Ice 协议的形式化分析*)

杨小刚　沈曾伟

（北京航空航天大学软件开发环境国家重点实验室　北京 100083）

摘　要　Ice 中间件是一个和 COBRA 同样强大却摒除了 COBRA 的缺陷的分布式对象平台，它为异构环境下的应用开发提供了一种新的方式。Ice 协议定义了客户和服务器通信的规则。本文使用 pi 演算描述分析 Ice 协议，从交互过程和协议实体两方面对协议进行了刻画，揭示其并发、分布的对象计算特征。

关键词　Ice，pi 演算，协议压缩

The Formal Analysis of Ice Protocol

YANG Xiao-Gang　SHEN Zeng-Wei

（State Key Lab. of Software Development Enviroment, Beihang University, Beijing 100083）

Abstract　Ice is the object-oriented middleware platform and is as powerful as CORBA without making all of CORBA's mistakes, it provides a new way to develop the application in heterogeneous environment. Ice protocol defines the communication regulation between client and server. This paper gives a formal analysis of Ice protocol from two sides, interaction and protocol entity, which highly embody the parallel and distributed feature of Ice.

Keywords　Ice, Pi-calculus, Protocol compression

Ice(Internet Communication Engineer)中间件是由 ZeroC 公司推出的新兴的分布式对象产品[1,2]，它在保留 CORBA 的基本体系结构的基础上，吸取了 CORBA 的精华，剔除了 CORBA 的缺点，提供了一个简单易用的、具有更好性能和可扩展性的分布式计算平台。

Ice 提供了构建客户/服务器应用的工具、API 和库的支持，支持在异构环境下的开发，客户端和服务器端可以使用不同的开发语言，Ice 内核处理了网络通信的细节，使得开发者只需要专著于自己的业务细节。

Ice 协议由 3 个主要部分组成：一组数据编码规则；一些在客户和服务器之间进行交互的消息类型；一组协调客户和服务器之间特定的协议和编码版本的规则。Ice 协议有以下一些特点：(1)支持在线路上进行压缩，从而节省带宽。(2)适用于构造高效率的事件转发机制，消息交换机不需要对消息进行任何编码或解码，它们可以简单地把消息当作不透明的字节缓冲区加以转发。(3)Ice 协议适用于构建双向操作：如果服务器想要把一条消息发送给客户提供的某个回调对象，这个回调对象可以通过客户原来创建的连接传给服务器。如果客户在防火墙后面，这种特性尤其重要，它提供了一种应用穿越防火墙的方式。(4)Ice 支持 TCP、UDP 和 SSL3 种传输协议。TCP 和 UDP 在局域网环境下传输效率极高，SSL 又增加了通信的安全性。

pi 演算是 R. Milner 等人在 CCS(Calculus of Communication System)基础上提出的传名演算[3~5]，可描述分布和并发系统。其计算实体中只有名字和进程，允许在进程之间传递和接收通道名。目前，对 pi 演算的研究有两大方向：一是从数学角度对该演算进行研究[3~5]；二是研究它在探讨语言解释与定义实际问题的能力[6~8]。

本文旨在用 pi 演算对非形式化描述的 Ice 协议进行形式化描述，揭示其并发、分布和协作的对象计算特征。首先对于几个重要的交互过程，如连接、断开连接、协议压缩、请求调用等给予详细的刻画；其次以同步请求调用过程为例，刻画了在调用过程中各协议实体的功能。

1　Ice 协议架构概述

Ice 协议基于典型的 C/S 结构。其基本结构如图 1 所示，可以对等地分为 4 层：应用层、消息层、编码/解码层和传输层。在客户端：客户应用负责发起调用请求(该请求包括 Ice 对象标识、接口、操作和参数等)，或者接收返回的响应结果；消息层负责将调用请求格式化为 Ice 规定的标准的消息结构，或者将标准的消息转化为响应结果；编码/解码层负责按照 Ice 所规定的编码/解码格式，对于消息数据进行整编/解编；传输层负责对于整编后的数据在线路上传输。在服务器端：服务应用负责接收、分派调用请求，并返回调用结果，其它层的功能与客户端对应层功能类似。

