internet 环境，而且只能迁移计算过程的部分内容。移动 Agent 扩充了 REV 的功能，可以实现包括数据、代码和控制在内的整个计算的连续移动，允许和其他移动 Agent 通讯。它具有以下一些优点：

·把远程交互变成局部交互，避免中间数据在网络上的传输，减轻网络负载，而且对网络延时、抖动等隐患不敏感；

·可以封装用户的私有协议，迁移到其他节点上后可以和用户按照私有协议规定的规程进行交互；

·Agent 可以自主控制，支持异步交互，不需要保持持续的网络连接，用户派出 Agent 后可以断开网络连接，等 Agent 回来的时候再恢复连接；

·容易支持服务的个性化，不需要服务器提供固定的服务，可以通过 Agent 来封装服务器上的多个服务，形成只给一个用户的个性化服务；

·更容易适应多变的网络环境，具有天然的容错性和坚定性。

三、迁移机制

迁移机制是移动 Agent 的基础核心技术，它支持 Agent 在网络上自主的移动，可以在合适的节点上停留下来，完成一定的任务，然后继续移动和执行，直到完成用户的任务，最后把结果返回给用户。Agent 的移动性可以带来很多好处，例如解决单纯 Remote Evaluation 的不足，平衡节点间的负载，提高分布式系统的容错能力和实时性，支持异步交互，减少通讯开销，具有自适应的扩展能力等等。

一般来说，Agent 在两个节点之间移动的基本过程如图1所示。迁移机制包括克服平台异构性、移动粒度大小、移动规程表示等方面的内容。Java[1]的平台无关性以及在 Internet 上的广泛分布，加上它的安全模型和丰富的网络支持使得 Java 语言可以较好地满足移动的基本要求，现有的大部分移动 Agent 系统都采用 Java 作为基础开发平台，下面的讨论也主要面向基于 Java 的移动 Agent 系统。移动粒度和移动规程是迁移机制的核心内容，它们分别从系统程序员的角度和应用程序员的角度来说明迁移机制，对应着 Agent 移动的实现和表示。

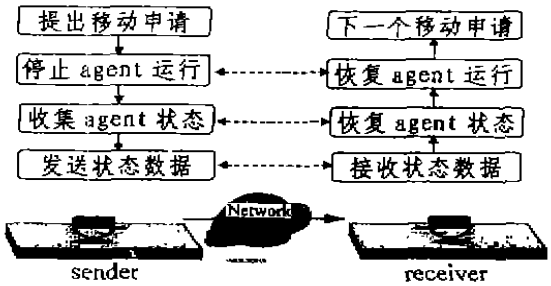

图1 Agent 流动过程简图

3.1 移动粒度

移动粒度指 Agent 移动的最小单元，可以是语句、过程等等，它是系统程序员关心的主要问题，关系到系统实现的难易程度和复杂性等方面。在基于 Java 的移动 Agent 系统中，移动粒度从细到粗，主要有语句迁移和过程迁移两种，语句迁移使 Agent 在目标节点上可以从迁移前的断点语句开始继续执行；过程迁移只能支持 Agent 在目标节点上从一个过程开始重新执行。

为了让 Agent 能在节点之间移动并连续计算，必须保存 Agent 在源节点上的状态，传输到目标节点上，然后恢复运行。从技术的角度，Agent 的状态信息可以分成三部分[1]：对象状态，指 Agent 对象变量的值；代码状态，指 Agent 执行所需的代码；执行状态，与 Agent 执行线程或进程相关的系统信息。

语句迁移比过程迁移要携带更多的状态信息，它需要捕获上述所有三部分状态，移动到目标节点后恢复成和移动前一模一样的运行环境，如图2所示，因此又称为强迁移；过程迁移只捕获对象状态和代码状态，在目标节点上只恢复 Agent 对象，而执行状态可以和源节点上的不一样，如图3所示，因此又称为弱迁移。

