计算机科学2000Vol.27№2.6

多 Agent 系统中实时消息传递机制*)

Message Transmission Mechanism in Multi-Agent Systems

王寻羽　朱淼良　邱　瑜　唐文斌
(浙江大学计算机系　杭州 310027)

Abstract　Message transmission is the key element which helps cooperating and self-organizing between agents in distributed multi-agent based systems. A postoffice-based model of message transmission proxy, via which a distributed computing environment was constructed between heterogeneous platforms, was proposed in this paper. It transmits messages of agents in real-time. The proxy system has been implied in autonomous robot architecture and the real-time capability and robustness was proved by simulation experiment.

Keywords　Multi-agent, Middleware, Network parallel, Distributed system

1　引言

Agent 是具有信念、能力、决策和承诺等精神状态的实体[1]，它的提出是以计算的社会学模型为背景，通过多个简单 Agent 协同工作，可以完成比较复杂的任务。分布式人工智能(DAI)的主要目标就是解决如何组织多 Agent 系统中的 Agent，使它们能够协同工作以完成复杂问题求解[2]。多 Agent 系统的关键问题是 Agent 间的协同，而根据计算社会学模型的观点，消息是 Agent 协同的中介，因此，多 Agent 系统首先要解决的问题是 Agent 间的消息传递。实现分布式多 Agent 间的通信及互操作有多种方法和机制[3]，大致可分为两类：一类是基于消息传递的实现机制，如 PVM；另一类是基于远程过程调用(RPC)的实现机制，如 CORBA。

PVM(并行虚拟机)是美国国家基金会资助的开放软件系统，它是对 UNIX 系统中进程间消息传递机制的扩展[4]。它通过运行在网络结点机上的监控进程完成进程间的通信，提供的接口主要是接收和发送函数，与 UNIX 系统中的消息传递调用相类似。PVM 的优点在于调用接口相对简单、易操作、系统规模小，但由于没有提供中断类的消息响应函数(系统只提供同步、异步及超时三种消息响应函数)和缺乏对面向对象的支持，使得基于 PVM 的应用程序必须自己考虑这些问题，

CORBA(公共对象请求代理体系结构)是 OMG 组织在1991年制定的一种规范，用于完成分布式对象的互操作[5]。CORBA 体系将对象的实现与对象的定义分离，客户端通过对象定义接口来完成对分布式对象方法的调用。CORBA 体系的核心部分是 ORB 和 IDL。ORB(对象请求代理)解释客户端对象的请求，将其发送至合适的服务器端，由服务器激活相应的对象。IDL(接口定义语言)是对公用对象接口的定义，用以指明此对象可提供的服务。CORBA 体系的优点是提供对通用对象组件的支持，有良好的面向对象特性，IDL 技术的引入使得对象的实现与对象的定义完全分离，也即对象的实现不局限于使用某种特定的语言。对于一个完成特定任务的多 Agent 系统而言，CORBA 体系结构显得过于通用和庞大，而且实时性能也不很理想。

综合考虑上述两种体系的优缺点，本文提出了一种基于邮局模型的消息传递代理中间件，用于完成多 Agent 间的通信。它应用于自主式机器人体系结构中，具有如下特点：

□　接口简单，系统规模小，其 C 语言接口只有5个，与 UNIX 系统中的消息传递调用相似，易于理解和使用。

□　实时性强，消息包采用简单协议，打包及解包的耗时只占网络传输时间的很小一部分。

□　提供对面向对象的支持，系统提供 C++ 语言

*)本文得到国防科工委九五攻关项目"智能机器人"资助。王寻羽　博士生，主要研究方向为分布式系统，计算机视觉，虚拟现实；朱淼良　教授，博士导师，主要研究领域为人工智能，计算机视觉，智能机器人体系结构。

接口，应用程序中的对象可通过继承的方式派生出有处理外部输入输出功能的对象。

□ 提供中断类消息响应接口，便于应用进程构造基于事件响应并具有行为特性的 Agent。

2 基于多 Agent 的体系结构

自主机器人是一个工作在变化环境中有一定自主能力并可以完成预期任务的系统。系统体系结构的合适与否决定了自主机器人的性能，也即系统实时性、智能程度及鲁棒性等指标。我们在国防科工委攻关项目中开发的基于分布式多 Agent 的自组织体系结构 I-RASO（自组织智能机器人体系结构）[6,7]，综合考虑实时性与智能程度间的矛盾，利用多个并发 Agent 间的协作及公告板系统做出的态势评估，给出总体决策，协调各 Agent 的行为。基于分布式多 Agent 的体系结构利用网络并行计算的优点，将自主机器人的各功能模块分布于不同的节点，通过消息传递代理完成各 Agent 间的协同。

