Cogent 后组装技术研究与实现

胡海洋,杨 玫,陶先平,吕 建

(1. 南京大学计算机软件新技术国家重点实验室,江苏南京 210093;2. 南京大学计算机软件研究所,江苏南京 210093)

摘 要: 构件组装技术是构件软件的核心技术之一.本文分析了当前构件组装机制的现状及几个有代表性的构件组装机制的不足之处,然后提出了一种基于移动 Agent 技术的构件组装技术,它利用后组装技术弥补了传统连线机制中构件组装机构改动困难、构件装配欠灵活的弱点.其方法是通过提出的组调用表与定位表的概念,实现构件功能体与组装机制的分离,各自独立开发与编译.最后本文通过一个实例说明了 Cogent 构件后组装技术的特点.

关键词: 移动 Agent; 构件组装; 构件

中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12-1823-05

Research and Implementation of Late Assembly Technology in Cogent

HU Hai-yang, YANG Mei, TAO Xian-ping, LÜ Jian

(1. *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu* 210093, *China*;
2. *Institute of Computer Software Nanjing University, Nanjing, Jiangsu* 210093, *China*)

Abstract: Component assembly technology is one of the core technologies of component software. This article analyzes the status of current component assembly mechanism, pointing out the weaknesses of some representative assembly mechanisms, and then presents a new component assembly technology based on mobile agent. It uses the late assembly technology to overcome such defects of traditional component assembly mechanism as being difficult to modify and inflexible. Relying on the Group Table and Location Table presented, we implement the separation of the functional body and the assembly mechanism, developing and compiling them independently. Last, an example was introduced in the article to illustrate the component late assembly technology in Cogent.

Key words: mobile agent; component assembly; component

1 引言

基于构件的软件设计已成为软件工程研究与开发实践所关注的重点[1].构件软件技术搭建的系统具有良好的适应性、灵活性和易维护性,能有效的支持软件复用[2,3],因而受到越来越多的软件研究与开发人员的关注,国内的研究人员也对此作了大量的工作[2~5].构件软件技术的基本思想是首先开发出一组软件构件,每个构件都提供了特定的功能,并将功能的实现隐藏在良好定义的接口之后,然后,应用某种连线机制,将一些构件以一定的方式组装起来以形成一个完整的软件系统.由此可见,作为将构件组合起来的"粘合剂",连线组装机制在构件软件技术中占有很重要的基础地位.

构件组装技术思想由来已久,从较早的 MIL[6] 到现在基于面向对象的 CORBA[7]、DCOM[8] 和 JavaRMI[9] 等,及其他实验系统[10~12].在分步式网络环境下,从 OSI 参考模型的层次划分,构件连线组装机制可抽象为三级[13]:低级、高级和中间件.低级连线如 Socket[14] 需要用户将过多的精力花在与应用逻辑无关的交互细节上,高级连线机制如 Erlang[15] 则通常与某一特定的编程环境结合在一起,不具有通用性,因此低级连线与高级连线在使用上都有着一定的局限性.而 CORBA、DCOM、JavaRMI 等构件组装技术是基于面向对象思想的中间件技术,它们对网络作了恰当的抽象,并提供了丰富的服务,因此现在较为流行.由于 CORBA、DCOM 和 JavaRMI 都是基于经典的客户机/服务器模型,其在自主性、灵活性、主动性等网络环境的适应性方面有一定欠缺;同时,它们过早的确定构件连接机制,且连接机制与功能代码编译在一起,当加入新的连接信息时,需改写源码、重新编译,从而造成应用程序开发效率低下、构件连接机制不灵活,这些使得它们不适合分布式环境下构件动态配置的要求.移动 Agent 技术为解决上述问题提供了良好的基础[16~19].基于以上分析,我们自行研制开发了构件框架及其支撑系统 Cogent,在其中提供了一种基于移动 Agent 的新型构件组装技术 - Cogent 后组装技术,来实现构件功能体与构件连接机制的分离,使得用户能灵活的进行构件的组装,从而满足分布式环境下构件动态连接配置的需要.