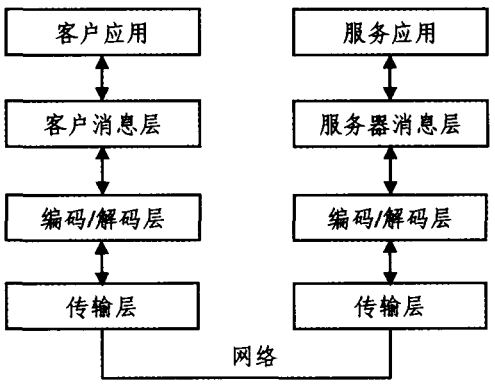

图 1　Ice 协议的基本结构

*)国家 973 资助项目(No. G1999032709)。**杨小刚**　博士生，研究方向为网络管理、移动计算等；**沈增伟**　硕士生，研究方向为网络管理、移动计算等。

2 pi 演算

本节简单介绍一下 pi 演算的语法和流图。

2.1 pi 演算的语法

设 P,Q 表示进程，x,y 表示名，则 pi 演算进程可定义为：

$P::=0|\alpha(x).P|\bar{\alpha}(x).P|\tau.P|P+Q|P|Q|(x)P|[x=y]P|A(y_1,y_2,\cdots,y_n)$

其意义解释如下：

0：表示非活动进程，它不能与任何进程交互。

$\alpha(x).P$：表示先通过通道名 α 从通道接收名 y，替换 x，再激活进程。

$\bar{\alpha}(x).P$：先通过通道名 α 向通道的另一端输出名 x，再激活进程 P。

$\tau.P$：τ 是内部动作，外部不可见。

$P+Q$：进程不确定计算，或者执行 P，或者执行 Q。

$P|Q$：进程 P 和 Q 并发执行。

$(x)P$：变量 x 是 P 的内部变量。

$[x=y]P$：条件进程，若 $x=y$，则激活进程 P，否则 0。

$A(y_1,y_2,\cdots,y_n)$：进程标识符，对于每个进程标识符来说，必须有其定义 $A(x_1,x_2,\cdots,x_n)::=P$，其中 $x_1,x_2,\cdots,x_n$ 是形参。

关于进程的自由名和受限名的定义请参看文[5]。在 $\alpha(x).P$ 和 $(x)P$ 中，x 是 P 的受限名；在 $\bar{\alpha}(x).P$ 中，x 和 a 是 P 的自由名。

2.2 流图

利用 pi 演算建模经常使用流图[3,4]来表示进程之间的连接关系，以及它们与外界的连接点。流图一般包含进程、端口和通道等元素，如图 2。

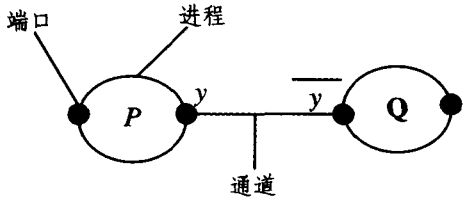

图 2 pi 演算流图

3 Ice 消息描述

Ice 使用了 5 种标准的协议消息：请求、批请求、答复、验证连接、关闭连接。其中请求和批请求是从客户到服务器的，答复是从服务器到客户的，验证连接和关闭连接只适应于面向连接的机制，批请求只适应于没有应答的单向请求。

定义 1 定义 Ice5 类协议消息分别为

requestmsg，*batchrequestmsg*，*replymsg*，*validateconnmsg*，*closeconnmsg*。

Ice 协议消息由一个 14 字节的消息头和消息体组成。消息头包含协议版本号、编码版本号、消息类型、消息的压缩状态、消息尺寸等信息。其中压缩是 Ice 的一个可选特性，取决于映射语言是否支持压缩、代理是否要求压缩、低于 100 字节的消息不压缩等若干因素。压缩时使用 bzip2 算法对消息体进行压缩，消息头不压缩。消息头的 compressionStatus 域有 3 种压缩状态值，规定了消息是否进行了压缩，以及发送者是否接收压缩的消息。其压缩状态值集合定义如下。

定义 2 压缩状态值集合为

$ComStatus=\{c0,c1,c2\}$

其中 $c0$ 表示消息未压缩，消息发送者也不能接收压缩答复。当消息接收方接收到这样的请求时，也将答复请求压缩状态设为 $c0$。$c1$ 表示请求消息未压缩，但是服务器可以返回压缩消息。$c2$ 表示发送的消息是压缩的，而返回的消息可以是压缩的，也可以是不压缩的。

