对象互操作的层次模型*)

李 贵 冯季昉 韩子扬 郑新录 王国仁

(沈阳建筑大学信息学院 沈阳 110168) (东北大学信息学院 沈阳 110006)

摘 要 在分析传统面向过程互操作的基础上,针对大规模环境下对象互操作问题,首先提出了一种对象互操作的分层结构模型,将对象互操作分为说明层、接口层和通信层互操作。针对每层互操作,提出了各自的机制与方法,为对象互操作提供了不同层次的透明性支持。

关键词 对象互操作,类型匹配,操作捆绑,对象映射

Level Model of Object Interoperation

LI Gui FENG Ji-fang HAN Zi-yang ZHENG Xin-lu WANG Guo-ren

(Shenyang Architecture University, Shenyan 110168, China) (Northeastern University, Shenyang 110006, China)

Abstract The procedure-oriented interoperability methods were discussed at first, and a level model was put forward for object interoperation in large-scale distributed computing environments. This model is divided into three interoperation levels, specification-level interoperability, interface-level interoperation, and communication-level interoperation, each level provides different levels interoperation transparency, and the interoperation mechanisms and methods were discussed in each level.

Keywords Object interoperation, Type matching, Operation binding, Object mapping

1 引言

实现透明对象互操作是分布对象管理系统的目标之一。传统的互操作技术研究基本上是基于远程过程调用 RPC (Remote Procedure Call)机制。我们称基于 RPC 机制的互操作为面向过程的互操作 POI(Procedure Oriented Interoperability)。在面向过程互操作的环境中,服务过程所提供的功能和接口与客户所请求的功能和接口是完全匹配的,互操作的关键问题是如何实现客户方的调用过程和服务方的被调用过程之间的参数传送和匹配。

利用 POI 方法实现的互操作对于大规模面向对象互操作环境来说还存在如下几个方面的不足:

· 接口退化问题。POI 所支持的互操作是在过程调用层次上实现的。然而,在大规模面向对象互操作环境中,通过一种接口所提供的服务通常是由一组相互关联的过程实现的。从用户的角度来说,重要的是它所提供的整体功能,而非其接口过程。如果把互操作接口定位在过程调用层次上,相互关联的接口过程之间就会丢失它们之间的关系语义,或者由用户来处理过程之间的关联语义。同时,提供互操作服务的接口不能作为一个独立的实体,而是被看作相互独立的过程,这将产生接口失配问题。

· 接口固化问题。在类似于 SLI 和 CORBA 的途径中,虽然不会产生接口失配问题,即用户必须遵照预先定义好的接口方式,然而这种预先定义接口的方式使得访问不同服务器上通过不同的接口提供相同服务功能的对象更加困难。这种方式对大规模对象互操作来说是一个很大的限制,因为在一个开放分布环境中,很难能预先定义所有对象服务的接口。

· 参数类型问题。基于 RPC 互操作途径的共同点是需要解决过程调用参数从客户环境向服务环境的移植问题,结果是只有可移植的类型才能作为过程参数。这些可移植的类型包括:基本数据类型(integer, string, real 等)和它们的聚集(array, structure 等)。而那些不可移植的抽象类型(abstract type),如数据库类型等就不能作为过程参数来进行传递。这种限制在面向对象的大规模互操作环境中显然是不合理的。

基于上述分析,本文针对大规模环境下对象互操作问题,首先提出了一种对象互操作的分层结构模型,将对象互操作分为说明层互操作、接口层互操作和通信层互操作,提供了不同层次的透明性支持,构成了一个基本的对象互操作体系;然后针对每层互操作,研究了相应的机制与方法。

2 对象互操作层次模型

依据互操作提供的透明性不同,我们将对象互操作分为:说明层互操作、接口层互操作和通信层互操作,如图 1 所示。

· 说明层互操作:包括两个方面的内容,首先要使用独立于任何编程语言的接口定义语言 IDL 来描述对象的接口和数据类型。IDL 是一个跨越语言、工具、操作系统和网络的说明性语言。然后在具体实现时通过 IDL 编译器把 IDL 描述的对象接口和数据类型映射到具体编程语言的类型模型上,并生成客户接口和服务接口的对应成分。语言映射定义了怎样利用客户接口和服务接口编写客户对象程序和服务对象程

*)本课题曾得到国家"863 计划"研究项目"基于 CORBA 的多源数据互操作与分布处理系统"资助。**李 贵** 博士,教授,主要研究方向为软件工程、分布对象技术和信息集成技术;**冯季昉** 硕士研究生,主要研究方向为分布对象技术和信息集成技术;**韩子扬** 硕士研究生,助教,主要研究方向为软件工程、分布对象技术和信息集成技术;**郑新录** 讲师,主要研究方向为软件工程和信息集成技术;**王国仁** 博士,教授,博士导师,主要研究方向为分布对象技术、XML 数据管理和生物信息技术。