收稿日期:2002-01-14;修回日期:2002-05-12

基金项目:国家自然科学基金(No.60273034);863 高科技项目(No.2001AA113110, No.2002AA116010);江苏省自然科学基金和高技术项目(No.BG2001012, BK2002203, BK2002409)

本文在介绍了 Cogent 后组装基本框架的基础上，主要讨论 Cogent 系统中后组装机制的实现技术，主要包括组调用表机制、定位表机制、后组装机制编译器以及一个应用实例.

2 Cogent 后组装机制框架

后组装是构件软件技术的核心概念，它是指构件连线与构件功能体的分离，由不同"卖方"各自独立开发构件功能体，再由使用者将这些部分组装起来使用，而组装过程中无需改变原功能代码[20]. 后组装机制可分为静态和动态两种. 在静态后组装中，需在应用运行之前手工地将各构件组装起来，这样形成的应用的结构是封闭的，在运行时是固定不变的. 动态后组装则是指在运行中，根据请求和服务接口的匹配关系，将满足要求的构件组装起来，这样形成的应用具有开放的结构，可以在运行时根据环境的变化动态地调整.

在 Cogent 后组装结构框架中(图 1)，为了实现灵活、方便的构件后组装，将构件功能体与构件连线机制分开，一个完整的 Cogent 构件应用程序被设计成三部分组成：构件功能体、构件接口和构件组装机制，各自独立开发、分别编译. 其中，构件的功能体是构件所提供的服务接口的具体实现；构件接口是构件之间交互的协议；构件的组装机制描述了构件的连线机制即构件的执行逻辑和服务构件的位置信息，它又被进一步设计成描述构件执行逻辑信息的组调用表(GroupTable)和描述服务构件位置信息的定位表(LocationTable)[16]两部分，这种执行逻辑与定位信息的分离有助于实现分布式环境下构件的重定位与构件的可替换，且支持组装的灵活性和复用性. 在调用服务构件时，移动 Agent 技术被用来取代传统构件连接机制，其优势在于利用移动 Agent 自主性、移动性、协作性和安全性特点来克服传统构件连接机制中的对网络适应性不强及执行效率不高的弱点[16].

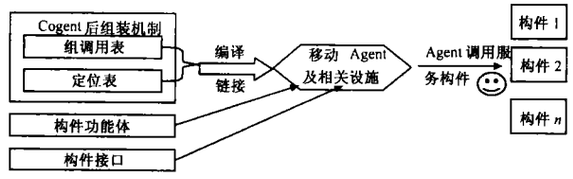

图 1 Cogent 后组装机制框架结构

为了让移动 Agent 准确无误的按照构件组装机制中的描述信息去执行调用服务构件，后组装机制编译器是必需的，它将构件组装机制自动编译生成相应的移动 Agent 代码，这样移动 Agent 将依照构件组装机制中提供的服务构件的执行逻辑信息和位置信息去连接服务构件，即去相应的目标节点进行服务构件的调用.

图 2 中给出了一个典型的 Cogent 后组装机制的工作过程. 构件应用程序需要完成一组图画的绘制、打印和装订. 存在着三个服务构件 Com1、Com2 和 Com3，它们分别提供了绘制、打印和装订功能，在程序执行过程中，Com1、Com2 和 Com3 通过移动 Agent 连接(如图 2 实线所示). 在程序运行中，系统发现有一服务构件 Com4 能够提供更高质量的打印服务，而且提供装订功能的服务构件 Com5 恰好与 Com4 在同一节点上，此时为了使用能提供较好打印服务的 Com4 和 Com5，Cogent 系统仅需修改定位表中的服务构件位置信息，而无需改动原先构件功能体，便能得到新的构件连线(如图 2 虚线所示)，

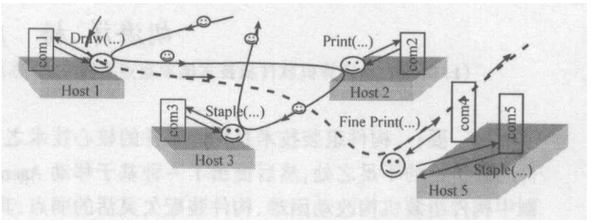

图 2 后组装机制工作过程

在 Cogent 系统中为了方便准确的定义和描述组调用表和定位表结构，一种描述语言 GLDL(GroupTable and LocationTable Definitation Language)，被设计用来对组调用表和定位表进行语法定义与描述. 下面在讲述组调用表和定位表的结构与内容时，将用 GLDL 语言给出它们的定义.