自主移动式机器人由感知、信息融合、规划、驾驶控制、决策等 Agent 组成。各 Agent 都有一定的自主行为，整体 Agent 表现为公告板系统根据收集到的其他各 Agent 的事件信号，通过对全局知识及预定任务的分析，动态重组各 Agent 的行为。在 IRASO 体系结构中，Agent 间的协同通过消息传递来完成，大致有两类消息。一类是描述道路环境及规划路径的纯数据流，另一类是事件消息，也即各 Agent 提交给公告板的状态信息及公告板系统广播给所有 Agent 的表示总体态势的推理结果。这两类信息在体系结构中统称消息，其中纯数据流消息一般发生在各个 Agent 之间，使得感知、融合、规划等功能模块按照流水线方式工作；事件类的消息用于处理异常情况或控制自主机器人完成相应任务，多发生在公告板与 Agent 之间，以形成状态-决策-行动闭环。

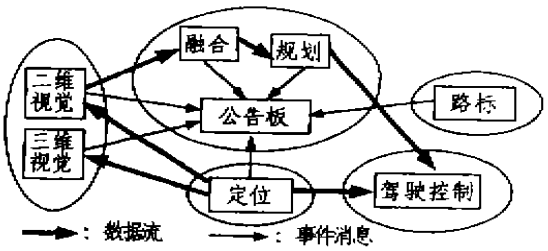

图1 体系结构示意图

为更好提高自主机器人的实时相应速度，IRASO 体系结构中的 Agent 被分布在不同的计算机上，图1是其中之一的体系结构。

3 消息传递代理

体系结构中的 Agent 是特定领域的问题求解单元，分布在不同的机器上独立运行的程序，其模型可简单表示为：

Agent 间的协同通过"端口"间的数据传递来完成，"开关"控制 Agent 的行为，内核即问题求解单元的内容，它可以是专家系统，也可以是设备驱动程序。消息传递代理被设计成实现 Agent 间数据交换的中间件，从某种意义上说，在完整的机器人体系结构中，它也是一种 Agent（为了区别，我们把上述的 Agent 统称为应用 Agent）。消息传递代理的实现使多台计算机在逻辑上成为一台多 CPU 的并行机，而消息传递代理则成为逻辑上的"软总线"。它提供服务来完成 Agent 模型中的"端口"功能。图2中方框外的箭头表示消息传递。

3.1 基于邮局模型的消息传递代理

一般而言，在分布式软件体系结构中，中间件包含三个要素：客户端向服务端发出请求所需的应用程序接口（API）、网络上客户请求的物理传输、客户请求的返回结果[8]。客户程序与中间件构成客户/服务器（C/S）模型。由于 TCP/IP 协议为大多数操作系统所支持，通常 C/S 模型被设计为基于 TCP/IP 的结构，当然 C/S 模型也可设计为基于本地的机制。在 IRASO 体系结构中，为均衡网络负载及提高 Agent 的响应速度，消息传递代理分布于网络中的每个节点，形成基于邮局

