一种带有 delegation 的对象演算*)

杨 群 陈宏兵 许满武

(南京大学软件新技术国家重点实验室,南京大学计算机科学与技术系 南京 210093)

摘 要 随着研究与应用的深入,传统的基于类的面向对象语言对动态变化要求的支持不足越来越明显。而且,由于软件系统复杂性的不断提高,这个问题变得更加突出。近年来,人们一直在进行着各种研究和尝试,寻求解决办法。delegation 是一种在基于原型的面向对象语言中实现的对象动态继承,由于它支持对象行为在运行期动态改变,能提供对象动态扩展功能的能力,因此探讨如何在基于类的面向对象语言中引入 delegation 成分,以提供软件运行时刻的结构与行为变更能力,是十分有意义的。本文提出了一种命令式、带有 delegation 的 Φ 对象演算,以该演算系统刻划程序设计语言的基本特征;通过给出 Φ 演算的语法和操作语义;详细描述程序中各种操作的实现方法;着重说明在程序语言中引入 delegation 成分后,对象之间共享方法和对象扩展功能所具有的灵活性和简单性。从而说明在程序语言中引入 delegation,以支持软件动态变更是一种有效且可行的途径。

关键词 基于类的面向对象语言,基于原型的面向对象语言,delegate,delegation,Φ 对象演算

Φ Calculus: An Object Calculus with Delegation

YANG Qun CHEN Hong-Bing XU Man-Wu

(State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

Abstract Due to the development of more and more large and complex computing systems, software systems are always required to be changed at run-time. However, in the software systems that are developed using the traditional class-based object-oriented languages, this can not be done easily. Nevertheless, delegation, which is a language mechanism supported in prototype-based object-oriented language and enables objects to share methods dynamically, provide a promising method to solve the above problem. Actually, in recent years, delegation is an active research field among programming language researchers. In this paper, we will investigate this language mechanism by presenting Φ calculus. Φ calculus is an imperative object-based calculus. We use it to model essential features of languages, focusing on how to incorporate delegation into programming languages to support dynamic software changing. We give the operational semantics of Φ calculus. We also state how delegation is used in object extending and method sharing between objects. We conclude that delegation is an effective way to dynamic software changes.

Keywords Class-based object-oriented language, Prototype-based object-oriented language, Delegate, Delegation, Φ object calculus

1 引言

自上世纪 60 年代 Simula 语言出现至今,面向对象语言得到了极大的发展。但是随着研究与应用的深入,传统的基于类(class-based)的面向对象语言对动态变化要求的支持不足越来越明显。采用基于类的对象式语言编制的软件是由类组成的,程序的运行就是由类所实例化出的对象的生存活动,对象结构在类编制的时候就已经确定,运行期间无法改变;此外,由于类继承的层次结构早在类的设计时就已确定,对象也难以更改通过继承已获取的共享方法,所有这些都限制了软件功能的灵活性。随着软件系统复杂性增加和实际应用越来越多的动态变化的要求,这个不灵活性问题更加突出。虽然采用设计模式可以部分地解决软件的动态组合问题,然而却进一步导致复杂性提高、效率低下。

基于类的语言对软件动态变化支持不足的问题长期以来受到理论研究的重视。受应用需求的激励,近年来进行着多种设计尝试,例如,Microsoft 公司 2002 年发布的 C # 语言[1]、德国波恩大学 2000 年研制的 Lava 语言[2],都是很好的例子,它们在语言中加入了一种提供对象扩展能力的 delegation 成分[3],而使 delegation 成为这些语言设计的显著特征之一。然而,由于引入 delegation 成分后,使语言类型系统变得较为复杂,在基于类的面向对象语言以何种形式实现 delegation 机制,至今仍处于多种途径的探讨阶段。

正如 R. Miller 在其图林奖讲演所说的,语言设计的关键是形式化描述[4]。本文提出一种带有 delegation 的、命令式的 Φ 对象演算,形式化地描述了面向对象语言的本质特征[5]。通过对 Φ 演算系统的研究表明,引入 delegation 成分,赋予对象动态改变自身行为能力,允许软件运行时刻的结构与行为变更,可以有效地支持软件动态变更,可为程序语言的设计提供理论和实现的依据。

