国 防 科 技 大 学 学 报
第 23 卷 第 3 期 JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY Vol.23 No.3 2001

文章编号：1001-2486（2001）03-0077-06

基于分布对象的高性能异步回调模型*

张小明，吴泉源，贾焰

（国防科技大学计算机学院，湖南 长沙 410073）

摘 要：分布对象技术是分布异构环境下软件开发和系统集成的良好解决方案，然而在高性能分布计算领域（如分布事务处理、分布交互仿真和分布并行计算），传统调用模型在异步特性方面的不足限制了分布对象技术在该领域的应用。因此，文章介绍一个新的基于分布对象的异步回调模型，着重探讨了该模型的定义、关键实现技术、优化策略和性能测试。

关键词：分布对象；异步性；回调

中图分类号：TN311 文献标识码：A

High-Performance Asynchronous Callback Model Based on Distributed Object

ZHANG Xiao-ming，WU Quan-yuan，JIA Yan

（College of Computer，National Univ. of Defense Technology，Changsha 410073，China）

Abstract：Distributed object technology is a good solution of software development and integration，but traditional invocation models of distributed object restrict the application of distributed object technology in high-performance distributed computing field（eg. Distributed transaction processing，distributed interactive simulation and distributed parallel computing）due to their lack of asynchronism. So，this paper presents a new asynchronous callback model based on distributed object and mainly discusses the definition，key implementation technology，optimization strategy and performance test.

Key words：distributed object；asynchronism；callback

随着计算机网络和软件产业的迅速发展，分布对象的思想和技术标准（CORBA[1]）日渐被人们接受。目前，分布对象作为分布异构环境下软件开发和集成的良好解决方案[2]，在制造、电信、金融、信息以及交通等各个领域得到了广泛应用。

分布对象的调用模型是客户运用分布对象技术开发分布式应用的基本通讯模式，对于一般的client/server应用，分布对象的传统调用模型基本能满足需要，但对于高性能分布式应用（如分布事务处理、分布交互仿真以及分布并行计算），传统调用模型则由于在异步特性方面的不足，限制了分布对象在高性能计算领域的应用。

因此，研究异步的分布对象调用模型成为学术界和企业界共同关注的一个热点。目前，国际上关于这方面的研究正处于起步阶段[3]。在国内，分布对象技术也已成为公认的开放式分布软件或系统集成框架所遵循和采纳的标准技术，但目前还没有一个软件平台或环境宣布支持异步的调用模型。

1 传统调用模型

在分布对象环境（如CORBA）中，传统上存在三种调用模型[1]：同步方法调用（SMI）、延迟同步调用（DSI）和单向调用（oneway）模型。为了更好地说明问题，给出这几种调用模型的形式定义。

定义1 分布对象环境 Ω 是一个四元组（P，ORB，O，M）。

· P 是客户程序，它由一个操作序列 $op_1op_2op_3\cdots op_n$ 构成。其中，任一操作 op_i 可以是本地操作，也可以是远程对象操作（或调用）。

· O 是远程目标对象，接受并服务于来自客户的请求。

* 收稿日期：2001-02-22
基金项目：国家863基金资助项目（863-306-ZD02-01-2）
作者简介：张小明（1971-），男，博士生。

· ORB是P和O之间的对象请求代理，起着通信中介的作用。

· M是消息集合，可表示为M = {m_1，m_2，…，m_k}，它包括P和O通过ORB传递的所有请求和应答。每个m_i（$m_i \in M$）的生命期都是从P开始，经ORB到O，然后再从O经ORB返回P，结束。当m_i从P经ORB传递给O时，m_i包含的是请求消息；当m_i从O经ORB返回P时，m_i包含的是应答消息。为了描述一个消息的生命期，P在发出每条消息的时候，都为它们分配了一个标识，我们用下标来表示。消息在传递过程中，内容可以由请求消息转变为应答消息，但消息的标识是不会改变的。为了保障标识的唯一性，也为了便于不同的消息进行比较，消息标识定义为时间的函数，即若P先发消息m_i，后发消息m_j，则一定$i < j$。

