层次协同系统中的对象映射模型*)

Object Mapping for Layered Cooperative Systems

李 冀 樊健生 李成错 茅 兵 陈贵海 谢 立
(南京大学软件新技术国家重点实验室 南京210093)

Abstract This paper introduces a new model for inter-layer message communication called object mapping model that achieves separation between message preparation and destination selection. In this model, as opposed to traditional message delivery model, sender obejcts of messages need not indicate the destination objects of the messages. The messages can be intercepted by special objects called mapper objects. When mapper objects intercept messages, they can decide to which objects the messages should be sent. Using this model, layered cooperative systems can be realized by using objects to construct each layer and using mapper objects to realize object mapping relationship. And object mapping model has many advantages such as modularity, transparency, implicit invocation and dynamic mapping.

Keywords CSCW, Object-oriented cooperative system, Object mapping relationship, Mapper object

1. 引言

自从80年代中期CSCW的概念提出以来,在学术界和工业界深受重视,并成为一个迅速发展的新兴研究领域。随着网络技术的快速发展和日常通信和协作的日益频繁,协同工作已经成为不可逆转的潮流,具有广泛的应用前景。

但是CSCW中诸如多用户、分布的界面、协同感知、协同等特征,使得协同系统的设计和开发非常的困难[1~4]。和单用户系统一样,协同系统必须执行一些交互的任务,如显示数据、读入输入和报告错误。协同系统还要执行一些协同任务,如从多个用户中选择一个作为输入,还要选择一个输出到多个用户,通知用户其它用户的动作,协调用户交互等等。这些要求意味着CSCW系统不仅要处理人机交互,而且要处理人人交互。当前协同系统开发的难题主要在于缺少协同系统设计和协同感知的通用模型。为了解决这些问题,我们提出了分层的面向对象的协同系统模型[5]。此模型提供了一个分层的体系结构,用对象构建层;用对象链接机制实现结构化的协同关系;它将CSCW系统分割为多层,每一层关注独立的问题,这将应用的语义行为同界面的交互属性分离开来;它还提供了保持应用的语义特征和协同分离的设施。使用此模型可以增强CSCW系统的灵活性,提高协同效率,方便开发。

对象是协同计算模型的基本构筑单位。在相关对象之间传递的消息会触发成员函数的调用,从而实现对象的理解和表达功能。消息的传递可以分为两步,即消息准备和目的地选择。传统的对象模型中,这两步都由消息发送对象完成。本文提出了一个新的层间消息传递的模型,即对象映射模型。该模型可以实现消息准备和目的地选择的分离。在我们的协同模型中,这种分离是实现层次协同必不可少的,后面将详细阐述其必要性和优点。与传统对象模型不同,消息发送者不需要指明接收对象,消息将被一个特殊的映射对象所截获,并由映射对象决定其目的地。为此,我们引入了一个新的类间关系:映射关系。该模型实现了基于对象的协同计算模型中的对象映射关系。

2. 相关工作

对专门领域软件体系结构设计方法的研究巩固了基于构件开发和设计软件的方法。对此的研究包括提供图形用户界面的交互软件的层次体系结构,相关的例子有文[6~9]。对体系结构风格的研究,一个重要原则是概念的分离。例如在图形用户界面软件的层次体系结构方面,现有的方案对各层分离出不同的抽象级以保证易重用性,易修改性和易移植性。如果层间的通信由独立的连接子(connector)来管理,层次型的交互系统将很容易适应协同需求。连接子是软件设计方法学和体系结构语言中的一种抽象,是负责计算构件之间通信的软件构件。因为连接子封装了所有的分布和