本文结构如下:第 2 部分剖析 delegation 语言机制;第 3 部分给出 Φ 对象演算,描述对象方法的更改、添加如何实现,着重说明引入 delegation 成分后,对象的功能扩展、对象之间方法共享所具有的灵活性和简单性;第 4 部分举例说明 delegation 的实际应用效果;第 5 部分,比较相关研究工作;最后是结束语。

*)本文受江苏省科技攻关项目(BE2003064)资助。杨 群 博士研究生,主要研究领域为软件方法学、新型程序设计、形式化方法等;陈宏兵 博士研究生,主要研究领域为软件方法学、形式化方法、自主计算等;许满武 教授,博士生导师,主要研究领域为软件方法学、新型程序设计、自主计算等。

2 delegation 语言机制

delegate 最早由 H. Lieberman 在 1986 年提出[3]，可描述如下：

1)一个对象，如果持有对另一个对象的引用，则称该对象为子对象(child)，称被引用的对象为父对象(parent)。

2)对于发送的一个消息，如果该消息的接收对象没有响应该消息的方法，则该消息将自动转递给消息接收者的父对象，这种消息的自动转递过程称为 delegate。

上述消息的转递过程中，消息需绑定一个始终指代消息初始接受对象的参数，该参数用 self 来表示。这种绑定使得最终对消息的响应就如同消息的初始接受对象做出的一样。

与 delegation 概念相对应的一个语言概念是 consultation，在 consultation 的定义中没有对 self 作上述限定，其消息转递过程中 self 指代的一直是方法的持有对象，最终消息的响应是方法持有对象做出的。目前在基于类的语言中，一般都支持 consultation，就是通常所说的对象方法的"引用"。

delegate 最初是在进行基于原型的(prototype-based)面向对象语言设计时提出的。根据上面的叙述可以看出它是一种基于原型的对象动态继承关系，由于它支持对象行为在运行期动态改变，因此自提出后一度引起很多研究者的兴趣，在其后设计的有影响的 SELF、NewtonScript、Cecil 等语言中实现了 delegate。但由于类型系统灵活，程序编制人员一般难以驾驭，这些语言都处于研究阶段。随着软件动态演化、动态变更等问题的提出，delegation 重新受到关注，并被考虑加入到基于类的语言中[6~8]。

3 Φ 对象演算

本节介绍一种命令式、带有 delegate* 的 Φ 对象演算，通过给出其语法和语义的形式定义，刻画 delegation 语言机制的本质特征，并说明其实现方法。

3.1 语法

Φ 演算的语法见表 1。

表 1 Φ 演算的语法

a,b::=	项
$[[m_i=b_i, i\in 1,\cdots,n \parallel d_j=\varsigma_j, j\in 1,\cdots,k]]$	对象(object)
let x=a in b	let
clone (a)	克隆(clone)
self	消息接收对象
a. m<+b	方法添加(method addition)
a◇. m	带有 delegate 成分的方法调用(delegate method invocation)
a. m◇<=b	带有 delegate 成分的方法更改(delegate method modification)

上表中，a,b 是 Φ 演算的项，包括对象、方法调用、方法更改、方法添加、let、克隆和消息的初始接收对象 self。

类似文[9]，Φ 演算将对象定义为一组方法的集合。由于带有 delegation 成分，因此对象记为 $o=[[m_i=b_i, i\in 1,\cdots,n \parallel d_j=\varsigma_j, j\in 1,\cdots,k]]$，其含义是：对象由两部分组成，一部分是对象自身实现的方法，另一部分是对象的父对象。其中，bi 是方法体，mi 是方法名。为简单起见，Φ 演算没有显式地表示域(field)，而是把域以一个相应的方法替代，其方法体为一个常量。对象的父对象用其地址来表示。如果对象的父对象不存在，则对象的父对象部分表示为空。