定义2 若op_i是远程操作，则op_i定义为一个动作序列a_1（m_{k1}）a_2（m_{k2}）…a_n（m_{kn}），（m_{k1}，…，$m_{kn} \in M$）。其中，动作a_i（m_{ki}）表示该动作对标识为k_i的消息m_{ki}进行处理，且a_i（m_{ki}）∈{S（m_{ki}），R（m_{ki}）}，其中S（m_{ki}）表示发送一个标识为k_i的消息给ORB，R（m_{ki}）表示从ORB接收一个标识为k_i的消息。

定义3 远程操作op_i属于SMI调用模型，当且仅当op_i = S（m_{ki}）R（m_{ki}），$m_{ki} \in M$。

定义4 远程操作op_i和op_j（$j > i$）属于DSI调用模型，当且仅当op_i = S（m_{ki}），且op_j = R（m_{ki}），$m_{ki} \in M$。

定义5 远程操作op_i属于oneway调用模型，当且仅当op_i = S（m_{ki}），$m_{ki} \in M$。

SMI模型采用了严格同步的方式处理请求消息和应答消息，即发出请求消息的操作一定要等待并接收到对应的应答消息，才能返回。SMI模型的好处是，当操作返回时，客户程序能够无歧义地确认请求已被目标对象服务并取得相应处理结果。但是，这种模型的不足是系统资源利用率低，在应答返回之前，客户程序不得不处于等待状态，无法处理其它操作，这对于许多高性能的应用是不能接受的。

在实现上，针对SMI模型的一个改进是引入多线程机制，即以不同的子线程加载SMI操作，当子线程等待应答消息的时候，主线程继续处理其它操作，这在一定程度上可以改善系统资源利用率。但是，线程的使用又带来了新问题。首先，过多线程的使用会降低系统的可伸缩性。当SMI操作的数目比较大时，众多的线程本身将占用系统大量的宝贵资源，同时线程之间的切换和管理也为系统增加额外的开销，这些将使系统性能大大降低。其次，在目前的分布对象标准（如CORBA）中，线程属于未被标准化的部分，因此线程的使用还存在可移植性的问题。

DSI模型放松了SMI模型的同步要求，它允许操作向ORB提交请求消息之后立即返回，并在以后的任意时刻，客户程序再通过一个新的操作从ORB读取应答消息。与SMI模型相比，DSI模型有利于系统资源利用率的改善，在提交请求消息和接收应答消息的操作之间，客户程序可以穿插处理一些其它操作。但是DSI的问题是，应答消息的接收需要客户程序启动一个新的操作，这个新操作的每次执行都会导致一次额外的应用层和ORB层之间的上下文切换。更重要的是，在接收消息的操作已启动，但应答消息还没有返回的情况下，接收消息的操作将不得不等待。这就又回到了SMI模型所碰到的基本问题上，即DSI模型本质上仍然没有解决同步带来的问题。

基于DSI模型的一个变种是，一个发送请求消息的操作可以对应多个接收应答消息的操作。如果第一个接收应答消息的操作启动时，不能获得应答消息，则立即返回。同理，如果第二个接收应答消息的操作启动时，不能获得应答消息，也立即返回。以此类推，直到某个接收应答消息的操作成功取得应答消息。其中，接收应答消息的操作之间可以穿插处理其它操作。这种模型有利于回避同步引发的系统资源利用率低的问题，但消息接收操作数的增加也使应用层和ORB层之间的上下文切换次数和开销上升，从而影响系统性能。

oneway调用模型是对前述模型的简单化，具有一定的异步性，即操作向ORB提交请求消息后即返回。但是，这种模型没有给应答消息的处理做出定义，客户程序仅通过oneway调用模型无法得知请求消息是否被目标对象服务以及服务结果如何。因此，对于许多应用（包括高性能应用），oneway调用模型由于太简单而不能胜任。