3 组调用表与定位表

参考文献[16]中给出了组调用表与定位表的设计思想，本文的重点在于给出组调用表与定位表的具体实现技术及 GLDL 语言的语法定义. 组调用表用于描述移动 Agent 的执行流程，Agent 将根据组调用表中的描述信息迁移至提供服务构件的目标节点完成请求并返回结果. GLDL 语言定义的组调用表结构如下(图 3)：

```
# Group file
[GroupName]〈标志符〉;
[Parameter]〈参数列表〉;
[PrivateData]〈变量声明〉;
[StructureType]SEQ|PAR|SLOOP|PLOOP;
[GroupBody]组调用表表体定义;
[After_Processing]〈后处理定义〉;
[Condition]〈布尔表达式〉;
```

图 3 组调用表的结构(GLDL 语言)

```
# Location File
[LocationName]〈标志符〉
[GroupName]〈标志符〉
[LocationName]〈定位表表体〉
```

图 4 定位表的结构(GLDL 语言)

[GroupName]描述了组调用表的表名. [Parameter]是组调用表进行服务构件调用时所需的参数列表，参数列表的 BNF 语法结构图为：〈参数列表〉::=〈参数模式〉〈参数类型〉〈参数标志符〉|;〈〈参数模式〉〈参数类型〉〈参数标志符〉〉|. 参数列表中的参数模式项用于表示 Agent 与服务构件进行交互时，所需的参数的输出输入及调用后相应结果的返回，参数模式有三种：IN、OUT 和 INOUT. [PrivateDate]是组调用表的私有数据区，它是新的数据类型、临时变量和命名常量定义之处. [StructureType]给出了以何种方式来连接服务构件，参照传统程序设计语言的执行控制方式定义了四种连接方式：顺序(SEQ)、并行(PAR)、循环(SLOOP)和并行循环(PLOOP)，它们表达了可能的构件连接方式. [GroupBody]给出了组调用表的表体内容，包含构件指引、接口方法和后处理定义三个方面. 构件指引指出了具体提供服务的构件的实例，接口方法给出了方法所属的接口，后处理定义是对调用后的结果作适当的处理. [GroupBody]的实际内容是随[StructureType]的连线方式

的不同而稍有变化的,具体表现为:

(1)[StructureType]为 SEQ 或 SLOOP 时

〈GroupBody〉::=〈顺序表体项列表〉

〈顺序表体项列表〉::=〈指引名〉;〈接口方法调用〉;〈后处理定义〉|;〈指引名〉;〈接口方法调用〉;〈后处理定义〉;|

(2)[StructureType]为 PAR 或 PLOOP 时

〈GroupBody〉::=〈并行表体项列表〉

〈并行表体项列表〉::=〈指引名〉;〈接口方法调用〉|;〈指引名〉;〈接口方法调用〉;|

〈接口方法调用〉::=〈接口名〉〈方法名〉|〈实在参数表〉|&〈方法型构说明〉

〈方法型构说明〉::=〈Mode〉〈类型〉|;〈Mode〉〈类型〉|

[After_Processing]给出了方法调用后的相应处理,由于在顺序(SEQ)与循环(SLOOP)的连线方式中,后处理已包含在组调用表的表体中([GroupBody])中,因此对于这两种连线方式而言,[After_Processing]项为空.[Condition]是一个布尔表达式,它给出了循环(SLOOP)和并行循环(PLOOP)的中止条件,而这一项对于连线方式([StructureType])为顺序(SEQ)和并行(PAR)是无意义的.