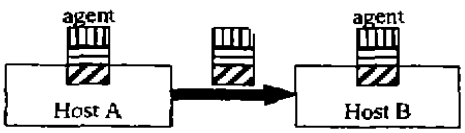

图2 语句迁移示意图

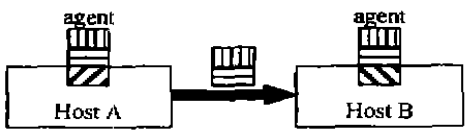

图3 过程迁移示意图

Agent 对象状态　Agent 代码状态　Agent 执行状态

语句迁移保存了 Agent 的完整状态，在目标节点上恢复成和源节点一模一样的运行环境，Agent 可以刚好从移动前的断点开始继续往下执行，对于 Agent 执行线程或进程来说是一种完全透明的迁移，Agent 本身并不了解是否移动了。但是它给实现带来了很大的复杂性和难度，因为在 Internet 的异构环境下，捕获和恢复 Agent 在节点上的所有状态，特别是和平台相关的执行状态，是十分困难的：

·Agent 的执行状态包括程序现场在内的一系列系统信息，应用程序很难直接访问和存取这些信息。在基于 Java 的移动 Agent 系统中，必须得到 Java 解释器的支持才能把 Agent 的执行状态从解释器中抽取出来或者重新插入到解释器中去，但目前的 Java 解释器并不支持这个功能。如果通过修改 Java 解释器达到目的，同时也丧失了 Java 平台通用性的优点。

·需要消耗较大的网络带宽。Agent 的执行状态数据一般都比较大，和 Agent 本身的状态数据相比无法忽略，特别在 Agent 本身状态不多时甚至会大于 Agent 的对象状态数据，使得 Agent 在移动过程需要携带更多的数据，不能充分发挥移动 Agent 可以减少网络负载的优点。

·难以保持和维护对本地环境资源的透明引用。一般来说，透明的迁移要求移动 Agent 程序可以透明地引用资源。以文件为例，实现透明文件引用需要一个分布式文件系统，支持打开文件、暂停文件连接，在其它地方恢复与文件的连接等等。虽然可能通过 NFS、CORBA 等系统实现文件的透明使用，但是对于所有其他局部资源同样需要类似的分布式管理，这十分困难。

过程迁移只支持 Agent 到达目标节点后重新开始执行一个过程，因此不需要在源节点上执行的程序计数器、方法调用栈等程序现场信息，根据 Agent 的对象状态以及所需的执行代码就可以从一个过程的入口开始重新执行。语句迁移的实现困难主要在于需要处理 Agent 的执行状态，而过程迁移只移动 Agent 的对象状态和代码状态，并不处理 Agent 的执行状态，因此过程迁移大大降低了系统实现的复杂性。但是过程迁移方式在一定程度上是把系统程序员的复杂性转嫁到应用程序员的头上，它要求应用程序员把握 Agent 整个迁移的过程，不支持透明的迁移，要求程序员知道每个过程会在什么时候、什么地方开始执行，可以有哪些假设等等，因此编写 Agent 程序会和传统顺序程序的开发不太一样。

3.2 移动规程表示

移动粒度在系统层面上说明了 Agent 迁移的系统支撑，移动规程则在应用一级上说明应用程序员如何表达 Agent 的移动信息。它包括两部分内容，一是在源节点如何表达移动意愿，二是在目标节点如何恢复执行。前者反映了 Agent 在源节点上什么时候开始移动，移动到什么地方去；后者说明了 Agent 在目标节点上从什么位置开始恢复执行。应用程序员希望移动规程能够满足各种移动情况，同时具有易用性好、自然化程度高等特点。

人的旅行一般有两种方式：一种是随意旅行，等旅行完一个地方后再决定下一个目的地；另一种是事先定好一个计划，然后按照计划旅行。和人的旅行方式类似，Agent 移动意愿的表达也有这样两种，我们分别称之为显式方式和隐式方式。