请求消息体包含调用某个对象所必需的数据，包括对象的标识、操作名以及输入参数等。

批请求消息体包含有一个或多个单向请求，出于效率而捆绑在一起。批请求包括对象标识、对象接口、操作名、调用模式、调用参数和调用上下文等。

答复的消息体包含一个双向调用的结果，包括返回值、输出参数或者异常。答复消息的前 4 个字节是请求 ID，后面跟着请求的状态，状态后面是相应状态的描述。Ice 答复有 8 种状态，定义其答复状态集如下。

定义 3 Ice 答复状态集为

$ReplyStatus=\{r0,r1,r2,r3,r4,r5,r6,r7\}$

其中，$r0$ 表示请求成功。$r1$ 表示用户异常。$r2$ 表示目标对象不存在。$r3$ 表示接口不存在。$r4$ 表示操作不存在。$r5$ 表示未知的 Ice 本地异常。$r6$ 表示未知的 Ice 用户异常。$r7$ 表示未知的异常。

验证连接的消息指服务器在收到新连接时会发送该消息，它只适用于面向连接的消息。该消息表明服务器已准备好接收请求。客户无需对这种消息发送答复。验证连接消息的作用有两个：(1)发送服务器端的协议版本和编码版本。(2)防止客户在确认能实际处理请求之前把消息写到本地缓冲区中，避免消息丢失。验证连接消息只有消息头，没有消息体，所有的信息都包含在消息头中。验证连接消息的状态总是 0。

关闭连接消息指通信某一方要关闭连接时，就发送该消息。该消息也只适用于面向连接的传输。整个关闭连接消息只有消息头，没有消息体。关闭连接的压缩状态为 0。连接的任何一方都可以根据自己的意愿发起关闭连接。

4 协议刻画

本节利用 pi 演算对于 Ice 协议进行建模和刻画。首先从交互的角度，对于协议中几个典型过程进行刻画和描述；其次以一次请求调用为例，从协议实体的角度，对协议进行完整的刻画。

4.1 交互过程的刻画

Ice 协议包含系列交互过程，如连接过程、关闭连接过程、消息压缩应答、请求调用过程等，下面分别给予刻画。

4.1.1 连接过程的刻画

客户端与服务器端建立连接的过程如下：客户端向服务器端发起调用请求，服务器端收到请求后，如果连接成功，则向客户端发验证连接消息，否则发连接异常消息。客户端收到正常答复消息，则进程继续，否则终止当前进程。为了描述这个过程，除了客户进程和服务进程，还需要一个定时器进程，用于处理连接超时问题，定时器由客户端在发起连接时启动。

设 x 为客户和服务器端的通道，y 是客户端和定时器通道，*client* 是客户端进程，*server* 是服务器端进程，T_0 是定时器进程。*connectrequest* 是客户端发出的连接请求消息，*replymsg* 是服务器返回的答复消息，$s\in Replystatus$ 是答复消息中的状态域。*stmsg* 是客户进程启动定时器消息，*timeoutmsg* 是定时器发向客户进程的超时消息。*stmsg* 和 *timeoutmsg* 不是标准的 Ice 协议消息，属于自定义消息。

令 $\overrightarrow{cp}=\{x,y,connectrequest,replyrequest,timeoutmsg\}$，它是客户进程的自由名；服务进程的自由名为 $\overrightarrow{sp}=\{x,connectrequest,replymsg\}$；超时进程的自由名为 $\overrightarrow{tp}=\{y,st\ msg,timeoutmsg\}$；则连接过程各进程可表示如下：

$client(\overrightarrow{cp})\triangleq(\bar{x}(connectrequest)|\bar{y}(stmsg)).(x(reply\ msg)([s=r0].client+[s\neq r0].0)+y(timeoutmsg).0)$

$server(\overrightarrow{sp})\triangleq x(connectrequest)\bar{x}(replymsg).server$

$T_o(\overrightarrow{tp})\triangleq y(stmsg).T_o+\bar{y}(timeoutmsg).0$

连接过程由上述三个进程并发组成。

$client(\overrightarrow{cp})|server(\overrightarrow{sp})|To(\overrightarrow{tp})$

4.1.2 关闭连接过程的刻画

关闭连接可以由客户或者服务器任何一方主动发起。下面分别予以讨论。

(1)客户端发起的关闭连接过程

客户端发起的关闭连接过程如下：(1)客户发送关闭连接消息。(2)客户关闭连接的写端。(3)服务器关闭连接的读端，对于客户关闭连接消息给予响应。(4)服务器关闭连接的写端。