为了克服 oneway 调用模型的问题，有人提出了双向 oneway 的调用模型，即当目标对象服务完请求消息后，也通过一个新的 oneway 操作，把应答消息返回客户。虽然双向 oneway 解决了应答的接收问题，但是它对目标对象却提出新的要求，即目标对象必须通过一个显示的 oneway 操作返回应答消息，这使得以前面向 SMI 或 DSI 模型所编写的目标对象的服务程序不能直接服务于双向 oneway 调用模型，也就是说双向 oneway 调用模型带来了新的可移植性问题。

综上所述，传统调用模型存在着各种各样的问题，使得它们不能有效支持客户程序和目标对象之间的异步行为，这不利于分布系统内在并行性的挖掘，因而限制了分布对象在高性能分布式应用（如分布事务处理、分布交互仿真以及分布并行计算[3]）领域的应用。

2 异步回调模型

针对传统调用模型在异步性上支持不足的问题，我们基于分布计算软件平台 StarBus 实现了一个遵循 CORBA 消息规范的异步回调模型。下面，我们给出该模型的形式定义。

首先，客户程序 P 的定义在异步回调模型中得到扩充，新的定义是：

定义 6 客户程序 $P = \{op_1op_2op_3\cdots op_n, o_1, o_2, o_3, \cdots, o_m\}$。其中，操作序列 $op_1op_2op_3\cdots op_n$ 的定义与定义 1 一致。o_1，o_2，o_3，…，o_m 表示驻留在客户程序中的一组回调对象。

其次，远程操作和动作的定义也得到扩充，增加了一种新的异步远程操作和相应动作，即

定义 7 若 op_i 是异步远程操作，则 $op_i = A(m_{ki}, o_{ji})$，$m_{ki} \in M$，$o_{ji} \in P$。其中，动作 $A(m_{ki}, o_{ji})$ 表示客户把标识为 k_i 的消息 m_{ki} 和标识为 j_i 的回调对象 o_{ji} 一起提交给 ORB。

最后，我们给出异步回调模型的完整定义。

定义 8 远程操作 op_i 属于异步回调模型，当且仅当 $op_i = A(m_{ki}, o_{ji})$，$m_{ki} \in M$，$o_{ji} \in P$，且 $m_{ki} \in C_{ORB}$。其中，C_{ORB} 表示 ORB 以回调方式处理的消息的集合。所谓回调方式是指 ORB 在处理应答消息时，把应答消息作为一个特殊的请求消息，调用前面异步操作 $A(m_{ki}, o_{ji})$ 所提交的回调对象 o_{ji}。

根据以上定义，异步回调模型展现出如下一些重要特征：

(1) 异步性：异步回调模型规定，客户程序向 ORB 提交请求消息和回调对象后，即可返回去处理其它操作。当应答消息返回时，ORB 以类似事件触发的形式把应答消息转发给回调对象。这使得客户程序在任何时刻都无需等待 ORB 返回应答，克服了 SMI 和 DSI 模型的同步约束，因而具有很好的异步性。

(2) 高效、简洁：与 DSI 模型相比，异步回调模型不要求客户程序显示地从 ORB 读取应答消息，因而减少了客户程序和 ORB 不必要的上下文切换次数，有利于提高系统性能。同时，也正是由于客户程序没有读取应答消息操作，使得客户程序更简洁。

(3) 可移植性：异步回调模型通过 ORB 对回调对象的激发来处理应答消息，一方面克服了 oneway 模型过于简单而无法处理应答的问题；另一方面，与双向 oneway 模型相比，异步回调模型不对目标对象的服务程序做出特别要求，即它与 DSI 和 SMI 使用的目标对象服务程序是一致的，因此它具有很好的可移植性。

(4) 可伸缩性：异步回调模型通过类似事件触发的回调方法实现对应答消息的异步处理，它对线程没有依赖性，因而，与线程化的 SMI 模型相比，异步回调模型不存在由于客户线程增加而引发的系统负荷增加的问题，也就是说，异步回调模型具有很好的可伸缩性。

可见，异步回调模型克服了传统几种调用模型的不足，提供了一种完善的异步性，它有利于分布式系统内在并行性的挖掘，提高系统性能。因此，我们基于分布式软件平台 StarBus 实现了该模型，下面将从实现的角度，探讨其中的一些关键技术和优化策略。