*)本文得到国家863高技术项目(863-306-ZT02-03-01)的资助。李 冀 硕士研究生,研究领域为分布式处理和并行计算。

网络连接管理等功能，其引入增强了软件体系结构的易修改性。

Chiron-2[9]中提出了一种新的体系结构风格，支持大粒度的重用和灵活的系统构造，支持分布和并发应用程序设计。异步通知和请求消息是其构件间通信的基础，连结子的主要职责是路由和消息的广播，用来在相邻层间建立I/O通道，并将构件与Chiron-2体系结构绑定。连接子可以和任意数目的构件和连接子相连。

CSDL[8]中提出了一个采用层次体系结构设计支持灵活的共享和协同的多用户交互软件系统的方法。CSDL基于的原则是将管理应用功能的系统构件和管理协同控制的构件分离[10~12]，CSDL中用协同的连接子取代通常的点对点的连结子，这些连结子在相邻层的多个实例间实现输入复用和输出过滤，提供了在协同透明的层次系统中加入协同支持的基础，同时保持了应用功能和协同功能的独立性。

3. 分层的面向对象协同模型

当前的大多数CSCW系统用以下三种系统结构模型之一实现协同：集中式模型、复制式模型和混合式模型[5,13]。这三种模型在信息共享的方式上都存在共同的缺点：通过应用、共享机制实现协同。这意味着一个用户的输入事件在被其他用户感知之前，必须经过共享应用程序的处理，然后把处理结果发送给各个用户显示，即多用户的协同只有一个信息出入口。这就不可避免地导致了协同效率的低效。由于协同功能是由应用提供和控制的，使得它与应用紧密相关，只能满足特定应用的协同需求，导致应用的协同特征被系统结构束缚。这个局限使得基于这些体系结构的协同系统很难开发、扩充和移植。为了解决这些问题，我们已经在文[5]中提出了一个面向对象的层次系统模型，它的每一层由对象构建。

3.1 模型的层次结构

在层次的面向对象的系统模型中我们把协同系统分为五层（自下而上）：设备层、概念层、目的层、协同层和计算层。前三层主要处理用户的交互工作，后两层主要用于实现系统的协同和计算功能，这样的模型既使系统易于移植和扩充，也方便了实现。

3.2 用对象构建协同系统

为了构建一个结构化的协同系统，我们用对象构建各层。我们的模型中，协同借助对象链接实现，而不象其它系统那样借助应用之间的共享。对象链接是这样一种机制：当某个对象收到一个事件时，将触发在另一用户计算机上的对应对象，这个对象称为被链接的对象。

3.3 对象间的协同

按照模块化原则，我们的模型将协同策略与数据流控制相分离。如图1所示，CSCW系统控制协同的部分定义了存取权限和协同感知权限，存取权限用于处理用户接口与共享工作区的数据交换，它被描述在存取控制模块AC(Access Control module)中，协同感知权限定义用户动作被多个参与者感知的能力，并被描述在协同感知控制模块CAM(Cooperative Awareness Management module)中。两者都由协同规则控制。

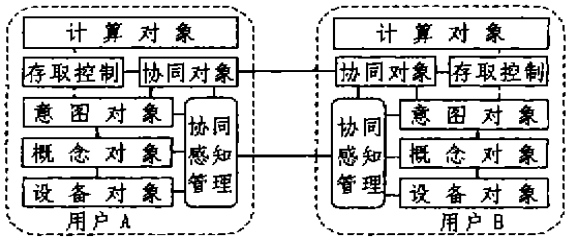

图1 基于对象的层次协同模型

系统支持参与协同工作的用户感知多层次的协同。我们的模型通过链接同层的相关对象来提供这一支持，从而导致层对层的协同。对应于系统的层次结构，存在四个层次的协同，分别为由协同感知模块(CAM)控制的设备层、概念层、表意层和由存取控制模块(AC)管理的应用层协同。应用层协同基本上类似于传统的应用共享的协同机制，是CSCW系统的基本协同方式，其余三个层次的协同基于用户的交互动作，能提供细粒度的协同。