方法调用操作表示为"◇."，方法更改操作表示为"◇<=="，方法添加操作表示为"<+"。

克隆操作用来创建对象，clone(a)得到的对象和对象 a 是相同的，通过方法更改和方法添加可以更改克隆得到的对象。

Self 指代消息初始接收对象。任何一个发送给对象的消息，最终的响应必须和 self 绑定。

3.2 操作语义

类似 δ 演算[10]，Φ 演算的操作语义把二元组〈项，存储〉(〈Term,Store〉)重写为〈结果，存储〉(〈Result,Store〉)。重写关系"⇝"的基调是：

$\rightsquigarrow$: Term×Store → Result ×Store

其中，各参数的含义如下：

Store =({self} → Address) ∪ (Address → Object)

Result ={null} ∪ Address

Object ={$[[m_1 : v_1, m_2 : v_2, m_3 : v_3, \cdots, m_n : v_n]]$}

($m_1, m_2, \cdots, m_n$ 是方法名，$v_1, v_2, \cdots, v_n$(Result)

上述式子含义如下：1)Store 是 self 集到地址集、地址集到对象集的有限(finite)映射。2)Result 是地址集或空集。定义 Result 可以为空值，是为了表示异常(exception)。3)Object 是对象集，每个对象是方法名集到结果集的有限映射。

(1)方法查找函数

查找一个方法从某一个对象所实现的方法开始，如果该对象中没有查找到该方法，则在该对象的父对象中查找。若在父对象中仍没找到，则沿着父对象的父对象查找。这个过程持续下去，直到找到所需查找的方法或待查找对象的地址变为空。如果找到，返回所找到方法的方法体和地址；若查找不到，则为未定义。

约定 Udf 表示未定义，则 Look(σ,τ,m)函数可定义如下。

其中，各参数含义是：

σ —— 存储

τ —— 待查找对象的地址

m —— 待查找的方法

函数调用结果为：所找到方法的方法体 b 和地址 τ_m。

算法：

1) if $\sigma(\tau)=[[\cdots m=b\cdots \parallel \cdots]]$ //自身实现了方法 m
2) then return b, τ_m
3) else if $\sigma(\tau)=[[\cdots d_i=\tau_i\cdots \parallel \cdots]]$ //得到父对象地址，在父对象中查找
4) then
5) loop i
6) $\tau=\tau_i$
7) Look(σ,τ,b)
8) end loop
9) else if τ=NULL
10) then return Udf //方法未定义

(2)约定：存储记为 σ，地址记为 τ。

(3)定义存储上的操作：

$\sigma(\tau<=m,b)$，其含义是，将 τ 处的 m 方法的方法体替换为 b；

* 本文中 delegate 指在对象将发送给它的消息自动转发给其了对象的过程，delegation 指的是实现 delegate 过程的语言机制。

$\sigma(\tau<+m,b)$，其含义是，在 τ 处添加方法 m，m 的方法体为 b。

(4)Φ 演算操作语义如图 1 所示。图中各项(term)的规约规则如下：

1)对象：归约为一组地址，这组地址与对象的方法、对象的父对象地址对应。

2)self：归约为消息接收者的地址。

3)克隆对象 a：首先，归约对象 a，假设结果为 τ(τ 表示 $\{\tau_i, i\in 1,\cdots,n\}$，以下同)；接着，将 $\tau'(\tau' \notin \sigma)$ 映射到对象 b；最后，拷贝对象 a 到 τ' 处，并返回克隆得到的对象 b 的地址。

4)方法添加：归约规则与方法更改类似，不同之处是方法添加之前对象 a 不包含方法 m。

5)带有 delegate 的方法调用：带有 delegate 的方法调用：首先，归约消息接收者对象 a，得到其地址，假设为 τ；接着，使用 $Look(\sigma,\tau,m)$ 查找名为 m 的方法，得到 m 的方法体 b 及存放的地址 τ_m；最后，将 self 设置为消息接收者 a 的地址 τ，计算出 b 的结果。此处，不管 Look 函数找到的 τ_m 是否属于 a 的地址集，参数 self 的值都设为对象 a 的地址，也就是说，不论消息接收对象 a 本身实现 m 方法与否，self 都指代对象 a，这就保证了消息的响应方式是 delegation 而非 consultation。

