突。方法名同义词冲突有两种可能的解决方法：(1)对发生方法名同义词冲突的方法进行换名。即 rename M1 of Cl to M2；或 rename M2 of C2 to C1；(2)建立方法名同义词字典以记载发生冲突的方法名同义词。

在两个相关的类 C1和 C2中若两个含义不同的方法具有相同的名字时就发生了方法名同形异义词冲突。它也有两种可能的解决方法：(1)对发生方法名同形异义词冲突的方法进行换名，即 rename M of C1 to M1；或/和 rename M of C2 to M1；(2)建立方法名同形异义词字典以记载发生冲突的方法名同形异义词。

②方法调用参数冲突。是指含义相同的方法具有不同的调用参数定义，具体的冲突表现有调用参数个数不同和参数的数据类型不同。在两个相关的类 C1和 C2中，若两个含义相同的方法 M1和 M2具有不同的调用参数个数时就发生了方法调用参数个数冲突。在全局概念模式中重新定义一个全局统一的方法。

在两个相关的类 C1和 C2中，若两个含义相同的方法 M1和 M2具有不同的调用参数类型时就发生了方法调用参数类型冲突。在全局概念模式中重新定义一个全局统一的方法。

③方法返回参数类型冲突。是指含义相同的方法具有不同的返回参数定义，即返回参数的类型可能不同。在两个相关的类 C1和 C2中，若两个含义相同的方法 M1和 M2具有不同的调用参数类型时就发生了方法返回参数类型冲突。在全局概念模式中重新定义一个全局统一的方法。

4.3 方法体定义冲突

在两个相关的类 C1和 C2中，若两个含义相同的方法 M1和 M2具有不同的体定义时就发生了方法体定义冲突。它的解决策略如下：(1)如类 C1是类 C2的一个超类，则通过方法提炼的概念对 C2中的方法 M2加以提炼；反之亦然；(2)否则只能在全局概念模式 中重新定义这两个方法。

参考文献

[1] Batini,C. and Lenzerini, M., A Comparative Analysis of Methodology for Database Schema Integration, ACM Computing Surveys, 18(4) 1986

[2] Shoval, P. and Zohn, S., Binary-Relationship Integration Methodology, IEEE Trans. on Knowledge Eng., 6,1991

[3] Kamel, M., Identifying, Classifying, and Resolving Semantic Conflicts in Distributed Heterogeneous Databases: A Case Study, J. of Database Management, 6(1)1995

[4] Spaccapietra, S. and Parent, C., Conflicts and Correspondence Assertions in Interoperable Database, SIGMOD RECORD, 20(4)1991

[5] Kim, W. and Seo, J., Classifying Schemantic and Data Heterogeneity in Multidatabase Systems, Computer, 24(12)1991

[6] Reddy, M.P. et al., A Methodology for Integration of Heterogeneous Databases, IEEE Trans. on Knowledge and Data Eng., 6(6) 1994

(上接第94页)

更是涉及到系统环境的很多方面，所以对其进一步优化，提高其运行效率是一项需要持续坚持的细致工作；3)继续跟踪 ODP 和 CORBA 的发展，使 OSEware 最终能和其它的同类系统进行互操作。

参考文献

[1] 唐雪飞，开放系统中的互操作性研究，电子科技大学博士论文，1996

[2] ISO/IEC JTC 1/SC 21/N 7047, Working Document on Topic 9.1-ODP Trader, 1992

[3] OMG,X/Open, CORBA 2.0 Pre-publication draft, 1995/6/2

[4] OMG, X/Open, The Common Object Request Broker: Architecture and Specification, Revision 2.0, Updated July 1994

[5] W. Rosenberg and Denney, Understanding DCE, Open System Foundation, 1992

[6] J. Shirley, Guide to Writing DCE Application, Open System Foundation, 1992

[7] Dimitri Konstantas, Design Issues of a Strongly Distributed Object Based System, Proc. of 2nd Intl. Workshop for Object-Orientation in Operating Systems(I-WOOOS'91), IEEE, Paloalto,1991

[8] Jack C. Wileden et al., Specification Level Interoperability, CACM, 34(5)1991

[9] James M. Purtilo and Joanne A. Atlee, Module Reuse by Interface Adaptation, Software Practice & Experience, 21(6)1991

[10] E. N. Elnozahy et al., Experience Using DCE and CORBA to Build Tools for Creating Highly-Available Distributed Systems, Intl. IFIP Workshop on Open Distributed Processing, Feb. 1995

计算机科学1998 Vol. 25№. 2

一个基于面向对象技术的互操作中间件

About an Interoperability Middleware based on Object-Oriented Technology

苏 森 唐雪飞 刘锦德

(电子科技大学计算机科学与工程学院 成都610054)

摘 要 In this paper, design consideration and practical implementation of a real interoperability middleware OSEware is introduced, and its main components which involve in providing interoperability between heterogeneous computer systems are clearly described. Those are trader, stub generator and COPL compiler. Besides, the effort made by many researchers to resolve interoperability is presented.