系统的协同通过同层相关对象的协同实现。有关用户接口的三个协同层次简述如下：

设备对象的协同 是最细粒度的协同，通过链接不同机器上的设备对象（例如鼠标、键盘、屏幕等）实现。一个设备对象一旦被一个输入事件激活，将触发其它所有与其相链接的设备对象，从而使所有相关的用户感知相同的结果。但要求同构的物理设备，且只能支持严格的WYSIWIS协同模式。

概念对象的协同 概念对象作为输出设备上视域对象的逻辑表示，其链接导致概念层的协同。链接的概念对象可有不同的特性（如外观、位置等等），能方便地实现WYSIWIS协同模式。

表意对象的协同 与前两种协同相比，这是一种间接的协同方法，它通常用于实现不能被下两层直接表达的协同，也可用于实现异构应用之间的协同。

4. 对象映射模型

4.1 对象映射关系

对象由数据、方法以及与其他对象通信的接口所

组成。在基于对象的协同框架中，一个对象不仅要与上一层和下一层的相关对象通信，以传送用户的动作要求和返回用户动作的结果，而且要与其他用户的相同层次的相关对象通信，以实现多用户的协同。

如图2所示，一个对象包含三种通信接口，即上端接口、下端接口和邻端接口，分别与上一层、下一层以及其他用户系统的相同层次的相关对象通信，每一种接口都有输入和输出端口。下面列出了对象的基本接口部件：(1)DIS 端口(Input Socket for Deifying)：接受从下一层相关对象传送来的消息；(2)ROS 端口(Output Socket for Reifying)：传送消息到下一层的相关对象；(3)RIS 端口(Input Socket for Reifying)：接收从上一层相关对象传送来的消息；(4)DOS 端口(Output Socket for Deifying)：传送消息到上一层的相关对象；(5)CIS 端口(Input Socket for Collaboration)：接收从其他用户的相同层的相关对象传送来的消息；(6)COS 端口(Output Socket for Collaboration)：传送消息到其他用户的相同层的相关对象，按照对象接口的输出方向，所包含的功能分为以下三类：

理解功能 一个来源于 DIS 和 CIS 端口的事件需经理解后才能通过 DOS 传送给相邻的上一层。

表达功能 一个事件不论从何而来，最终往往都需要反馈给相关用户。一个对象对一输入事件处理后，通常组织一个表达事件通过 ROS 传送给相邻的下一层的相关对象。

协同功能 从 RIS 和 DIS 传来的输入事件，可能需要传送给其他用户的同一层次的相关对象，从而实现多用户在这一层次的协同感知。

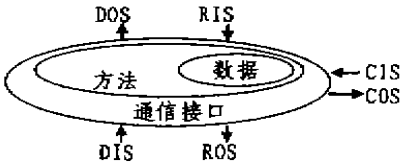

图2 对象的结构

在同一用户的相邻层次的相关对象之间的消息传递的目的是实现对象的理解和表达功能，相关对象之间的关系被称为对象映射。在我们的模型中，对象映射关系由对象映射知识管理，对象映射知识可以修改以适应某一层的变化，而不改变系统的其余部分。

与此同时，在我们的模型中，协同是以对象链接机制实现的，而不是象其它系统那样 依赖于应用共享。对象链接是这样的一种机制，一个对象接收一个事件的同时，它触发在其他用户机器上的与其有对象链接关系的对应对象，这些对应对象也称为链接对象。链接对象通常是相同的类结构的不同实例，它们拥有相同的消息集。

4.2 映射对象

图3是传统消息传递模型[3]。Sender 对象调用 Receiver.handle()，消息的目的对象由发送对象选择，消息发送的结果是相应方法在目的对象的激活。