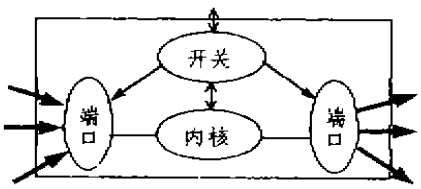

图2 Agent 简单模型

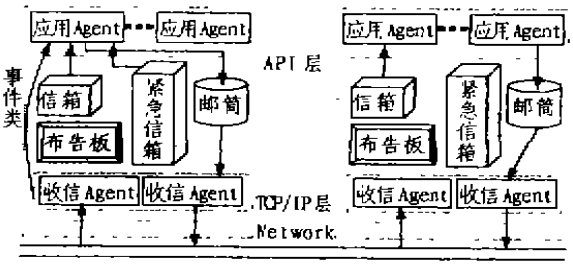

图3 基于邮局模型的消息传递代理

模型的对等结构，如图3所示。

在邮局模型中，Agent 的发送和接收请求均借助于应用 API 由本地的消息传递代理完成，网络传输则在相应节点机上的发信 Agent 和收信 Agent 间进行。邮筒和信箱是消息传递代理与应用 Agent 间的桥梁，其机制类似于邮局与用户间的关系，故称邮局模型。应用 Agent 也即应用进程欲完成发送消息的操作，只需利用消息传递代理中间件提供的 API，将一条消息送至邮筒。邮筒是一个队列缓冲池，在缓冲池满时，应用 Agent 将挂起，等待发信 Agent 取走一条消息，或返回失败信息（在非阻塞方式下）。在 IRASO 体系结构中，这种情况一般不会发生，仿真实验也证实了这一点。这是因为发信 Agent 与现实中定时取信的邮差不同，它被设计成不停从邮筒中取信并根据收件人地址将信件送至收件人所在节点的"勤快"的邮差，为合理分配 CPU 计算资源，发信 Agent 在缓冲池为空时，进入睡眠状态，等待"新信件到来"事件的唤醒。这样，应用 Agent 发送消息的请求可以得到实时响应。由于应用 Agent 及发信 Agent 分属不同进程，邮筒（也即消息队列缓冲池）成为临界资源，存在同步及冲突等问题，为此，消息传递代理系统引入生产者/消息者（Producer/Consumer）模型[3]，利用信号灯机制解决同步和冲突问题。其实现算法为：

发信 Agent

```
/* 初始化各信号灯赋初值 */
semaphore mutex=1;/* 互斥用 */
semaphore empty=QUEUE_LEN;/* 空槽计数器 */
semaphore full=0;/* 满度计数器 */
message msgBuffer;
/* 取信过程 */
while(1){
        P(full);
        P(mutex);
        GetMsgFromQueue(&msgBuffer);
        V(mutex);
        V(empty);
        SendMsgViaNetOrRS232(&msgBuffer);
```

应用 Agent 中的发送消息过程

```
/* 以共享方式得到信号灯当前值 */
GetSemaphoreValues();
message     msgBuffer;
/* 准备欲发送数据 */
PrepareMsg(&msgBuffer);
/* 送入邮筒，由 API 完成的操作 */
P(empty);
P(mutex);
PutMsgIntoQueue(&msg);
V(mutex);
V(full);
```

在上述算法中，P 操作是对信号灯值减1，如减1后的值小于0，则不做减1操作并挂起调用进程，直至此信号灯值大于0；V 操作对信号灯值加1。P 操作和 V 操作均为原子操作。信号灯 mutex 用于互斥，解决对临界资源的冲突问题，避免出现数据不一致现象；empty 和 full 用于控制生产者和消费者的同步，使得消费者在缓冲池空时被挂起，而生产者在缓冲池满时被挂起。

在消息传递代理中间件系统中，信件也即消息的格式被设计为统一的格式：

```
typedef struct rmsg{
    int serialNo;   /* 序号，系统调试用 */
    int type;   /* 消息类型 */
    string src;   /* 发信 Agent 名称 */
    string dst;   /* 目的 Agent 名称 */
    virtualTime sendTimeTag;   /* 发送时间标签 */
    VirtualTime recvTimeTag;   /* 收到时间标签 */
    char msgBody[MSG_LEN];   /* 无格式数据流，装载
                                客户消息数据 */
}rmsg_t;
```

发信 Agent 根据消息中的目的 Agent 名称在 Agent 列表中找到相应 Agent 所在节点，将消息通过网络送至相应节点机上的收信 Agent，由它分发给本机的 Agent。对于广播类消息则发送给所有节点机。如果目的 Agent 在本地，则无需通过网络，直接送至本地信箱。

Agent 列表

Agent	所在主机 IP 地址	本地标志位
Vison2D	210.32.132.160	0
Fusion	210.32.132.200	1
Plan	210.32.132.163	0
……	……	……

由于在 IRASO 体系结构中存在两大类消息，收信 Agent 的行为略为复杂。综合考虑系统稳定性、实时性及实现的效率，消息被细分为三类：一是普通数据流消息，通常是应用 Agent 工作流程中必需部分。如在自主机器人的流水线工作方式下，融合 Agent 在每个工作周期中必需得到二维视觉 Agent 传送过来的数据流方可进行下一步工作；二是事件类消息。通常由公告板系统作出形势评估后发给相关 Agent 的事件信息，以控制整个系统的行为，它是不定期的，并需要相应 Agent 立即响应；三是定期广播的数据流消息，它通常是定位系统发出的当前位置和姿态信息。随时