定位表用于描述服务构件(包括构件对象的初值)所在的位置信息.移动 agent 正是根据定位表的信息来决定流动的目标节点.为了清晰的描述定位信息,定位表中需要描述以下的信息,即"主机"、"构件"、"实例"、"接口"和"方法"."主机"指出了构件在网络中的位置,"构件"指定了提供服务的具体构件,由于每个构件可以同时有多个实例运行,"实例"指出具体请求哪一个构件实例的服务,一个构件可同时实现多个接口的服务,"接口"指定了应该哪个接口中的"方法"被执行.下面给出了定位表的 GLDL 语法结构(图 4):其中,[LocationName]表示本定位表表名.[GroupName]表示与本定位表相关联的组调用表名.[LocationBody]表示本定位表的内容,其具体语法形式如下:〈定位表表体〉::=|〈指引名〉,〈节点地址〉,〈构件名〉,〈构件实例名〉|

4　后组装机制编译器

在 Cogent 环境中,后组装机制编译器用于生成移动 Agent 代码并使其严格按照组调用表定义的执行逻辑与定位表所描述的位置信息作相应的迁移与执行,同时对 Agent 进行有效的管理.它实际是一个自动代码生成器,它的作用是根据 GLDL 定义的组调用表和定位表信息,自动生成构件连接的所需代码,所需代码具体而言有 Marshaller、Agent 和 Agent 管理器这三个部分组成.其结构图如图 5 所示.

Marshaller:它是构件功能体和 Agent 管理器间的"粘合剂",其主要功能是生成和启动 Agent 管理器,将构件功能体中组调用的"IN"和"INOUT"模式的参数传递给 Agent 管理器,将 Agent 管理器中得到的"OUT"和"INOUT"结果返回给构件功能体.其实现伪码见图 6.

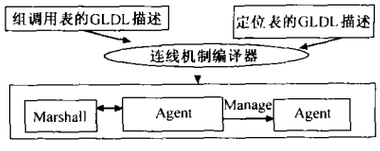

图 5　Cogent 开发环境编译器结构图

```
Marshaller(参数)
{    //创建 Agent 管理对象;
     //完成 IN,INOUT 模式的参数传递,并设置好 Agent 管理器对象相应的实例变量的值;
     //调用 Agent 管理器中的方法,启动其运行;
     //将 OUT 和 INOUT 参数由 Agent 管理器对象相应的实例变量的值中取出并返回给构件功能体;
}
```

图 6　Marshall 的实现伪码

Agent 管理器:它是一种特殊类型的移动 Agent,主要依据定位表和组调用表的信息,完成构造 Agent 的迁移计划的工作,它固定在客户端,专门管理其他 Agent 之责,如生成 Agent、进行参数传递、在 SLOOP 和 PLOOP 结构中判断循环是否继续及收集执行结果.Agent 管理器的实例变量包括在组调用表的接口中的参数及在组调用表中定义的私有数据,这些私有数据可包括新的类型定义、命名常量及用于中间处理的临时变量.Cogent 后组装机制有四种构件连接方式,Agent 管理器也有四种,分别对应于不同的连接方式.图 7 中给出 PAR 连接方式的 Agent 管理器实现伪码.

```
Class PAR-AgentManager( )
{   //组调用表中参数列表的实例变量定义;
    //组调用表中私有数据的实例变量定义;
    public RAR-AgentManager( ){}//构造器}
    public synchronized vokd run( ){
      // 定义要派出的 Agent 个数
      //创建旅行路线为 PAR 的 Agent;
      //将实例变量的值从 Agent 管理器传输至 Agent;
      //启动 Agent;
      // 等待所有的 Agent 返回;
    }
    public ynchronized vokd put-1( ){//将 Agent 1 中的实例变量的值返回给 Agent 管理;}
    .....
    public synchronized put_n( ){// 将 Agent n 中的实例变量的值返回给 Agent 管理器;}
}
```

图 7　Agent 管理器的实现伪码

Agent:Agent 是实际完成构件连接的实体,它会依据 Agent 管理器提供的迁移计划负责迁移到被调构件所在的结点,代表调用构件与被调构件作进程间局部交互、请求其服务,并携带中间状态及最终结果在网络上迁移.Agent 迁移到服务构件所在结点后,先与当地的构件的名服务设施交互,从而得到其所要请求构件的引用,然后实际地提出服务请求.Agent 完成旅行并返回到出发结点后,结果返回给 Agent 管理器.