通过 Look 函数可以查找到待调用方法的方法体，设置正确的 self 参数则可以保证消息的响应始终是消息的初始接受对象，由此，“有 delegate 的方法调用”既可以表达对某个对象本身实现了的方法的调用，也可以表达对一个对象的父对象所实现方法的调用。

6)带有 delegate 的方法更改：首先，归约对象 a，得到其地址，假设为 τ；接着，使用 $Look(\sigma,\tau,m)$ 查找名为 m 的方法，得到 m 的方法体 b 及存放的地址 ι_m；最后，将 ι 处名为 m 方法的方法体用 b 替换，返回 a。

“带有 delegate 的方法更改”可以灵活地表达多种情形下的方法更改，它既可对某个对象本身所实现的方法进行更改，也可对某个对象的父对象(该对象的父对象是可动态改变的)所实现的方法进行更改。

7)let $x=a$ in b：首先，归约对象 a，假设其结果为 v；接着，将变量 x 的值设为 v'，并对项 b 归约，得到的结果即为最终结果。

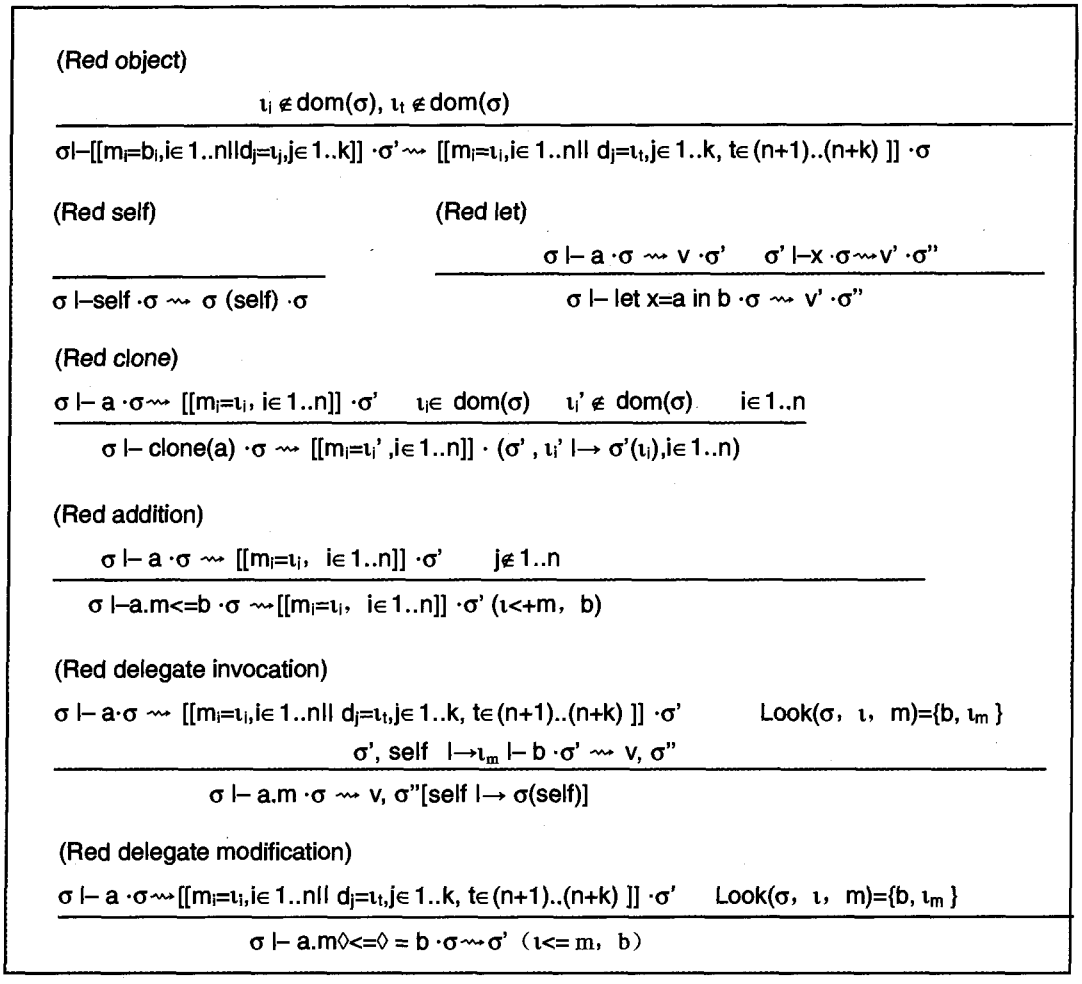

图 1 Φ 演算的操作语义

4 实例研究

基于上述分析，我们设计并实现了一种 delegation 机制，应用于我们与南京电视台合作的“城域网海量视听信息系统”项目中，解决其中媒体传输策略的动态切换问题：当用户需求量很大时，系统能自动切换为速度快但视频质量不高的传输策略，从而保证传输速度；而当用户需求量小时，系统又能自动切换为视频质量为标准的或者较高的传输策略。

其实现方法如图 2 所示。其中，VideoServer 的功能是控制和维护视频服务器的视频流传输；VideoDisplay 则定义视频流的传输策略。VideoDisplay 中有一个方法 display，VideoDisplay 的各子类均需实现此方法，不同的实现对应不同的传输策略。