刷新，并可以被所有 Agent 在需要定位数据时随机访问。针对三类消息，收信 Agent 分别给出一般信箱、紧急信箱、布告板三种机制来实现消息的分发。

信箱机制大体上是邮筒机制的逆过程，其区别在于存在多个消费者。为此，收信 Agent 根据 Agent 列表中本地 Agent 个数分配多个逻辑上不同的消息队列缓冲池，以构成多组一对一的生产者/消费者模型。紧急信箱机制是收信 Agent 在得到事件类消息后将消息送入消息缓冲池内，并向目的 Agent 发出中断信号，目的 Agent 响应中断，读出消息，再调用预先注册的目的 Agent 的事件处理函数来处理。布告板机制相应简单一些，在收信 Agent 得到广播类的数据流时，用新的数据流覆盖原有消息，完成刷新操作，本地应用 Agent 对这一类临界资源的读取只需做互斥访问即可。

3.2 客户端的应用 API

客户端的 API 是中间件的必需要素之一，它用来向中间件通知客户端进程的请求，使中间件完成相应的服务。应用 API 隐藏了中间件的实现细节和机制，只提供少量必需的服务请求，简化了客户端的编程。在 API 保持不变的前提下，中间件实现算法的改动如算法优化之类并不影响客户端软件的功能，也就是说，基于中间件的客户端软件成为"软插件"。

在 IRASO 体系结构中，消息传递代理提供了 C 语言和 C++语言两种 API，以方便客户端应用 Agent 的使用。应用 Agent 是一些分布于异质平台上的应用程序，有些是用 C 语言编写，有些用 C++编写。为 C 语言提供的 API 如下

1）int OpenRmsg(char *agentName) 初始化操作，向消息传递代理注册应用 Agent 名称，取得各种共享资源，如邮筒、信箱、紧急信箱、布告板及用于实现生产者/消费者模型的信号灯等。

2）void CloseRmsg(void) 关闭操作，释放所占有的共享资源，断开与消息传递代理的联系。

3）int SendRmsg(rmsg_t msgBuffer, int flag) 发送消息。参数 flag 表明是否阻塞，如为阻塞方式，则在邮筒满时挂起，直至邮筒进入非满状态；如为非阻塞方式，则在邮筒满时立即返回，返回值为-1，表示发送失败。

4）int RecvRmsg(rmsg_t* msgBuffer, int flag) 从信箱或布告板中读取消息，消息 msgBuffer 中的参数 type 决定消息读取的位置，参数 flag 表明是否阻塞，只对从信箱读取时有效，因为布告板中的消息是刷新定位数据，读取后立即返回。在阻塞方式下，如信箱为空，则挂起直至信箱有新消息到；非阻塞方式在信箱为空时，返回失败。

5）void RegistEventDealFunc(void(*handleOfEventDealFunc)(rmsg_t*msgBuffer)) 注册应用 Agent 中处理事件类消息的函数，通常在初始化操作之后进行。

针对 C++语言提供的应用 API 是一个通用基类，与纯应用 Agent 类共同导出运行于分布式环境下的"活"的 Agent 类，如下图所示：

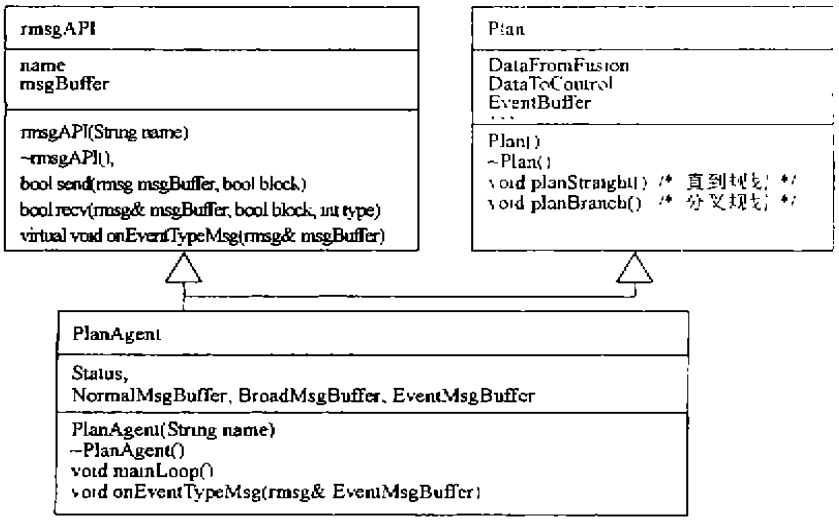

图4 C++API 示意图

4 实验结果

基于邮局模型的消息传递代理中间件系统已在自主车辆体系结构仿真中得到应用，分别在 SGI 的 IRIX、Sun 的 SunOS 及 Microsoft 的 Windows NT 操作系统中开发了相应的软件，连接异质平台，形成统一的分布式计算环境。实验结果表明，系统的稳定性和实时性均达到预期的效果。表1和表2分别列出了中间件在网络传输及客户端应用 Agent 的消息响应时间。