3 关键实现技术和优化策略

3.1 总体结构

StarBus 是一个遵循当前分布对象标准（即 CORBA）的分布式软件平台，它采用层次化的开放体

系结构。基于StarBus，我们实现的异步回调模型主要由如下部分构成：Client、回调对象、回调Skeleton、回调POA（可移植对象适配器）、ORB内核和异步Stub。其中，Client是发出请求消息的客户程序，它为回调对象提供了生存空间。回调对象是应答消息的处理对象，封装了client应用提供的应答处理逻辑。当client应用程序发出请求消息时，它会把请求消息连同回调对象一起提交给异步stub。

异步stub是Client应用程序和ORB内核之间的连接纽带，它给client应用提供了一个强类型的静态调用接口，能够对应用层的请求参数进行类型安全检查，并负责把这些参数编码成公共数据表示的格式（CDR）[1]。

回调POA协助ORB内核把应答消息作为新的请求而绑定到回调对象，它的任务包括：接收回调对象的注册、激活回调对象（如果需要）、对象适配、与回调skeleton合作调用回调对象的方法。在实现上，回调POA通过一个活跃对象表登记用户创建的所有活跃的回调对象skeleton以及回调对象标识。在对象适配时，回调POA根据请求中的对象标识在活跃对象表中进行查询，并与匹配对象的skeleton合作完成应用层方法的调用。

回调skeleton负责把应答消息的字节流从CDR格式解码为对应用层有意义的具体类型参数，然后调用应用层中回调对象的适当方法。

ORB内核是消息传递的通信骨干，它负责把client的请求消息传递给目标对象，并接收返回的应答消息。在实现系统中，ORB采用了异步的方式处理应答消息，即它把返回的应答消息作为一个新的请求消息，激发回调对象上相应方法，而不是像SMI或DSI模型中的ORB被动等待客户程序来读取应答消息。为了保证互操作性，ORB遵循GIOP/IIOP协议[1]，该协议规定了ORB进行消息传递的内容、格式和交互方式等。

进一步，ORB内核的系统组成包括：事件发生器（Reactor）、客户引擎（GIOPClient）、引擎管理器（ClientManager）等。其中，GIOPClient遵守GIOP/IIOP互操作协议[1]，实施和保证ORB的互操作，如GIOP/IIOP消息报文的发送、接收、打包、拆包以及相应GIOP连接的管理等。ClientManager负责GIOPClient的创建、删除和复用等管理，并接受外部（如异步stub）对GIOPClient的查询。Reactor为ORB内核提供了一种基于事件的驱动模型，它能够把系统感兴趣的信号或状态变化转化成为一些事件（如网络事件、时间事件等），然后通过激活或触发对应的事件处理器来处理事件。比如，GIOPClient就是一个重要的网络事件处理器。

3.2 “报文重组”协议

在实现系统中，GIOPClient把应用层和异步STUB提交的目标对象指针、请求操作名和参数编码字节流，进一步打包成请求报文。请求报文一般包括两类信息：一类是报文信息，如GIOP/IIOP标志、协议版本号、报文类型、报文大小以及字节的高低次序等；另一类是请求信息，如请求标识、对象标识、请求的操作名以及参数编码字节流等。

类似地，GIOPClient处理的应答报文一般也是由两类信息组成：报文信息和应答信息。但不同的是，应答信息一般不包括对象标识和操作名。这是由于传统调用模型往往利用同步线程保存现场环境，因而能够直接接收和处理应答报文，无需进行对象或操作的匹配。但是消息回调模型却不存在类似的同步线程，它只能通过对象适配和操作匹配来处理应答，因此它需要对象标识和操作名。为了解决这个问题，有人提出扩充应答报文的办法，即在应答报文中增加回调对象和操作名的有关内容，但这会破坏传统调用模型和消息回调模型之间的互操作性。所以为了既保证消息回调模型的对象回调，又维护互操作性，我们定义了“报文重组”协议。其内容是：