序。通过 IDL 定义对象的标准接口的最大的优点是，在分布应用系统中的客户对象和服务对象可以在不同的环境下，使用不同的语言来实现。说明层互操作极大地提高了异构环境下的软件复用性。

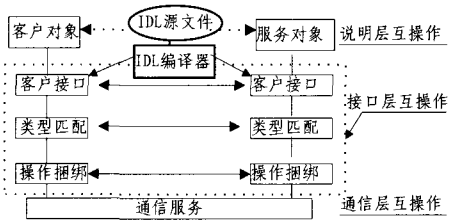

图 1　对象互操作的层次结构模型

• 接口层互操作：重点解决客户接口和服务接口之间的类型匹配和操作捆绑问题。在大规模互操作环境中，客户对象和服务对象通常是处于不同的操作环境（不同的操作系统、不同的编程语言和不同的工具等）中，客户接口和服务接口之间一般来说是不直接匹配的。其原因一方面是客户环境和服务环境的类型模型是不同的，另一方面是不同的对象可能通过不同的服务接口提供相同的服务功能。所以说，解决客户接口和服务接口之间的类型匹配和操作捆绑问题是实现大规模对象互操作的基本前提。

• 通信层互操作：主要关心的是客户和服务器之间的通信机制。通信机制的标准化对实现任意客户到任意服务器的连接是非常必要的。通信机制不仅包括对象通信的协议，同时包括对象接口的标识转换问题，以及服务对象和服务器的定位问题等。

对于一个开放的分布对象处理环境来说，只有将这 3 种层次的互操作有机地结合起来，才能提供一个完整的互操作体系。

3　说明层互操作

3.1　说明层互操作的实现结构

说明层互操作的目标是为互操作程序的开发者提供一种有效的共享对象类型的抽象表达能力，这种抽象表达是建立在一个公共对象模型和接口定义语言的基础上，使得开发者可以依据公共对象模型和接口定义语言自由定义共享对象的类型。说明层互操作的焦点是共享对象类型的定义和接口语言映射，其实现结构如图 2 所示。主要功能包括：

• 提供一个统一的类型模型 UTM(Unified Type Model)和接口定义语言 IDL

UTM 是描述共享对象类型的理论基础，是一个统一的对象模型，能够充分描述共享对象类型的特性。IDL 是基于 UTM 的接口定义语言，即 UTM 的形式化描述方法，用来描述对象的接口和数据类型。IDL 是一个跨越语言、工具、操作系统和网络的说明性语言。

• 语言映射

对于基于 UTM 的 IDL 描述和一个特定的编程语言，要通过一种方法和机制把 UTM 中的类型定义与编程语言中的类型定义对应起来，这种 UTM 和不同编程语言之间的映射称为语言映射。这种语言映射在具体实现中是通过 IDL 编译器把 IDL 描述的对象接口和数据类型映射到具体编程语言的类型模型上，并生成客户接口和服务接口的对应成分。语言映射定义了怎样利用客户接口和服务接口编写客户对象程序和服务对象程序。通过 IDL 定义对象标准接口的最大的优点是分布应用系统中的客户对象和服务对象可以在不同的环境下使用不同的语言来实现。

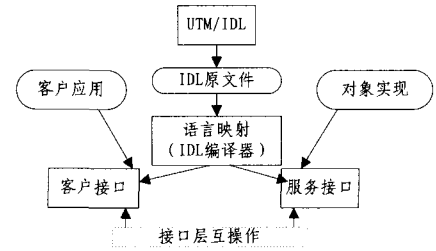

图 2　说明层互操作的实现结构

3.2　扩展对象模型

当前，在不同的领域（包括分布应用框架、数据库、用户接口、编程语言和类库等）存在不同的对象模型，OO 模型的多样性在很大程度上影响了软件的可移植性和互操作性。为支持分布对象说明层互操作，需要有一个公共的集成对象模型。为此，我们基于 CORBA 标准，提出了一种可扩展的对象模型，以此作为支持分布对象说明层互操作的统一类型模型 UTM。

扩展对象模型由 3 部分构成：核心对象模型（Core Object Model）、组件（Component）和领域套件（Profile）。扩展对象模型的基本结构如图 3 所示。