·在显式方式下，Agent 系统提供一条移动命令，如 go、jump、migrate 等，在 Agent 程序内直接调用这条命令，并附以适当的参数如目的节点地址，可以随时把 Agent 移动到目的节点去。对应用程序员来说，这种方法灵活性好，表达能力强，可以满足任何移动要求。很多现有移动 Agent 系统都采用这种方式，如 Agent Tcl、Mole、Aglet 等。但是同时也增加了管理的复杂度，由于 Agent 功能体和 Agent 移动信息混杂在一起，如果不对 Agent 的功能、移动条件和要求等情况有全局而深入的了解，Agent 的设计、实现和调试都有相当的难度，尤其是在 Internet 环境下，节点和资源都在动态变化，更是难以管理和控制 Agent 的移动和执行，而且不符合现代软件工程的思想，Agent 功能体以及移动信息都无法复用，每一次都需要重新开发。

·在隐式方式下，一般采用旅行计划的形式。应用程序员事先给 Agent 制定一个旅行计划，说明需要访问的目标节点地址以及相应需要执行的过程。启动后，Agent 将逐步解释执行旅行计划中的每一项，直到完成整个计划。采用旅行计划的形式使得应用程序员可以控制整个旅行过程，便于管理和控制。同时它分离了 Agent 的功能体和移动信息，易于复用和二次开发。但是它的缺陷是灵活性不够，表达能力比显式方式要弱，对动态环境的适应能力比较差。因此，一般都提供对旅行计划的修改方法，使得 Agent 在旅行过程中可以动态修改旅行计划。需要注意的是动态修改使隐式方式类似于显式方式，Agent 的移动变得难以管理和控制，因此我们需要在两者之间寻找一个平衡点，既方便 Agent 的管理和控制，又保证表达能力和灵活性。

Agent 到达目标节点后需要根据携带的状态恢复运行，对于语句迁移来说，Agent 将从迁移前的断点开始往下继续执行；但对于过程迁移来说，因为它是重新开始执行一个过程，这样我们就有两种选择，固定的入口过程或者是任意的入口过程。固定入口过程是指 Agent 到达目标节点后都执行一个统一的过程，在

这个过程中根据Agent的对象状态以及环境状态决定应该执行哪个动作；任意入口过程是指Agent在迁移前就说明目标节点地址以及需要执行的入口过程名，到达目标节点后将执行指定的过程。固定入口过程易于实现但不够灵活，需要在过程内判断状态；任意入口过程灵活方便但实现比较复杂一些。

四、几个常见的迁移机制和典型系统

迁移粒度和移动规程可以组合很多不同的迁移机制，其中有一部分组合是不合理的，如语句迁移不可能支持隐式规程和方法入口；过程迁移不可能实现断点语句入口等等。

表1 迁移粒度和移动规程组合表

	显式命令 语句入口	显式命令 单一入口	显式命令 多入口	隐式计划 语句入口	隐式计划 单一入口	隐式计划 多入口
语句迁移	可以	不可能	不可能	不可能	不可能	不可能
过程迁移	不可能	可以	不常见	不可能	不合理	可以

去除不可能或不合理的，一共有下面这样三种常见的迁移机制，下面我们分别就几个现有移动Agent系统来说明：

4.1 语句迁移，提供显式的移动命令，断点语句入口——D'Agent[4]

D'Agent(Agent Tcl)是Dartmouth大学开发的移动Agent系统，采用Tcl语言实现，Agent在解释器上运行，可以为每种语言定制一个解释器，因此它支持多种语言的Agent在其系统上运行。它的移动过程如图4所示。

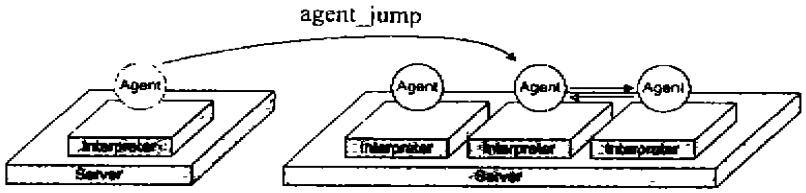

图4 D'Agent迁移示意图

Agent通过Agent_jump命令在节点之间移动，这个命令收集Agent的全部状态，传输到目标节点，然后从Agent_jump之后的指令开始继续执行。

4.2 过程迁移，提供显式的移动命令，固定入口过程——Aglet[5]