关键词 Trader, Interoperability, Middleware, Open system, Open distributed processing.

当前,分布式计算正向开放式分布处理发展。"开放"意味着系统中可以存在大量的异构实体,如异构硬件平台、异构操作系统、异构语言、异构应用等。互操作性就是异构实体之间的相互协作。作为开放系统领域的一项关键技术的互操作性已经成为当今国际学术界的一个研究热点。许多标准化组织所制定的标准和不少实际的开放式分布处理系统都将其作为一个设计目标,如ODP[2]、CORBA[3]、DCE[4,5]等。

ODP(Open Distributed Processing)是国际标准化组织(ISO)和国际电讯联盟(ITU)联合推出的一个标准。此标准中引入了交易器模型,异构的应用实体可以通过交易器进行互操作。由于ODP标准于1995年才获正式通过,看来从标准到正式的实用系统出现还有一段相当长的距离,因为尚有大量的实现技术需要研究。

OMG(Object Management Group)和X/Open于1995年6月联合推出了CORBA 2.0[3]。与其前身CORBA1.0不同是的是,互操作技术成为CORBA 2.0的重要内容。在CORBA 2.0中,异构的应用实体通过ORB(Object Request Broker)相互协作,不同厂商开发的ORB则通过一组协议进行互操作。这组协议包括:IIOP(Internet Inter-ORB Protocol)、GIOP (General Inter-ORB Protocol)、ESIOP(Environment-Specific Inter-ORB Protocol)。

OSF(Open System Foundation)于1992年推出DCE的初样。作为一个设计,DCE是比较成功的,当时还存在的OSF的对立集团UI(Unix International)也不得不承认这点。但是,实践证明,作为产品的DCE太复杂了,编制一个很小的程序就需要学习很长时间,如果DCE用于建造大型系统,它必须提供另外的工具以简化编程;另外,在使用过程中DCE经常出现瘫痪事故,也令用户难以忍受[9]。

在"八五"和"九五"攻关项目的支持下,我们对互操作技术进行了研究和探索,并且将其作为开放系统环境中的关键技术加以突破,完成了名为OSEware(Open System Environment Middleware)的互操作中间件。当然后者也适用于开放式分布处理系统。本文就是我们的部分研究成果的总结。

1 OSEware的设计思想

作为开放系统的关键技术,互操作性已经得到了较为深入的理论研究[1,6-8]。根据互操作所作用的层次不同,可将解决问题的途径(方案)分为两类:面向过程的互操作方案和面向对象的互操作方案。

面向过程的互操作方案,它的互操作是在过程调用时发生作用的,其实现将借助于传统的RPC。使用此方案的前提是,服务器提供的过程与客户调用的过程必须精确匹配,否则,二者很难进行互操作。

面向对象的互操作方案,它的互操作是在对象级发生作用的。在异构的面向对象的环境中所遇到的问题是,不同对象其界面不同。如果一个对象无法理解另一个对象的界面,它们就无法进行协作。研究面向对象的互操作的基本目的就是解决这一问题。

根据所采用的方法不同，这类互操作方案又可以分为两种。第一种是界面统一化。这是一种比较流行的互操作方案，ODP、CORBA 均已采纳。为了与国际上有影响的系统兼容，OSEware 也采用了此方案。其基本思想可以用图1来表示。