VideoDisplay 与 VideoServer 之间存在 delegation 关系。

系统运行期间，若调用 VideoTransfer 的实例 aVideoTransfe 的 display 方法，实际执行的是 aVideoTransfer 父对象(根据实际用户流量选择 aFastTransfer、aStandardTransfer、aHighQualityTransfer 之一)的 display 方法，但执行时的上下文为 aVideoTransfe 对象的上下文。

实验表明[9]，采用传统的基于类继承方式实现上述功能，需书写代码 3,835bytes，而采用 delegation 实现则需 2,768 bytes，代码量减少了 20%。同时，由图 2 可看出，媒体传输策略的切换只需更改 aVideoTransfer 的父对象，而这可在系统运行时动态进行，不需重新编译源代码，大大提高了系统的灵活性，为系统动态更新、动态演化提供了有力支持。

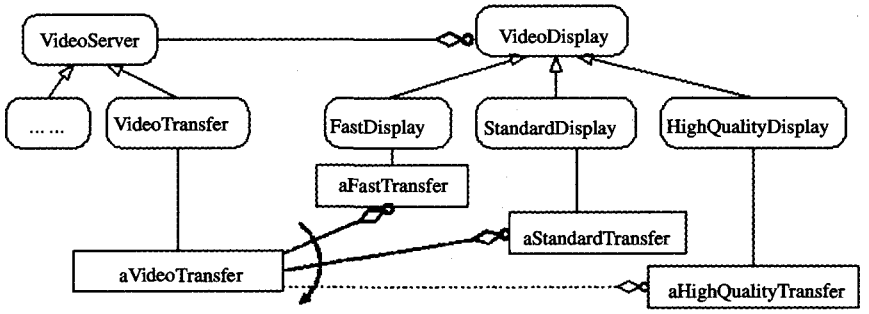

图 2　采用 delegation 实现媒体传输策略的动态切换

5　与本文相关的研究工作

文[10～16]都对 delegation 的动态特性进行了研究，但只有文[10,11]中提出用基于对象的演算刻画 delegation 的特征。文[10]提出一种称为 ζ 的对象演算，在 ζ 演算中，对象被定义为方法的集合，在对象上可以执行方法调用、方法变更等操作。文[11]则提出了一种命令式、带有 delegate 的 δ 对象演算。类似 ζ 演算，δ 演算中对象也定义为方法的集合。在文[11]中，对象的方法包括对象自身实现的方法和若干 delegate 方法。在对象上可以执行的操作包括方法调用、方法更改、方法删除、delegate 方法调用、delegate 方法更改、delegate 方法删除等。