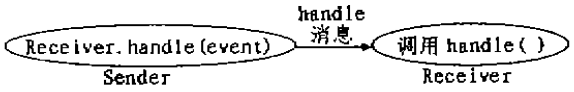

图3 传统消息传递模型

由于消息是直接传递的，目的地选择必须在发送对象中指出，不可能在不牺牲透明性的情况下抽象出目的对象选择部分。例如，如果在图3中插入一个中间对象以选择 Sender 对象发出的消息的目的地，Sender 必须被修改为调用该中间对象，当中间对象需要被替换或删除时，Sender 的代码也需要改写。

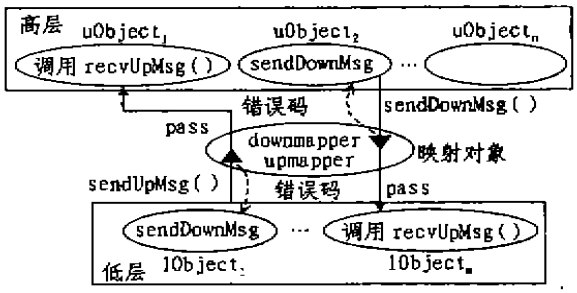

图4 对象映射模型

图4是对象映射模型。映射对象作为其客户对象的消息目的地选择器位于两个相邻层间。$lObject_{1..m}$ 和 $uObject_{1..n}$ 是映射对象的客户，相对较低层的 $lObject_{1..m}$ 称为下层客户，高层的 $uObject_{1..n}$ 称为上层客户。$lObject_{1..m}$ 和 $uObject_{1..n}$ 间的两个方向的消息传递都经由映射对象，映射对象映射的是这些对象的成员函数。当这些成员函数被调用时，映射对象自动选取一个相邻层对象作为消息的目的地并触发对这个目的对象成员函数的调用。例如，当一个对象($lObject_1$ 自身或其他对象)调用 $lObject_1$.sendUpMsg(msg)，映射对象选择 $uObject_1$ 作为目的对象并且调用 $uObject_1$.recvUpMsg()。另一个例子是当调用 $uObject_2$.sendDownMsg(msg)时，映射对象选择 $lObject_m$ 作为目的对象并调用 $lObject_m$.recvDownMsg()。映射对象具有以下功能：

(1)截获上行/下行的消息：上行/下行的消息是相邻两层间传递的消息，映射功能分别称为上行映射和下行映射。

(2)对消息的操作：映射对象可以改变消息的参数

以及执行一段代码。

(3)错误处理:映射对象检查消息是否有错,并返回错误代码给消息的发送对象。

(4)目的对象选择:映射对象在截获消息后选择消息将要发送到的目的对象。对上行的消息,映射对象从紧邻上层中选择目的对象;对下行消息,在紧邻下层中选择。

(5)消息传递:映射对象在选定目的对象后就将消息传递到目的对象,触发目的对象的方法调用。

4.3 映射对象的性质

(1)模块性:映射对象与其位于两个相邻层的客户对象分离,既不破坏客户对象的封装性,客户对象也不会破坏映射对象的封装性。

(2)透明性:由于消息发送对象不知道映射对象的存在,当映射对象加入、替换和删除时不需改变发送对象的代码。

(3)隐式调用:发送对象不知道所发送消息的目的对象,但通过映射对象能够隐式地调用目的对象的方法。

(4)映射选择:映射对象可以选择性地截获消息。客户对象的方法可以是未映射的,也可以是被映射的。这使得映射对象可以为多个消息实现相互独立的映射代码。

(5)动态映射:客户对象可以动态改变其映射对象。映射对象的动态绑定发生在两个级别上:首先映射对象可以被删除和替换,其次映射对象的映射方法可以在运行时刻改变。

4.4 必要性与优点

消息准备阶段和目的对象选择阶段的分离对实现层次协同是非常必要而且有益的。

首先,层次协同系统中每层都由许多对象构成,如设备层有鼠标对象、键盘对象、屏幕对象等,因此消息发送给某一层时必须指明目的对象。如果每层只用一个对象去构造,它必须处理发向此层的所有消息,这破坏了面向对象系统中对象的封装性。