5　示例

下面将用一个电子商务中的例子来给出 Cogent 后组装机制的实际运行过程.谈判是电子商务中一项比较常见的活动,尤其是在买卖双方进行交易的过程中.在这里将采用 Cogent

构件框架来模拟电子商务中的该活动.前提:网上需要进行交易的站点能提供谈判交易的服务.要求:客户给出自己的商务要求,希望给出结果.Client 端在本地节点 sklab 上,三个服务节点(knuth,polya,hoare),每个节点上均有一计算服务 compute( );

例子过程:

(1)本地机派出 Agent 顺序遍历各节点进行交易

组调用表为:

```
# Group    file
[GroupName]      Quote;
[Parameter]      in String name;inout float price;out String hostname;
[PrivateData]  float returnPrice;float lowestPrice;String lowestHost;
[StructureType]  SEQ;
[GroupBody]
 /* 服务节点 polya 上相应代码 */
  polya_ refVar; CORBAQuoteServer. Hello ( hostname ) &out String;
{lowestHost = hostname;}; polya_ refVar; CORBAQuoteServer. Quote(name,
price,returnPrice)&in String,in float,
  inout float;{lowestPrice = returnPrice;System. out. println("Price is " +
 returnPrice); price = returnPrice/2;}; polya_ refVar; CORBAQuoteServer.
 Quote(name, price, returnPrice) & in String, in float, inout float; {if
 (lowestPrice>returnPrice)lowestPrice = returnPrice;price = lowestPrice * 3/
 4;System. out. println("Price is" + returnPrice);};
 /* hoare 服务节点的交易代码(略) */
 /* knuth 服务节点的交易代码(略) */
[After_ Processing]
[Condition]
```

定位表为:

```
# Location    file
[LocationName]      Quote;
[GroupName]      Quote. gt;
[LocationBody]    polya_ refVar, $ polya, QuoteCOM, polya_ instance;
                  hoare_ refVar, $ hoare, QuoteCOM, hoare_ instance;
                  knuth_ refVar, $ knuth, QuoteCOM, knuth_ instance;
```

(2)派出多个 Agent 同时访问各节点进行交易,即构件连接方式为并行结构.此时定位表不需要修改,只要对组调用表进行改动即可.

并行结构的组调用表:

```
# Group    file
[GroupName]    Quote;
[Parameter]    in String name;inout float price;out String hostname;
[PrivateData]  float return1,return2,return3;String host1,host2,host3;
[StructureType]  PAR;
[GroupBody]    /* polya 服务节点上的交易代码 */
  polya_ refVar;CORBAQuoteServer. Hello(host1)& out String;
  polya_ refVar;CORBAQuoteServer. Quote(name,price,return1)& in String,
in float,inout float;
      /* hoare 服务节点上的交易代码(略) */
      /* knuth 服务节点上的交易代码(略) */
[After_ Processing]  price = return1; hostname = host1; if(price> return2)
{price = return2; hostname = host2;} if(price> return3) {price = return3;
hostname = host3;}};
[Condition]
```

当需增加新的服务节点时,仅需修改定位表即可.由此,可以看出 Cogent 后组装机制在构件动态装配方面的灵活性.

6 结论及相关工作的比较

在分布式计算环境下,构件分布在各处,需要有一连线机制以某种方式将其组装起来以完成特定的功能调用.目前主流的中间件连接组装机制 JavaRMI、DCOM 和 CORBA 等都有着自己的实现机制与方法.基于移动 Agent 技术设计了 Cogent 后组装连接机制,下面将这些连接组装机制作一定的比较(表 1).