在关闭连接的客户端，有 3 个进程实体(如图 3)：关闭连接客户进程 Clp，写端进程 $writer$，读端进程 $reader$。$writer$ 接收 $client$ 的关闭写端消息 $closewmsg$ 或者打开写端消息 $openwmsg$；$reader$ 接收 $client$ 的关闭读端消息 $closermsg$ 或者打开读端消息 $openrmsg$。u,v 分别是 Clp 与 $writer$ 和 $reader$ 的通道。

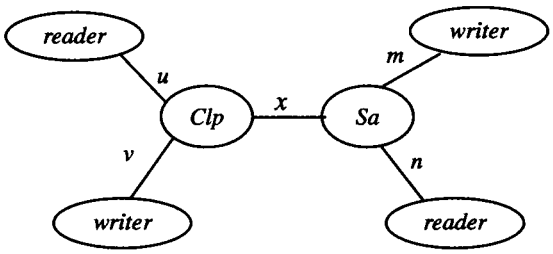

图 3 关闭连接流图

令

$\overrightarrow{cp}=\{x,u,v,closeconnectionmsg,replymsg\}$，

则

$Clp(\overrightarrow{cp})\triangleq\bar{x}(closeconnctionmsg).writer(u,closewmsg).x(replymsg).reader(v,closewmsg).client$

$writer(u,closewmsg,openwmsg)\triangleq\bar{u}(closewmsg).writer+u(openwmsg).writer$

$reader(v,closermsg,openrmsg)\triangleq\bar{u}(closermsg).reader+u(openrmsg).reader$

关闭连接的服务端与客户端类似。

令

$\overrightarrow{sp}=\{x,m,n,closeconnection,replymsg\}$，

则

$Sa(\overrightarrow{sp})\triangleq x(closeconnectionmsg).reader(u,closermsg).\bar{x}(replymsg).writer(v,closewmsg).server$

(2)服务器端发起的关闭连接过程

服务器端发起的关闭连接的过程如下：(1)丢弃连接上的所有请求，也就是关闭连接的读端。(2)等待仍在执行的请求完成，并返回结果。(3)服务器向客户端发送关闭连接消息。(4)服务器关闭连接的写端。(5)客户端关闭连接的写端和读端，对服务器关闭连接做响应。

服务器端进程表示如下：

$Sa\triangleq reader(u,closermsg).waitforresult(t,waitmsg).\bar{x}(closeconnectionmsg).writer(v,closermsg).server$

其中，$waitforresult$ 进程接收 $server$ 的 $waitmsg$ 消息，等待仍在执行的请求，直到所有的请求执行完，自动退出。t 是 $server$ 和 $waitforresult$ 进程的通道。

$waitforresult\triangleq\bar{t}(waitmsg).waitforresult$

客户端进程表示如下：

$Clp\triangleq x(closeconnectmsg)\bar{x}(replymsg).writer(u,closewmsg).reader(v,closermsg).client$

4.1.3 消息压缩的应答

Ice 支持消息的压缩，这是 Ice 区别于 Corba 的重要特性之一。通过设置一个配置参数，可以使网络通信数据被压缩，这在客户和服务器端需要交换大量的数据时候非常有用。服务器端是否返回压缩消息，取决于客户端请求消息的压缩状态，其规则如第 3 节所述。设 $c\in ComStatus$ 是客户端端请求消息的压缩状态，则消息压缩请求应答过程中进程刻画如下：

令

$\overrightarrow{cp}=\{x,requestmsg,replymsg\}$，

则

$client(\overrightarrow{cp})\triangleq\bar{x}(requestmsg).x(replymsg).client$

$\overrightarrow{sp}=\{x,requstmsg,replymsg\_c0,replymsg\_cl\}$

其中 $replymsg\_c0,replymsg\_c1$ 分别为压缩消息和非压缩消息。则：

$server(\overrightarrow{sp}\triangleq x(requstmsg).([c=c0]\bar{x}(replymsg\_c0)+[c=c1](\bar{x}(replymsg\_c0)+\bar{x}(replymsg\_cl))+[c=c2](\bar{x}(replymsg\_c0)+\bar{x}(replymsg\_c1))).server$