本文提出的 Φ 对象演算与 ζ 演算不同之处在于：ζ 演算采用编码(encoding)的方式表达 delegation，而 Φ 对象演算为更好地研究引入 delegation 语言成分后对象在运行期间的特性，采用显式(explicit)的方法表达 delegation，并增加了 self 项。Φ 演算与 δ 演算的不同之处在于：在 δ 演算中，对象的组成分为两个部分，第 2 部分是对象通过共享得到的所有方法，而在 Φ 演算中，第二部分是对象的所有父对象，因为这样可方便地表达出一个对象共享某个父对象的所有方法，而不是仅限于列举出的方法；同时，这也更符合 delegation 的惯用法。Self 等语言也是采用列举对象的父对象这种办法来表达对象之间的方法共享。

结束语　本文提出一种带有 delegation 语言成分的 Φ 对象演算，并给出该演算系统的操作语义。对 Φ 演算的研究表明，在程序设计语言中引入 delegation 成分有如下两点好处：其一，通过 delegation 可以实现对象间方法共享，而且由于共享得到的父对象的方法并不固定作为子对象的一部分，是在运行期通过方法调用动态取得的，因此，基于 delegation 的对象方法共享可以节省空间；其二，通过更改 delegate 对象的方法或改变 delegate 对象，对象可在运行期动态扩展自己，而且这种改变会波及到关联的所有子对象。

参 考 文 献

1　C # Language Specification. http://msdn. microsoft. com/vcsharp/programming/language

2　Kniesel G. Dynamic Object-Based Inheritance with Subtyping: [PhD Thesis]. Computer Science Department III, University of Bonn, July 2000

3　Lieberman H. Using Prototypical Objects to Implement Shared Behavior in Object-oriented Systems. In: OOPSLA'86 Conference Proceedings (Portland, Oregon, Sept. 26-Oct. 2). ACM SIGPLAN, 1986, 21(11) :214～223

4　Milner R. Elements of Interaction - Turing Award Lecture. Commun, ACM, 1993, 36(1): 78～89

5　Abadi M. Baby Modula-3 and a Theory of Objects. Journal of Functional Programming, 1994, 4(2): 249～283

6　Kniesel G. Delegation for Java: API or Language Extension? [Technical report]. IAI-TR-98-4. ISSN 0944-8535, CS Dept III, University of Bonn, Germany, May 1998. http://javalab. cs. uni-bonn. de/research/darwin/

7　Ostermann K. Dynamically Composable Collaborations with Delegation Layers. In: Proceedings of the 16th European Conference on Object-oriented Programming (ECOOP), Malaga, Spain, 2002

8　Viega J, Tutt B, Behrends R. Automated Delegation is a Viable Alternative to Multiple Inheritance in Class-based Languages: [Technical Report]. CS-98-03. 2, 1998

9　崔琳，许满武，杨群. 一种 Delegate 机制的设计与分析. 计算机科学，2004，31(5)

10　Abadi M, Cardelli L. An Imperative Object Calculus: Basic Typing and Soundness. In: SIPL'95 - Proc Second ACM SIGPLAN Workshop on State in Programming Languages: [Technical Report UIUCDCS-R-95-1990]. Department of Computer Science, University of Illinoist at Urbana-Champaign, 1995

11　Anderson C. Delta - An Imperative Object Based Calculus with Delegation: [Full Technical report]. http://www. binarylord. com/work/delta. pdf, 2002

12　Fisher K, Honsell F, Mitchell J C. A lambda calculus of objects and method specialization. Nordic Journal of Computing, Spring 1994, 1(1): 3～37

13　Fisher K, Mitchell J. A Delegation-based object calculus with subtyping. In: Fundamentals of Computation Theory (FCT'95). Springer LNCS, 1995

14　Bono V, Bugliesi M, Liquori L. A lambda calculus of incomplete objects. In: Proc. MFCS '96, Springer LNCS 1113, 1996. 218～229

15　Bono V, Fisher K. An imperative first-order calculus with object extension. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 1998

16　Drossopoulou S, Damiani F, Dezani-Ciancaglini M, et al. Fickle: Dynamic object re-classication. In: ECOOP '01, LNCS 2072, Springer, 2001. 130～ 149