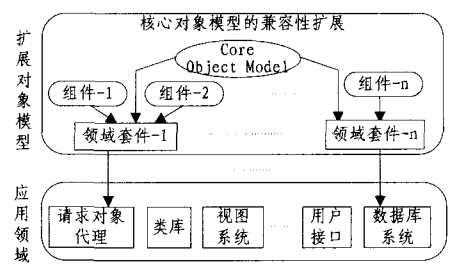

图 3　扩展对象模型的基本结构

核心对象模型：定义一组所有面向对象领域（包括分布应用框架，数据库、用户接口、编程语言和类库等）都支持的公共的特性或概念，包括对象、对象类型、类、操作、子类化和集成等。核心对象模型是实现对象移植和互操作的基础。

组件：由于面向对象应用领域的广泛性，仅仅利用核心对象模型来完整刻画一个领域是不够的。为此利用组件技术来满足不同领域的特殊性要求，组件是面向特定领域的对核心对象模型的一种兼容性扩展。

领域套件：对于给定的技术或应用领域将相关的组件结合起来，与核心对象模型一起构成描述特定领域的一个外延，满足该领域不同应用的需求。领域套件可以分为两类：一类是基于技术的，比如数据库领域套件和编程语言领域套件等；另一类是基于应用的，比如面向 CAD 应用领域的套件，面向 OA 或 MIS 应用领域的套件等。

4　接口层互操作

接口层互操作是说明层互操作的基础，所要解决的问题是客户环境和服务环境的类型不一致性问题，以及客户接口和服务接口之间的接口匹配问题。为此本文提出了类型匹配、操作捆绑机制。类型匹配用于定义不同环境中对象类型之间的关系；操作捆绑定义了客户接口和服务接口之间的对应关系。

4.1　类型匹配

4.1.1　类型关系

类型关系表示本地对象类型和远程对象类型之间的对应

关系。我们依据本地类型和远程类型之间的转换方式将类型关系划分为三种：类型等价、类型转换和类型匹配。

类型等价关系：如果本地类型 $X=\{X_C,X_S,X_O\}$ 的结构、操作和语义与远程节点类型 $Y=\{Y_C,Y_S,Y_O\}$ 的结构、操作和语义是相同的，即在移植一个对象时只移植其数据部分即可，那么 X 和 Y 是等价的，表示为：

$X \Leftrightarrow Y$

在等价类型之间的对象交互只涉及数据的交互，不涉及对象操作和语义的转换。

类型转换关系：X 为本地类型，Y 为远程节点类型，$X+>Y$：T_Function 表示为本地类型 X 通过转换函数 T_Function 转换为远程节点类型 Y。

本地类型到远程类型转换是依据用户定义的转换函数实现的。用户可以根据对象的语义和应用的具体需求定义不同的转换函数。

类型匹配关系：如果我们定义了一个本地类型到远程类型之间的捆绑过程（即从客户接口到服务接口的捆绑与转换），则称该本地类型和远程类型具有类型匹配关系。类型匹配关系分为单类型匹配和多类型匹配。

单类型匹配表示为 $X->X'$；本地类型 X 匹配到远程类型 X'。

多类型匹配表示为 $X->\langle X',X''\rangle$；本地类型 X 匹配到远程类型 X' 和 X''。

类型关系是针对节点而定义的，而且是一种单向不对称的关系。

4.1.2 类型关系描述

为了能够比较清楚地描述不同节点之间的类型匹配关系，我们采用一种类型匹配描述语言 TMSL（Type Matching Specification Language）作为类型匹配的一种形式化描述方法。

在下面的讨论中，我们将以两种面向对象的语言作为互操作环境的例子。二者的主要差别在于它们所采用的对象模型是不一致的。在此例子中，L_node 作为本地节点，R_node 作为远程节点。

类型关系描述的语法结构为：

ID of remote_node :: 〈Type_relation〉

其中，ID of remote_node 为远程节点标识；〈Type_relation〉表示类型关系，可以是类型等价、类型转换和类型匹配。

下面给出一个具体的对象互操作的例子。本地节点 L_node 上的类型 windowServer 中定义了访问对象服务的接口（客户接口），如图 4 所示。

```
type windowsServer: abstract{
    newWindow: (integer : topLeftX,integer: topLeftY,
        integer: botRightX,integer: botRightY)(integer: windowsId;
    newSquareWin: (integer: topLeftX, integer: topLeftY, integer: side)
                  →integer: windowsId;
    refreshDisplsy: (displsy) →bollean;
    readCoordinates: (mouse, keyboard, touchScreen, integer: scaleFactor)
                  →point;
    windowSelectd: (mouse,keyboard,touchScreen,) →integer;
```