4.1.4 请求调用过程的刻画

Ice 的客户请求调用方式有 5 种：同步、异步、单向、数据报和批处理方式。同步调用：在调用期间，客户线程被挂起，并在调用完成时恢复；异步调用：客户调用除传递通常的参数外，还要传递一个回调对象，客户调用完成后立即返回，不必等待调用结果，回调对象负责接收调用结果；单向调用：客户应用只发出调用要求，服务器不返回应答；数据报调用：与单向调用类似，客户端只发出调用要求，不要求服务器响应，与单向调用所不同的是它的传输机制要求是 UDP 而不是 TCP；批处理方式：包括成批单向调用和成批数据报调用，它将多个调用先在客户端缓冲起来，然后用一条消息送出，这种方式的好处在于调用效率高。

定义 4 定义 CS 为调用方式集

$CS=\{sychromous,asychromous,oneway,datagram,batched\}$

其中 $sychromous,asychromous,oneway,datagram,batched$ 分别表示同步、异步、单向、数据报、批处理方式。

设 $\overrightarrow{cp}=\{x,c,smsg,srmsg,amsg,armsg,omsg,dmsg,bomsg,bdmsg\}$，

其中 c 表示回调通道，$smsg$ 表示同步消息，$srmsg$ 表示同步响应消息，$amsg$ 表示异步消息，$armsg$ 表示异步响应消息，$omsg$ 表示单向消息，$dmsg$ 表示数据报消息，$bomsg$ 表示批处理单向消息，$bdmsg$ 表示批处理数据报消息。$s\in CS$ 是调用方式，则客户端请求调用进程可表示如下：

$Clp(\overrightarrow{cp})\triangleq$
$[s=synchromous]\bar{x}(smsg).x(srmsg)+$
$[s=asychromous]\bar{x}(amsg).x(c).c(armsg)$
$+[s=oneway]\bar{x}(omsg)+[s=datagram]\bar{x}(dmsg)$
$+[s=batched]([t=batchoneway]\bar{x}(bomsg)+$
$[t=batchdatagram]\bar{x}(bdmsg)).Clp$

(下转第 246 页)

(2)详细设计并实现多机并发系统通信模型的分层抽象。

(3)最后通过系统的实验研究,验证了该方法的有效性。

目前仅从系统、节点、进程层次对多机并发系统通信结构进行恢复,如何处理多线程并发问题是下一步的研究重点。

参 考 文 献

1 Chikofsky E J, Cross J H II. Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, 1990, 7(1): 13~17

2 Kiczales G, Lamping J, Mendhekar A, et al. Aspect-Oriented Programming. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), June 1997

3 Liu M L. Distributed Computing Principles and Application. 清华大学出版社, 2004. 4

4 陈平. 反射结构和对象标识的研究:[博士学位论文]. 西安电子科技大学, 1991

5 李青山,陈平,王伟,宋海鸿. 逆向工程中反射植入的研究. 计算机学报, 2004, 4

6 Li Q S, Chen P. XDRE 2.0 Reference Manual Software Engineering Institute. Xidian University, 2004

7 杜宽利. 多机通信结构图抽取与状态图抽象的研究:[硕士学位论文]. 西安电子科技大学, 2005

(上接第 242 页)

令 $\overrightarrow{sp}=\{x, c, smsg, srmsga, msg, armsg, omsg, dmsg, bomsg, bdmsg\}$,

则服务器端相应的进程表达为:

$$Sa(\overrightarrow{sp})\triangleq([s=synchrmous]x(smsg).\bar{x}(stmsg) + [s=asychromous]x(amsg).x(c).c(armsg) + [s=oneway]x(omsg) + [s=datagram]x(dmsg) + [s=batched]([t=batchoneway]x(bomsg)+ [t=batchdatagran]x(bdmsg)).Sa$$

4.2 利用协议实体对于协议进行刻画

本节对 Ice 协议进行完整的刻画。说明在一次请求调用的过程中,各协议实体如何协同地工作,从而反映 Ice 协议分布、并发的特性。如图 4,分别定义客户端应用层、消息层、编码/解码层、传输层进程为 Clp, Cm, Cc 和 Ct;服务器端对应层进程为 Sap, Sm, Sc 和 St。在一个请求调用过程中,各协议实体的进程代数刻画如图 4。