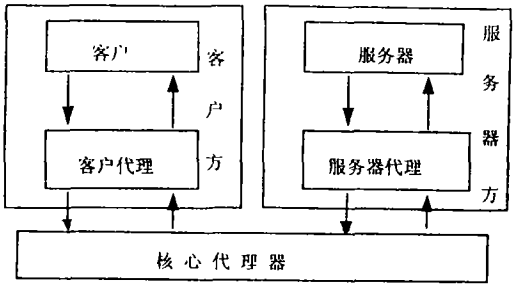

图1 界面统一化方案

通过服务器代理，系统将服务器所能提供的服务在核心代理器所维持的一个信息库中进行登记。客户通过客户代理发出自己的请求，即所要求的服务。核心代理器利用客户提供的信息查找并定位服务器，然后完成客户和服务器之间的连接。在这种方案中，由于核心代理器的存在，客户看到的界面和某个具体的服务器没有必然的联系。这也是在这类系统中能够采用动态调用（即在执行过程中确定目标对象）的基础。这是传统的基于 RPC 的系统（如 DCE）所无法做到的。此处的核心代理器是一个概念性成分，在不同的标准或系统中都有相应的成分与之对应，但一般取不同的名字。如，在 CORBA 中称之为 ORB，在 ODP 中称之为交易器等。

在实际使用中，用户只需用界面描述语言或动态调用例程向系统说明自己所需的或所能提供的服务，系统根据这些信息产生互操作所需的成分。其中客户只关心自己所看到的界面，而不关心目标服务器所在的机器以及相应的操作系统。

第二种是界面桥接。使用界面统一化方法，异构对象虽然可以通过互操作平台进行互协作，但这并没有彻底解决互操作问题。尚存在的问题是，不同的厂商可能会开发出各自的互操作平台，不同互操作平台上的异构对象之间无法进行协作。为解决这一问题，需要引入一组协议，用于传输并转换不同互操作平台上的各域内容，以使不同平台上的对象可以相互协作。CORBA2.0的 IIOP、GIOP 和 ESIOP 就是这样一组协议。

2 OSEware 的实现方案

OSEware 采用客户/服务器结构，并体现着 ODP 的基本特点，如具有互操作性、基于面向对象技术、利用交易器模型等。它易于学习，便于使用，在实践中得到了用户的好评。

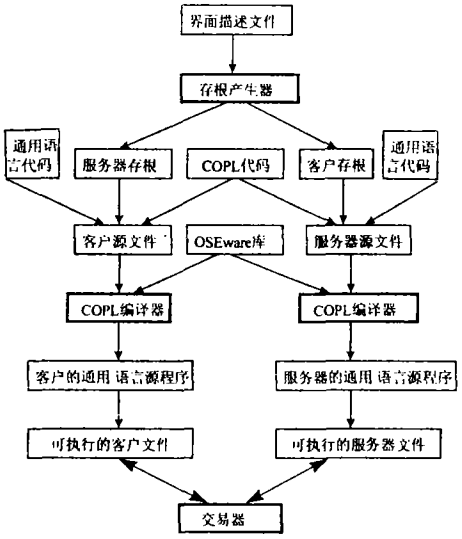

图2 OSEware 的体系结构和一对典型的客户

图2以一个典型的客户实体和服务器实体的生成过程，给出了 OSEware 的实现方案及其体系结构。

OSEware 中有三个与互操作直接相关的重要成分：交易器（Trader），存根产生器（Stub Generator），合作处理语言 COPL（COoperative Processing Languge）编译器。

用户首先用界面描述语言 IDL 根据自己的需求写出界面描述文件。OSEware 中的界面是一个用于说明数据结构和操作的集合。存根产生器作用于界面描述文件后产生客户存根和服务器存根。可以把存根看做是一个编程框架。在此框架的基础上，用户只需在客户对象中向交易器输入自己所请求的操作，服务器对象则向交易器输出自己所能提供的服务，用户不必关心二者如何相互协作。

实际的程序由通用语言源代码（C/C++）和嵌入式 COPL 语句组成。COPL 语句具有三个功能：说明界面类型；创建和取消界面；调用界面实例中的操作。

通过 COPL 编译器，由混合程序生成通用语言源程序。混合程序中的 COPL 语句在通用语言源程序中由 OSEware 的库函数取代。这些库函数的实现是相当复杂的，因为它们既要处理与网络协议有关的细节以及应用对象之间的连接，又要使所有这些对用户都是透明的。所以，COPL 嵌入式语言的引入

大大简化了用户的编程任务。此处需要强调的是，在一般情况下，客户对象和服务器对象是异构的。这也是开发此互操作中间件的本意所在。

3 交易器

本节描述交易器所用的概念和术语都来自文[2]，文中不再特别注释。互操作中间件的一个重要功能是解决客户对象与服务器对象之间的动态连接问题。在OSEwaer中，此功能在交易器中完成。服务器向交易器输出界面，以宣布自己所能提供的服务。客户对象向交易器输入请求，申请自己所需的服务。交易器根据客户的请求查找界面类型空间，匹配之后向客户返回服务对象所提供的界面参考。客户对象一旦拥有界面参考，它就可以调用相应界面中的操作。图3以一个实例说明了客户和服务器的交易过程。