图 4 L_node 上的类型 windowServer

在远程节点 R_node 上的类型 WINDOW_CONTROL 中提供了 windowServer 的实现，定义了服务方接口，如图 5 所示。

```
TYPE WINDOW_CONTROL =
  OBJECT
    METHOD create_win (IN botRigutX: INT, IN botRightY: INT,
          IN topLeftX: INT, IN topLeftY: INT, IN color: INT): INT
    METHOD redisplay_all ( IN display: DISPLAY): INT
    METHOD get_Position(IN inDevices: IO_DEVICES,
      IN scaling : INT): POSITION
    METHOD select_Window(IN position: POSITION): INT
  BODY
  ......
  END OBJECT
```

图 5 节点 R_node 上的类型 WINDOW_CONTROL

针对上面的例子，在本地节点 L_node 上存在如下类型关系：

R_node :: integer ⇔ INT;

R_node :: boolen ⇔ BOOL;

R_node :: string +> ARRAY OF CHAR:string to array of char

.....

R_node :: windowServer→WINDOW_CONTROL{〈operation binding〉};

.....

4.2 操作捆绑

类型匹配解决了类型之间在抽象服务功能上的一种对应关系，并没有解决类型接口或方法上的对应关系。一般情况下，接口不同体现在两个层次上：一是支持的操作类型不同，二是操作参数不同。接口异构性的解决要依赖于类型匹配定义过程中的参数和接口分析的结果。操作捆绑是将客户接口中的每一个操作对应到服务接口中的对应的操作上，这种对应关系可能是一对一的，也可能是一对多的。操作捆绑过程包括参数捆绑阶段和接口匹配阶段。操作捆绑就是解决互操作对象之间在类型接口或方法上的匹配问题。

4.2.1 参数捆绑

假设服务接口所提供的功能与客户接口所要求的功能是对应的，此时需要解决的问题是用户接口和服务接口之间的参数匹配问题。本地客户接口的提供参数和远程对象服务接口所需参数的不同可分为：参数的顺序不同；参数的表示不同；参数的语义不同；参数个数不同。

依据上面的分析，提出如下三种参数捆绑方式：移植参数、映射参数和多类型映射参数。

(1)移植参数

移植参数方式用于客户接口和服务接口在功能上是匹配的，并且它们的对应参数类型之间有等价或转换关系，可通过移植参数方式来实现对远程对象服务的访问。在类型 WINDOW_CONTROL 上不存在与操作 newSquareWin 对应的操作接口，但是利用给定的参数值调用方法 create_win 可以实现 newSquareWin 的功能。捆绑格式如下：

newSquareWin:create_win(bottom RX(1,3),bottomRY(2,3),1)

其中，bottomRX 和 bottom RY 是用户定义的适配函数。通过适配函数的转换，实现客户接口的提供参数与服务接口的需求参数之间的匹配。

(2)映射参数

当用户接口的参数不能移植到远程节点时,即在远程节点没有对应的等价或转换类型,那么这些参数应该本地访问。这将通过类型匹配的方式把远程对象映射到本地参数来实现。在上面的例子中,操作 refreshDisplay 和 redisplay_all 就属于此种情况,为此需要在节点 R_node 上建立如下的类型匹配关系:

Hybrid_node :: DISPLAY→display {……}

这样,refreshDisplay 和 redisplay_all 之间捆绑可表达如下:

refreshDisplay:redisplay_all (1 : display←DISPLAY) ^ int_to_bool(RET)

其中,方法 redisplay_all 中第一个参数为一个对象,该对象将按照节点 R_node 上指定的类型匹配映射到 refreshDisplay 中的第一个参数。另外,方法 redisplay_all 的返回类型为整数,需经适配函数 int_to_bool 转换为 Boolean 类型。

(3)多类型映射参数

在有些情况下,一种参数的类型可能由远程节点上的多种参数类型的组合来表达。在上面的例子中,R_node 节点的 IO-DEVICES 就属于此种情况,它对应节点 L_node 上的类型 Mouse,Keyboard 和 touchScreen 的组合。类型关系定义如下:

Hybrid_node :: IO_DEVICES→〈keyboard@,mouse@,touchScreen@〉{…}

在上面的例子中,中间代理对象 IO_DEVICES 将被连接到节点 L_node 上的 Keyboard,Mouse 和 touchScreen 对象上。

4.2.2 接口适配

在定义类型之间的操作捆绑时,有一种情况是本地局部请求的功能由多个远程操作功能的"组合"来实现。为了实现请求接口与服务接口之间的匹配,所要做的工作包括从简单的操作组合到复杂的编程。TMSL 允许在操作捆绑的说明中定义简单的操作组合。比如节点 L_node 上的操作 windowSelected 可以通过节点 R_node 上的操作 get_Position 和 select_window 的嵌套组合来实现。操作捆绑定义如下:

windowSelected: select_Window

(WINDOW_CONTROL. get_Position(〈2,1,3〉:〈keyboard,mouse,touchScreen〉

←IO_DEVICES,4))-RET

在上面的定义中,首先在远程节点上调用方法 get-Position,其结果不返回调用节点,而是作为方法 select_Window 的第一个参数。

5 对象映射

类型匹配解决了接口层互操作的类型失配问题。在通信层互操作中我们采用一种对象映射机制,对象映射提供了对象互操作实施过程中的动态支持,实现对远程对象服务的访问。对象映射分静态和动态两部分:对象映射的静态部分依据类型匹配说明建立代理对象和代理类,代理对象是代理类的一个实例,代理对象负责客户对象和服务对象之间连接的建立、请求和结果传送等;对象映射的动态部分负责互操作过程中所用对象的建立(实例化)和管理。对象映射的实现模型如图 6 所示。

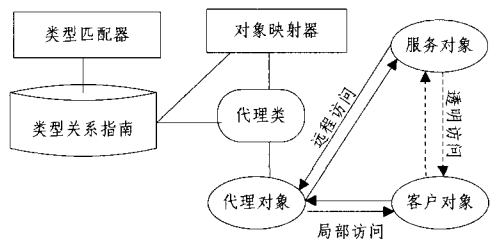

图 6 对象映射的实现模型

结束语 对象互操作是实现开放分布处理的一种有效途径。本文主要针对对象互操作的分层模型进行了研究,提出了三层结构模型,并对每层互操作提出了具体的解决方法。针对该模型的实现机制与方法方面,本文课题组基于原型系统正做进一步的研究,并在其他文献里做进一步介绍。

参 考 文 献

[1] Vallecillo A, Hernández J, Troya J M, et al. Object Interoperability// Proceedings of the ECOPP'99 Workshop on Object Interoperability (WOI'99). Universidad de Málaga, 1999

[2] Hernández J, Vallecillo A, Troya J M, et al. New Issues in Object Interoperability// Proceedings of the ECOOP'00 Workshop on Object Interoperability (WOI'00). Universidad de Extremadura, 2000

[3] Vallecillo A, Vasconcelos V, Ravara A. Typing the behavior of objects and components using session types// Proc. of FOCLASA 2002. ENTCS, 2002, 68 (3)

[4] Durán F, Vallecillo A. Formalizing ODP Enterprise Specifications in Maude. Computer Standard & Interfaces, 2003, 25(2): 83-102

[5] Durán F, Roldán M, Vallecillo A. Using Maude to Write and Execute ODP Information Viewpoint Specifications. Computer Standard & Interfaces, 2005, 27(6): 597-620

(上接第 130 页)

体映射评价标准的缺陷,同时保证了广泛的适用性。另外,根据分布式描述逻辑,我们借鉴了概念相似度的计算方法,提出了一组针对实体的纯语义查全率和查准率函数以及权重函数,实现了这个框架。由于篇幅所限,研究成果将另文发表。

参 考 文 献

[1] Berners-Lee T, Hendler J, Lassila O. The Semantic Web. Scientific American, 2001, 284(5): 34-43

[2] Ehrig M, Euzenat J. Relaxed Precision and Recall for Ontology Matching// Ashpole B, et al, eds. Proc. K-Cap Workshop on Integrating Ontology. New York: ACM, 2005: 25-32

[3] 袁洋,李善平. 基于语义 Web 的本体映射方法综述. 计算机科学,2004,31(5):5-8

[4] van Rijsbergen C. Information Retrieval. London: Butterworths, 1979

[5] Euzenat J. Semantic Precision and Recall for Ontology Alignment Evaluation// Proc. IJCAI'07. Berlin: Springer, 2007: 348-353

[6] 石莲,孙吉贵. 描述逻辑综述. 计算机科学,2006,33(1):194-197

[7] Borgida A, Serafini L. Distributed Description Logics: Assimilating Information from Peer Sources // LNCS 2800. Journal of Data Semantics, Berlin: Springer, 2003: 153-184

[8] Haase P, et al. A Framework for Handling Inconsistency in Changing Ontologies// Gil Y, et al. eds. Proceedings of the Fourth International Semantic Web Conference (ISWC 2005). LNCS 3729. Berlin: Springer, 2005: 353-367