Aglet是IBM开发的基于Java的移动Agent系统，它的生命周期图如图5所示。

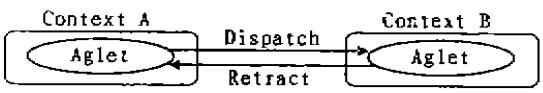

图5 Aglet迁移示意图

它的特点是提供了主动迁移(dispatch)和被动迁移(retract)两种移动，并且采用事件监听机制来处理移动过程，提供各种接口来处理事件，接口onDispatch、onArrival分别响应主动迁移和到达两个事件。

4.3 过程迁移，提供隐式的移动表达，任意入口过程——Concordia[6]

Concordia是由Mitsubishi开发的基于Java的移动Agent框架。它通过一个Itinerary模型来刻画Agent的迁移信息，提供目标节点地址和入口过程的列表，Agent根据Itinerary逐步移动。在移动过程中，Concordia支持动态修改Itinerary，以提高灵活性和表达能力，但是允许任意动态修改从极端上来讲和提供jump等命令是等价的，削弱了隐式规程的优点。它的移动过程如图6所示。

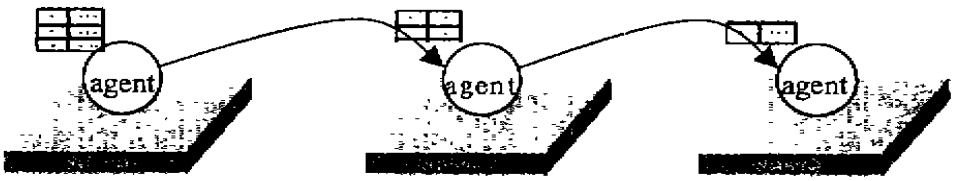

图6 Concordia迁移示意图

五、Mogent 系统的结构化迁移机制[1]

Mogent 系统[7]是由南京大学计算机软件新技术国家重点实验室自行研制的移动 Agent 系统，以 Internet 为基础网络环境，以 Java 语言及其环境为基本的语言支撑。从概念上讲，Mogent 系统由 host 和 mogent 两个实体构成，host 是 Internet 上的节点的抽象；mogent 是移动 Agent 的概念模型。

结构化迁移机制以旅行计划模型为核心，实现过程迁移，采用隐式表达方式，支持任意入口过程。主要特点是用旅行计划把移动信息从 mogent 功能体中分离开来，可以实现动态装配，并用结构化的方法来描述旅行计划，它在保证表达能力和灵活性的基础上降低用户开发 mogent 的负担。

5.1 mogent 功能体和移动信息的结构分离

移动 Agent 所面临的网络环境是不确定的，应用程序员事先无法知道运行时刻的情况，这种不确定性给移动 Agent 的设计和编写带来很大的困难。用结构化的旅行计划模型把 Agent 的移动信息从功能体中分离开来，用户可以分别进行分析和设计，有效地降低了设计和编写 mogent 的复杂性，也便于用户对 mogent 的调试和管理。同时旅行计划和功能体的结构分离可以实现旅行计划和功能体的动态装配，提高了复用程度。

5.2 结构化的旅行计划模型

旅行计划由旅行模式和若干个旅行步组成，一个旅行步说明了一站旅行要求，旅行模式是对这若干个旅行步的解释方法。我们采用结构的方式来描述一个旅行步，同时对旅行步的解释也按照固定的规程进行，一旦旅行计划制定完成后，就不可以随意进行修改，只能在旅行计划全部解释完之后才能修改和调整。旅行计划表如下所示，