表1 网络传输时间(单位：毫秒)

	O2↔Sun	Sun↔Pentium	Pentium↔O2
1K 字节	1.034	1.273	1.227

表2 客户端 Agent 消息响应时间(单位：毫秒)

	O2工作站	Sun Sparc 工作站	Pentium 166/NT 平台
阻塞被唤醒	0.140	0.960	0.253
事件类消息	0.250	1.906	0.379

其中阻塞被唤醒的响应时间是指客户端进程在因信箱为空被挂起后又因为新消息到来而被唤醒的时刻与消息传递代理将此消息放入信箱时刻的差值；事件类消息的响应时间则是指客户端进程调用事件类消息函数时刻与收信 Agent 收到事件类消息时刻的差值。

网络传输时间是在带宽为10Mbps 的以太网中测试得到，每条消息的大小为1K 字节。在这里，串口通讯的时间没有给出，这是因为串口通讯是点到点的，不存在传输冲突问题，其速度与串口的传输波特率基本吻合。

上述实验结果表明，消息传递代理系统保证了 I-RASO 体系结构中的 Agent 实现实时通信，去除了异质平台给通信带来的影响，构成了方便系统集成的分布式计算环境。

参 考 文 献

1 Shoham Y. Agent Oriented Programming. Artificial Intelligence，1993，60

2 Decker K S Distributed problem-solving techniques. A survey. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 17(5)

3 Sahota M K Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In: Proc of 12th National Conf on Artificial Intelligence, AAAI'94

4 Sunderam V S, et al. The PVM concurrent computing system: evolution, experiences and trends. Parallel comput, 1994, 20(4)

5 Object Management Group. The Common Object Broker, Architecture and Specification, Object Management Group, Framingham, Mass, 1998

6 吴春明，张友军，朱淼良. 分布式自主移动机器人集成环境. 软件学报，1997，8(10)

7 张友军，吴春明，朱淼良. 自主式移动机器人流水线调度模型的设计与实现. 电子学报，1998，26(2)

8 Lewandowski S M. Frameworks for Component-Based Client/Server Computing ACM Computing Surveys, 1998, 30(1)

9 Tanenbaum A S. Operating Systems: Design and Implementation, Prentice-Hall International, Inc., 1997

(上接第67页)

最近草案中声明采用单一 SA 的反重放攻击服务将不能应用于多个发送者的环境中，多播安全的实现应该保证接收者不执行序列号的处理和验证。

为此，给出两种重放攻击的保护机制：(1)使用多个 SA，即：所有的 SA 是单一 MSA 的一部分，而且每个发送者具有一个 SA。(2)将反重放攻击保护放到诸如 SAM 等高层模块中。该方案就需要对于多播消息进行应用层成帧。

3.5 IPSec 对于多播包的处理

目前，现有的 IP 协议栈的实现将丢弃任何目标地址为 D 类且协议域不是 UDP 的 IP 包。所以，需要对其加以改进以便支持由 IPSec 保护的 IP 多播包。

结束语 从分析目前 IP 多播通信的安全需求出发，本文在对 IPSec 安全机制进行深入研究的基础上，讨论了一种安全 IP 多播体系结构框架，标识了其基本组成部件及功能，并说明了这些部件之间以及与运行环境之间的操作关系。进而，结合我们在 IPSec 上的研究成果，将该体系结构框架在 Linux 平台上进行了设计与实现。实验表明，该体系结构框架扩展性强，具有简单性，灵活性，及与现有系统易融合的特点。

参 考 文 献

1 Kent S. Security Architecture for the Internet Protocol. RFC2401, Nov 1998

2 Kent S. IP Authentication Header. RFC2402, Nov. 1998

3 Kent S IP Encapsulating Security Payload (ESP) RFC2406, Nov. 1998

4 Harney H. Group Key Management Protocol (GKMP) Specification. RFC2093, July 1997

5 Harney H Group Key Management Protocol (GKMP) Architecture RFC2094, July 1997

6 Ballardie A Scalable Multicast Key Distribution. RFC 1949, May 1996

7 Mittra S. Iolus: A Framework for Scalable Secure Multicast In: Proc. of ACM SIGCOMM'97, Cannes, France, Sep. 1997