①在请求报文发出前，GIOPClient在回调表（表1）中登记请求标识、回调对象标识和操作名；

②服务器按照与对待传统调用模型一致的方式处理请求包文，即把请求标识转移到应答报文中。

③Client收到应答报文后，GIOPClient根据其中的请求标识，从回调表提取对应的回调对象标识和操作名，然后把这些信息与应答报文进行重组，构造一个以回调对象为目标的等价“请求报文”；

④GIOPClient把这个特殊“请求报文”提交给回调POA及回调skeleton，进行对象及操作的匹配、解码和分发。

表 1 GIOPClient 的回调表成员

Tab.1 The member of callback table of GIOPClient

请求唯一标识	回调对象唯一标识	回调操作名
0	KEY0	Query
1	KEY1	Query
2	KEY2	Draw

“报文重组”协议巧妙利用了客户端 GIOPClient 的回调表保存了有关回调信息，没有对目标对象所处的服务器提出额外要求，因此保证了与传统调用模型之间的互操作性。

3.3 混合优化策略

在消息回调模型中，迅速、及时地处理应答对许多高性能应用至关重要，因此我们对对象回调的关键路径进行了反复的优化。

一般地，对象回调的关键路径主要包括两个层次的匹配工作：①回调 POA 根据回调对象标识实施对象适配，定位并引发回调 skeleton；②回调 skeleton 根据方法名实施匹配和分发。它们被统称为层次化的复用分解（DD）[9] 策略。在 DD 的实现上，我们最初采用了简单顺序查找法，其最坏代价为 O（n^2）。经过优化，我们对 POA 的对象适配采用了 Hash 技术以及线形搜索的冲突解决办法，而对回调 skeleton 的方法匹配采用二分法技术，结果最坏代价变为 O（$n\log_2 n$）。为进一步提高性能，我们最后优化出一种更加高效的轻量级复用分解（LD）策略。按照 LD，GIOPClient 在处理应答时，跨过 DD 策略的两个匹配层次，利用回调表预留的回调对象指针和回调方法编号，直接提取和调用回调对象上指定方法，其最坏代价仅为 O（1）。

从时间开销上，LD 是最理想的策略，但是仅有 LD 策略还是不够的。这是因为，当回调对象被减活时，只有 POA 才能把回调对象激活，而 LD 由于跨过了 POA 的对象适配，因而无法激活对象。也就是说，DD 策略仍然需要保留。所以，最后我们把两种策略有机结合起来，提出了所谓的混合策略，即：在正常情况下，模型选择 LD 策略处理应答消息；而当回调对象处于不活跃状态时，模型选择 DD 策略。在一般情况下，混合策略能利用 LD 快速复用分解的优势，大大缩短应答处理时间，改善系统性能。

3.4 事件处理的轮转法策略

Reactor 管理事件的产生、注册、注销等。其中，注册/注销的内容称为网络事件处理元组，它们由网络地址、网络事件类型和事件处理对象构成。GIOPClient 在发送和接收报文时，要向 Reactor 注册或注销事件处理元组。Reactor 能够同时监听多路网络事件：如果某处理元组注册的网络事件类型为“读”，且对应网络地址上有数据到达，则“读”事件发生；如果事件类型为“写”，且对应网络地址允许数据发送，则“写”事件被引发。这些网络事件一旦发生，Reactor 就启动处理元组中相应处理对象。

Reactor 在“读/写”事件的处理上，会碰到两种极端情况：一个极端是“写”事件优先，即请求报文优先发送，但是当请求报文源源不断地产生时，应答报文将总是得不到处理，即被“饿死”；另一个极端是“读”事件优先，即应答报文被优先处理，但这又可能导致请求报文不能及时被发出，浪费服务器资源。为了解决这个问题，我们提出一种折中的轮转法策略，其算法是：

① Reactor 检测网络事件。如果“读”、“写”事件仅发生其中之一，那么转②或③；如果“读”、“写”事件同时发生，转④。

② Reactor 处理“读”事件，启动 GIOPClient 处理一个应答报文。转①。

③ Reactor 处理“写”事件，启动 GIOPClient 发送一个请求报文。转①。

④ Reactor 优先处理“读”事件，启动 GIOPClient 处理一个应答报文。接着，Reactor 处理“写”事件，启动 GIOPClient 发送一个请求报文。转①。