表2 旅行计划表模型

SEQ/ALL	P_0	H_0	G_0	M_0	T_0
	P_1	H_1	G_1	M_1	T_1
	.....				
	P_{n-1}	H_{n-1}	G_{n-1}	M_{n-1}	T_{n-1}

SEQ 表示以顺序方式解释旅行计划表，ALL 实现一种分发移动模式，以满足对并行性的要求。旅行步中 H 是目标节点地址，M 表示入口方法，G 是入口方法的卫士函数，判断环境以及 mogent 本身状态，只有当 G 方法返回为真时，才会执行 M 方法，提出 G 方法的目的是分离用户任务的实现和对环境的判断，提高各部分的独立性。

结构化模型要求在旅行过程中保证旅行计划结构不受破坏，不允许 mogent 在运行的过程中修改。为了保证表达能力和灵活性，我们在旅行步中提供旅行步控制 P 和更新方法 T，mogent 通过它们对旅行计划进行动态调整。由于对 P 和 T 的解释执行是按照解释规程在特定时刻进行的，因此不会破坏旅行计划的结构。P 在移动前进行状态判断，返回假则跳过该旅行步，转向下一个旅行步，这样可以间接调整旅行计划。T 是唯一可以直接修改旅行计划的方法，但它只能在旅行计划被全部解释后才能执行。

我们可以发现这个模型和结构化程序设计语言有一定的相似性。SEQ 模式对应顺序程序设计，对旅行步的顺序逐步解释类似于程序的顺序执行，旅行步控制类似于选择语句，旅行计划更新方法则可以完成循环语句的功能。ALL 模式则对应着简单的并行程序设计结构，根据旅行计划生成多个 mogent 并行执行。如果把移动信息的表达看成刻画 Agent 在 Internet 环境中移动的"程序"设计，则 Agent_jump、dispatch 等就相当于程序设计中的 goto 语句，能力强但不易理解和控制；我们的旅行计划方式可以看成是结构化程序设计，它具备顺序、选择和循环三种程序设计的基本结构，具有足够的表达能力，可以表达所有的移动要求，而且表达出来的移动信息易于理解和控制。

总结和进一步工作 本文从移动粒度和移动规程两个角度对移动 Agent 系统的迁移机制进行说明和归类，阐述了语句迁移、过程迁移、显式表达方式以及隐式表达方式的含义，并结合几个常见移动 Agent 系统说明了几类不同的迁移机制。最后介绍了由南京大学计算机软件新技术国家重点实验室自行研制的 mogent 系统的结构化迁移机制，通过分离功能体和移动信息来降低开发复杂度，提高了复用程度，同时采用结构化旅行计划模型保证了表达能力和灵活性。

移动 Agent 迁移机制的研究还有很多内容，怎样解决多线程之间的移动冲突，如何提高移动的性能和效率，如何提高 Agent 移动的容错性等问题需要做进一步深入的研究，同时也需要加强对移动 Agent 理论的研究，提供理论基础。

参 考 文 献

1 陶先平、吕建 一种新的移动 Agent 结构化迁移机制的设计和实现. 软件学报，2000，11(7)：918～923

2 Lange D B. Mobile Objects and Mobile Agents: The Future of Distributed Computing? In: Proc. of ECOOP'98, pp. 1～12

3 Funfrocken S. Transparent Migration of Java-based Mobile Agent. In, Proc. of Second Int Workshop on Mobile Agents 97, MA'98, pp. 26～37

（下转第60页）

出现有效电平，立即向控制逻辑发出中断申请信号。在中断服务程序中读入中断排队寄存器的内容，从而可判断出发出中断申请的从机号。当有两个或两个以上从机同时发出中断申请时，可用软件实现中断排队。同时读入状态输入寄存器的内容，判断出数据欲发往的目的机号，并根据情况决定是否响应其中断。若响应其申请，可向从机选择寄存器发中断响应代码。从机译码器译出中断响应代码后，开始向目的机发送数据过程。与此同时，主机根据目的机号，启动目的机接收数据过程。