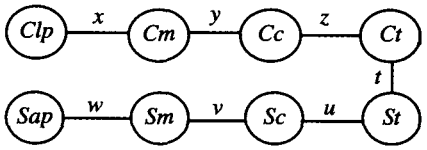

图 4 Ice 协议的流图

令 $\overrightarrow{cp}=\{x, requestmsg, replymsg\}$,则

$Clp(\overrightarrow{cp})=\bar{x}(requestmsg).x(replymsg).Clp$

Clp 进程沿 x 通道发出调用请求 $requestmsg$,沿 x 通道接收响应 $replymsg$。

令 $\overrightarrow{cmp}=\{x, y, reqestmsg, replymsg, msg, rmsg\}$,

则

$$Cm(\overrightarrow{cmp})=(x(reques\ tmsg).\bar{y}(msg) + y(rmsg).\bar{x}(replymsg)).Cm$$

Cm 进程沿 x 通道接收请求 $requestmsg$,组合成 Ice 的协议消息 msg,沿 y 通道送出去;或者接收沿 y 通道送来的消息 $rmsg$,转换成应答消息 $replymsg$ 沿 x 通道回送给 Clp。

令 $\overrightarrow{ccp}=(y, z, msg, marshalingmsg, unmarshalingmsg\}$,

则

$$Cc(\overrightarrow{ccp})=(y(msg)\bar{z}(marshalingmsg)+ z(marshalingmsg).\bar{y}(unmarshalling)).Cc$$

Cc 进程主要负责消息的编码/解码工作,沿 y 通道接收原始消息 msg,编码成 $marshalingmsg$ 消息并沿 z 通道送出去;或者接收来自 z 通道的编码消息 $marshalingmsg$,解码成 $unmarshalling$ 消息,并沿 y 通道送出。

令 $\overrightarrow{ctp}=\{z, t, msg\}$,则

$Ct(\overrightarrow{ctp})\triangleq(z(msg).\bar{t}(msg)+t(msg).\bar{z}(msg).Ct$

Ct 进程主要负责在服务器和客户端之间传输消息。沿通道 z 接收消息 msg,并沿 t 通道传输到服务器端;或者沿 t 通道接收消息,并沿 z 通道传回。

整个客户端的行为如下:

$client(t, msg)\triangleq Clp|Cm|Cc|Ct$

同理可以描述服务器端各进程,本文不再赘述。

结论 本文利用 pi 演算对于 Ice 协议建模,从交互和协议实体两方面刻画了协议。pi 演算着眼于并发进程之间传递的消息,通过描述消息交换来说明系统成分与环境之间的通信行为。它以直观、透明的方式反映通信行为,省略了传统的形式描述技术必须考虑的大多数诸如程序变量以及变量值这样的细节,特别适用于说明通信协议。同时 pi 演算是基于严格代数基础的形式化方法,利用 pi 演算描述 Ice 协议,消除了理解上的二义性,而且为进一步的演算提供了形式化基础,可以在此基础上讨论 Ice 协议的活动性与安全性问题。

参 考 文 献

1 Henning M. A new approach to object-oriented middleware. IEEE Internet Computing, 2004, 8(1): 66~75

2 Henning M, Spruiell M. Distributed programming with ICE. www.zero.com. 2003

3 Milner R, Parrow J, Walker D. A Calculus of mobile process, parts I and II. Information and Computation, 1992, 100: 1~77

4 Milner R. The polyadic π-calculus: a tutorial:[Technical report]. Department of Computer Science, University of Edinburgh, 1991

5 Milner R. Communication and mobile systems: the pi-Calculus. Cambridge, Cambridge University Press, UK: 1999

6 Park J, Miller R. A compositional approach for designing multifunction time-dependent protocol [A]. In: Proc. of 5th Intl. Conf. on Network Protocol. Los Alamitos, USA. IEEE Computer Society, 1997. 105~112

7 Carbone M, Nielsen M. A formal model for trust in dynamic networks. In: Proc. of the Software Engineering and Formal Methods, SEFM'03. IEEE Computer Society Press, 2003

8 Albert T R, Esterline C. Using the pi-Calculus to model multiagent systems. Lecture Notes in Computer Science, 2001, 1871: 164~179

9 焦文品,史忠植. 形式化多主体系统中的交互及交互协议. 软件学报, 2001, 12(8): 1177~1182