在算法的④中，“读/写”两类事件被交替处理，即每处理一个应答报文将总是伴随着一个请求报

文的发送。这种策略避免了两个极端情况，体现出一定的公平性。

4 结语

我们在实验室建立起一个基于 StarBus 的测试环境。在该环境中，client 是发出远程请求的客户程序，server1 和 server2 是支持目标对象的两个远程服务器。在配置上，client 采用 Pentium III/500MHz/128M 内存/Windows NT4.0，server1 采用 Pentium III/500MHz/128M 内存/Redhat linux 5.1，server2 采用双 CPU 的 Celeron/400MHz/128M 内存/Redhat linux 5.1，网络环境采用 100Mbps 的以太网。

测试主要衡量的性能指标是 client 的平均延迟（Latency）。对 CORBA 系统而言，延迟是指从 client 开始调用本地操作开始，到 client 收到该操作返回值为止所经历的时间，并减去目标对象所执行操作的时间。

图 1 给出了单机和 client/server1/server2 分布式测试的结果（按照 CORBA 标准，我们对 DSI 模型的测试是通过动态调用接口（DII）实现的），其中 MCB 与 MCB ' 分别代表消息回调模型采用混合策略前后的结果。测试结果表明：(1) 在单机上，SMI 作为最简单的调用模型，其平均延迟最低；(2) 在分布环境中，MCB 能够更大程度地开发网络资源和系统内在并行性，平均延迟最小；(3) 混合优化策略能够进一步改善 MCB 性能。

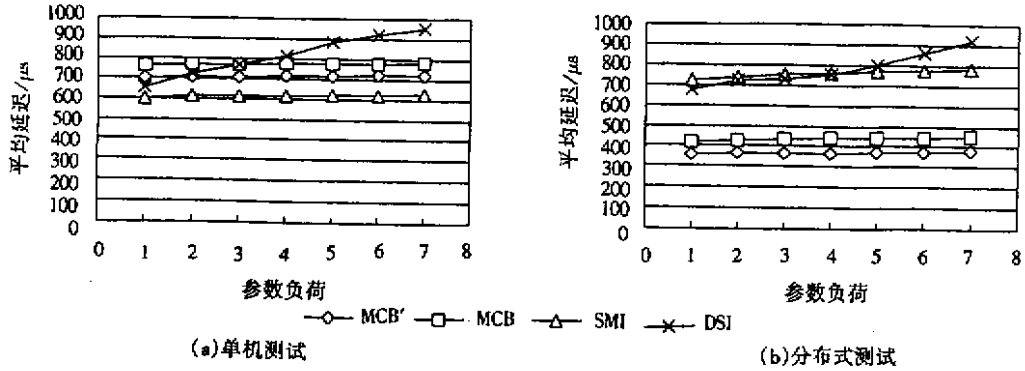

图 1 测试结果

Fig.1 The test result

参考文献：

[1] Object Management Group. The Common Object Request Broker：Architecture and Specification [CP]，2.3 ed. June 1999. http：//www.omg. org.

[2] Vinoski S，CORBA ：Integrating diverse applications within distributed heterogeneous environments [J]. IEEE Communication Magazine，1997，14 (2)：46 – 55.

[3] Arulanthu A B. O 'Ryan C，Schmidt D C. The design and performance of a scalable ORB architecture for CORBA asynchronous messaging [C]. In：ACM/IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing，New York，2000. http：//www. cs. wustl. edu/ ~ schmidt/ami2. ps. zip.

[4] 张小明，王怀民，吴泉源. 高性能虚拟分布对象 [J]. 计算机研究与发展，2000，37：102 – 107.

[5] 周立，张小明. 对象请求代理 YHCSBroker/ORB 的研究与实现 [J]. 计算机学报，1997，20：83 – 89.

[6] Object Management Group. CORBA Messaging Specification [CP]，May 1998. http：//www. omg. org.