其次,为实现对象的理解和表达功能,消息在相邻层间传递。为决定消息的目的对象,在目的对象选择阶段预先对到达消息的理解和表达是必要的。而消息的发送对象仅拥有对本层的知识,因此无法确定发向相邻层的消息应由何对象来接收。

同时,这种分离是非常有益的。传统消息传递模型中,目的地选择阶段与消息准备阶段是紧密结合的,这导致不可能在不破坏透明性的前提下抽象出目的对象选择阶段。对象映射模型使得用模块化的方式开发出目的对象选择策略成为可能,并且这种分离提供了透明地修改层对象和映射对象的能力。同时,协同系统可能要求它的目的对象选择策略可以被动态改变,对象映射模型的透明性和模块化使得这成为可能。

结论 我们提出了一个新的层间消息传递的模型:对象映射模型。该模型可以实现消息准备和目的地选择的分离。同时引入了一个新的类间关系:映射关系。该模型实现了基于对象的协同计算模型中的对象映射关系。对象映射模型使得用模块化的方式开发出目的对象选择策略成为可能,并且这种分离提供了透明地修改层对象和映射对象的能力。而且,协同系统可能要求它的目的对象选择策略可以被动态改变,对象映射模型的透明性和模块化使得这成为可能。

未来的软件构件设计中,对于协同系统应用的考虑将越来越重要,而且层次结构的协同系统更加重要。在研发此类系统时,我们应重视层间通信。虽然本文的重点放在层次协同系统,但对象映射模型具有更广泛的应用前景。

参考文献

1 Dewan P, Choudhary, R. A High-Level and Flexible Framework for Implementing Multiuser User Interfaces. ACM Trans. on Information Systems, 1992, 10(4): 345～380

2 Li D, Muntz R COCA Collaborative Objects Coordination Architecture. In: Proc. of ACM CSCW'98, Seattle, Washington, 1998. 14～18

3 Dourish P. Using Metalevel Techniques in a Flexible Toolkit for CSCW Applications. ACM Trans. on Computer-Human Interaction, 1998, 5(2): 109～155

4 Mao B, Xie L. An Object-Based Model for Prototyping User Interfaces of Cooperative System. ACM Software Engineering Notes, 1997, 22(2): 72～76

5 Mao B, Xie L. An Object-Based Computing Model for CSCW Systems. Science in China (Series E), 1998, 41(1): 22～30

6 DePaoli F, Sosio A. Requirements for a Layered Software Architecture Supporting Cooperative Multi-User Interaction. In: Proc. of the 18th Intl. Conf. on Software Engineering. IEEE, 1996

7 Hill R D. The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications. In: Proc. of ACM CHI'92. Monterey, California, 1992. 335～342

8 Taylor R N, Johnson G F. Separations of Concerns in the Chiron-1 User Interface Development and Management System. In: Proc. of ACM CHI'93. Amsterdam, 1993. 367～374

9 Taylor R N, Medvidovic N, Anderson K M. A Component and Message-Based Architectural Style for GUI Software. IEEE Trans. on Software Engineering, 1996, 22(6): 390～405

10 DePaoli F, Tisato F. Development of a Collaborative Application in CSDL. In: Proc. of the 13th Intl. Conf. on Distributed Computing Systems. IEEE, Pittsburgh, 1993. 210～217

11 DePaoli F, Tisato F. Cooperative Systems Configuration in CSDL In Proc. of the 14th Intl. Conf. on Distributed Computing Systems. IEEE, Poznan, 1994

12 DePaoli F, Tisato F. CSDL: A Language for Cooperative Systems Design. IEEE Trans on Software Engineering, 1994, 20(8): 606～615

13 Joshi R K, Vivekananda N, Ram D J. Message Filters for Object-Oriented Systems. Software-Practice and Experience, 1997, 27(6): 677～699