表 1 连接组装机制的比较

	计算模式	网络负载情况	对语言的支持	对断开式交互支持	交互过程
JavaRMI	Client/Server	较高	纯 Java 语言	不支持	多次远程交互
CORBA	Client/Server	较高	支持异种语言	不支持	多次远程交互
DCOM	Client/Server	较高	在 Microsoft 平台上运行良好	不支持	多次远程交互
Cogent 后组装	移动 Agent	较低	支持异种语言	支持	通过 Agent 迁移将多次远程交互转为局部交互

从上表的比较中,可以看出基于移动 Agent 技术的 Cogent 后组装机制具有灵活性强、动态调配性能好和对网络环境具有良好适应性的特点.以 Cogent 后组装机制为核心的 Cogent 构件框架及支撑系统体现出其作为构件连线中间件的优势,并已通过专家验收与鉴定.今后的工作包括对 Cogent 连线机制中安全性问题的进一步改进,使之功能进一步完善;对可移动构件框架的设计,使得框架中的构件实现可移动;将构件框架设计成完全面向构件的计算模型,这些都需要进一步的努力.

参考文献:

[1] Gray T Leavens. Foundations of Component-Based System [C]. Cambridge, UK: Cambridge University Press, 2000.

[2] 杨芙清,等.软件复用与软件构件技术[J].电子学报,1999,27(2):68-75.

[3] 梅宏,等.青鸟构件库的构件度量[J].软件学报,2000,11(5):634-641.

[4] 常继传,等.青鸟系统中可复用软件构件的表示与查询[J].电子学报,2000,28(2):20-23.

[5] 顾明,等.构件类和构件的概念及其定义语言和操作语言[J].软件学报,1997,8(9):673-679.

[6] DeRemer, F. Programming-in-the-large versus programming-in-the-small [J]. IEEE Transactions on Software Engineering SE-2, 1976, 2: 321-327.

[7] Object Management Group. CORBA, OMG Website [DB/OL]. 2000. http://www.omg.org.

[8] Microsoft Corporation. Microsoft COM homepage [DB/OL]. 2000.

http://www.microsoft.com/com.

[9] Gosling J, et al. The Java Language Specification[M]. Boston, USA: Addison Wesley, 1996.

[10] Markus Lumpe, et al. A formal language for composition [A]. Foundations of Component-Based System [C]. Cambridge, UK: Cambridge University Press, 2000.

[11] Kevin Lano, et al. Composition of reactive system components [A]. Foundations of Component-Based System [C]. Cambridge, UK: Cambridge University Press, 2000.

[12] Stephen J. Garland, et al. Using I/O automata for developing distributed systems [A]. Foundations of Component-Based System [C]. Cambridge, UK: Cambridge University Press, 2000.

[13] Andrew S Tanenbaum. Computer Networks[M]. New Jwesey, USA: Prentice Hall, 1989.

[14] Richard Stevens. Unix Network Programming, Second Edition, Volume 1. Networking APIs: Sockets and XTI[M]. New Jersey, USA: Prentice Hall, 1998.

[15] Joe Armstrong. Concurrent Programming in Erlang[M]. New Jersey, USA: Prentice Hall, 1999.

[16] 吕建,等.基于移动 Agent 技术的构件软件框架研究[J].软件学报,2000,11(8):1018 - 1023.

[17] 陶先平,等.Mogent 系统的通信机制[J].软件学报,2000,11(8):1060 - 1065.

[18] 张冠群,等.Mogent 系统迁移机制的设计与实现[J].计算机研究与发展,2001,38(9):1035 - 1041.

[19] Lujian, et al. A hierarchical framework for parallel seismic applications [A]. Communications of The ACM[C]. New Work, USA: 2000, 43 (10): 55 - 59.

[20] Clemens Szyperski. Componen Software, Beyond Object-Oriented Programming[M]. Boston, USA: Addison-Wesley, 1997.

作者简介:

胡海洋　男,1977 年出生于江苏宝应.南京大学计算机科学与技术系硕士生.研究方向:构件技术、软件过程技术.

杨　玫　女,1978 年出生于湖南湘乡,南京大学计算机科学与技术系硕士生.研究方向:构件技术、软件过程技术.