主机与从机接口卡上的控制逻辑负责产生控制收发三态门的控制信号及其它各类控制信号。

各微机间数据总线的连接采用485标准接口，8对非屏蔽双绞线以总线方式互连在一起。接口芯片面MAX3485，带宽10M。

三、紧耦合多机系统 DMA 传输的简单通信协议

1. 数据格式

发送字节数	数据块
2byte	小于64kB

2. 主机发送，从机接收

(1)发送方

```
A. start
B. initializing local DMA;初始化本机 DMA;
C. write(code)→port(s);向"从机选择端口"写"选择从机中断"编码;
D. open gate(t);打开数据通道发送控制门;
E. begin(DMA)→1(fifo);启动本机 DMA 向本机接口板上 FIFO 写数据;
F. if EF=1
     then close gate(t),return(success);若 FIFO 被读空(EF=1),则发送成功,关闭发送控制门,返回
   else
     if timeover then close gate(t),return(fail),否则,若超时,则发送失败,关闭发送门,返回,未超时,继续等待
       FIFO 读空,
   else
     goto F
   endif
 endif
```

(2)接收方

主机发出的"选择从机中断"编码经各从机译码后，选中的从机产生本机中断请求信号。在中断服务程序中：

```
A. start
B. open gate(r); 打开从机数据通道接收门;
C. r(count)=read(I/O); 先用 I/O 读,从主机 FIFO 中取出两个字节数据,作为接收字节数。
D. initializing local DMA;初始化本机 DMA
E. if T/C=1
     then close gate(r),return(success),若 DMA 读结束信号 T/C 有效,则关闭接收门,收成功,返回。
   else
     if timeover then close gate(r),return(fail);否则,若超时,则关闭接收门,接收失败返回。
     else
       goto E;未超时,继续等待 T/C 信号
     endif
 endif
```

(3)从机向主机或另一从机发送数据的情况与主机向从机发送数据类似，此处从略。

结论 该系统经实验测试，以 DMA 方式实现主机与从机之间，从机与主机之间的数据传递正确无误。启动一次 DMA 最大连续传送字节数，可达60kB，对于大于60kB 的数据块，可分组传递。

参 考 文 献

1 Ripps D L. An Implementation Guide to Real-time Programming. 1989 by Prentice-Hall. Inc.

2 刘文涛，李桂琴. AI 型智能网络系统结构与体系结构. 数据通信，1992(1)

3 李桂琴，刘文涛. IW 的多微机并行推理系统. 数据通信，1992(3)

4 金兰，等. 并行处理计算机结构. 国防工业出版社，1982

5 Liu Wentao, et al. Development of Intelligent Workstation. ICCC, 1991

(上接第73页)

4 Gray R S. Agent Tcl: A flexible and secure mobile agent system. In: Proc of 4th Annual Tcl/Tk Workshop, 1996. 9～23

5 Lange D B, Oshima M, Mitsuru O. Programming and Deploying Mobile Agents With Aglets. Addison-Wesley, 1998

6 Wong D, et al. Concordia: An Infrastructure for Collaboration Mobile Agents. First International Workshop on Mobile Agents 97, MA'97

7 吕建，陶先平，董桓，李新，张冠群. 基于 Internet 的移动 agent 系统总体设计报告：[南京大学计算机软件新技术国家重点实验室技术报告]

8 Arnold K, Gosling J. The Java Programming Language. Addison Wesley, 1996

9 Singhal M, Shivaratri N G. Advanced Concepts in Operating Systems: Distributed, Database and Multiprocessor Operating Systems. McGraw-Hill Series in Computer Science. McGraw-Hill, New York, 1994

10 Gifford D K, Glasser N. Remote pipes and procedures for efficient distributed communication. ACM Transactions on Computer Systems, 1988, 6(3): 258～283

11 Stamos J, Gifford D. Remote evaluation. ACM Transactions on Programming Languages and Systems, 1990, 12(4): 537～565