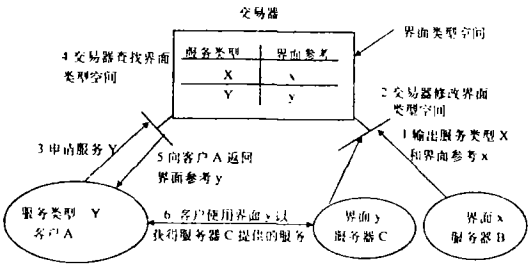

图3 客户和服务器的交易过程

4 存根产生器和COPL编译器

OSEware的界面描述语言和COPL语句的语法和语义详见文献[1]。此处以一个简单的实例说明存根产生器和COPL编译器的功能以及OSEware的使用方法。为了说明简单并不失普遍性，此实例选用C语言作为通用语言。

在此例中，客户需要输出一个矩形窗口。客户对象只需发出一个请求，具体输出窗口的任务将由客户所未知的服务器去完成。此例的界面描述文件如下：

```
Window:INTERFACE=/* 界面类型名 */
{
OPERATION;/* 界面中的操作定义 */
  OutputWindow();/* 输出窗口的操作 */
}
```

假设此文件的文件名为Window.tod，那么将该文件输入存根产生器后，将产生以下主要文件：

```
Window.inf   其名包括COPL编译器所需的信息；
cWindow.c    客户存根文件；
sWindow.c    服务器存根文件。
```

在cWindow.c和sWindow.c中加入COPL语句，分别生成client.cpl文件和server.cpl文件，其结构如图4和图5，其中以"!"开头的语句是COPL语句，其它是原来存根文件的内容。

经过COPL编译器产生两个C语言源程序：client.c和server.c。最后由make工具生成两个可执行文件：client和server(在DOS和Window环境下为client.exe和server.exe)。

OSEware向用户屏蔽了客户对象和服务器对象相互协作所需的一切细节问题，既满足了客户和服务器的互操作要求，又减轻了用户的编程负担。

```
#include "ose.h"
! INCLUDE Window/* 说明即将引用的界面类型 */
! INCLUDE Trader/* Trader是系统公用的一个界面 */
! DECLEAR {Window_server_ifr};Window SERVER
/* 说明Window界面的界面参考为Window_server_ifr，
   参数SERVER说明该程序将在此界面上提供服务 */
void body (int argc, char *argv, char *envp)
{
ose_InterfaceRef Window_server_ifr;
! CONSTRUCT Window_server_ifr OF Window WITH
  (2);
/* 创建Window界面的一个实例，其界面参考为window_
   server_ifr，同一时刻最多能处理的请求数是2 */
! EXPORT {Window_server_ifr}TraderRef WINTH
    (Window,other_papameters)
    /* 把服务器的界面实例Window_server_ifr输出到
       交易器 */
    void Window_Output Window()
    /* 完成Window界面中OutputWindow()操作的具
       体函数 */
    {
    ⋮/*C语言代码*/
    }
```

图4 server.cpl文件的结构

```
#include "ose.h"
! INCLUDE Window
! INCLUDE Trader
! DECLEAR {Window_client_ifr};Window CLIENT
/* 参数CLIENT表示该程序将请求这个界面提供的服务
*/
void body (int argc, char *argv, char *envp)
{
    ose_InterfaceRef Window_client_ifr;
! IMPORT Window_client_ifr FROM TraderRef WITH
    (Window, other_parameters)
/* 从交易器中输入Window界面的界面参考，并把它赋予
   Window_client_ifr */
! DO Window_client_ifr.OutputWindow WITH()
    RETURN()
/* 调用界面中的OutputWindow操作 */
}
```

图5 client.cpl文件的结构

结束语 以上我们详细论述了互操作中间件OSEware的设计思想和实现方案。OSEware已经在三类操作系统(Unix,DOS,Window)上正常运行，所使用的协议是TCP/IP。

我们下一步的工作主要包括三方面：1)为系统增加更多的实用工具，以完成系统的实用化和工程化；2)OSEware是一个大型软件系统，互操作技术

(下转第77页)