第5卷第2期 软件学报 Vol. 5, No. 2
1994年2月 JOURNAL OF SOFTWARE Feb. 1994

反射对象的概念、结构和实现*

奚建清 胡守仁

(国防科技大学计算机系,长沙 410073)

摘要 本文较系统地介绍了面向对象数据库语言ROL中对象的计算反射的思想和实现方法.反射对象的引入是为了解决对象库中复杂的管理问题,并为设计智能化程度更高的对象提供一个良好的基础.本文还讨论了反射结构带来的问题,即封装性问题和效率问题,以及一种解决方法.

关键词 计算反射,面向对象程序设计语言,面向对象数据库.

ROD(reflective object data base)是目前我们正在开发的基于面向对象语言ROL的面向对象数据库(OODB)系统,ROL是综合了CLOS[1]以及ObjVlisp[2]语言的一些特点的面向对象数据库语言,OODB是面向对象语言(OOPL)和数据库管理功能的基本无缝隙的集成,关于OOPL已经有许多文章作了介绍,如CLOS,Smalltalk.对象库是包含大量对象的系统,它提供了编程、访问、查询、和管理这些对象的手段和方法,与传统的关系库相比,对象库以对象为基本元素,对象不同于关系或元组,它是一个逻辑结构和行为封装在一起的计算单位,在概念上可以认为是相对独立的主动客体.OODB中的库管理也比关系库复杂,这些功能包括,对象的访问权限与访问方法,对象的版本控制,永久对象,远程对象访问(分布式情形),对象状态(已存在或尚未产生,并发执行情形),对象的聚集(cluster)等,更详细的介绍见[3].

如此复杂的管理过程需要很多的信息及控制(如对象的历史信息),没有清晰简单的信息生成及组织方式和高效的控制手段,整个OODB系统就会变得十分复杂和难于理解.为此,我们引入了反射对象的概念(Reflecting object).反射对象不仅使得整个对象系统的框架结构更为清晰,而且也为进一步提高和设计对象的智能化行为的程度提供了更好的基础.

1 计算反射

在[4—6]中介绍了一般的计算反射思想概念和例子,正如Maes所述的那样,计算反射是反射系统呈现的一种行为,这个系统是一个计算系统,它能因果相关地(in a causally

* 本文1991年10月16日收到.1991年11月4日定稿

本研究课题由国家自然科学基金资助.作者奚建清,32岁,1992年博士毕业于国防科技大学,主要研究领域为OODB,认知模型,OOPL,OOKB.胡守仁,68岁,教授,主要研究领域为计算机体系结构,人工智能.

本文通讯联系人:奚建清,江阴 214431,江苏省江阴市103信箱505号

connected way)对自身进行处理. Smith 又据计算单位所表示的内容和推理能力划分了反射和内射(introspective)系统,一般说来,内射的系统是指系统能描述的(部分)内部状态,结构及性质的系统,并能对之进行因果相关性推理的系统,这些系统诸如引入了 Believe, Know, Consistent 等有关系统内部状态的谓词的系统,以及象现在一般的元级程行设计,3-lisp 语言等系统就是这样一些系统,而反射的系统不仅能因果相关地描述和计算自身的内部结构,更重要的是能把自身置于(局部)环境中用"局外人"的眼光看待自己,从而能表示出包括自身在内的局部环境并对之进行计算,换句话说,反射系统是能把自己作为某种参数值(包括自己的一些局部环境因素)引入原来的省缺表示结构中,如把"在左边"转换成"在我左边",又如把"吃"转换成"吃饭","吃什么",对局部(动态)环境的表示以及自身作为参数大大扩充了表示的语义,如可以对诸如"here"、"now"、"在上边"、"本单位"等语法结构进行推理,这些环境表示以及因果相关的保证,为系统的智能化行为提供了更好的条件和基础.

[8]中分析了基于过程模型的一般反射模型,指出了一般系统的"自身"概念以及对计算反射思想的理解和设计的重要性,正如 Maes 指出的那样,至今没有一个系统是真正反射的而只是内射的(在 Smith 意义下),也即"自身"一旦理解和设计成整个程序系统,把整个用户程序(系统)理解成一个整体,即反射塔中的最低级,则用户程序的反射(内射)行为一般只能在语言实现级上反映出来,只能靠语言或其实现提供的明显的或隐含的设施来实现反射计算,这些设施构成了这个用户程序的内部环境. 显然,用户要改变这个环境是很困难的或是不可能的.

在传统程序设计中,上述"自身"的理解是很正常的并且是很自然的,因为一般用户程序是由主模块(函数,过程)和子模块(函数,过程)构成,这些(子)功能模块相互间往往是单向的调用关系. 结构与主要功能由整个用户程序体现. 而在面向对象系统中,对象是构成用户系统的唯一或主要单位,每个对象是具有复杂结构以及复杂行为的整体,从而在对象系统中更自然的做法是为每个对象引入"自身"的概念. 这样做具有第一种做法所不能取得的优点:

1. 把对象作为具有反射能力的单位使得这种反射对象成为 Smith 意义下真正的反射系统,虽然整个对象世界是不反射的或是内射的. 这是因为在对象世界内,一个对象的局部环境以及其内部结构、行为均能明显地在对象级上表示出来.

2. 对象的"自身"、其环境、对象的反射及其它智能行为能够易被用户理解,反射的能力(或反射的理论相关性)能由用户控制,而不是由对象语言所限制.

3. 对象的计算反射能力能把对象的许多复杂行为局部地实现而无需使全局控制复杂化. 这种设计是符合当前软件系统分散控制的潮流的[5].

4. 这种对象具有"自身"的意义,使得此对象的设计者更着眼于把对象作为行为的主体,从而能注重于此对象的环境或与其它对象的交互作用,有可能设计出智能化程度更高的对象.

2　对象的"自身"

在一个典型的 OOPL 或 OODB 中,对象具有它的类规定的结构以及赋予的一组行为方法,从而一个对象的行为由它的结构内容和类规定的结构语义和方法过程以及解释器(包括

机器硬件)决定.对象还处在和其它对象复杂的交互作用中,这包括访问或激发其它对象(包括这个对象自己),也包括被其它对象所访问或激发,以及一些相对静态的关系,如类和这个实例对象的模板一致关系,和其它实例的结构限制关系,如某个实例变量(iv)即属性的值不能大于另一对象的相应iv值,一个对象的部分局部与内部环境可刻划在图1中.

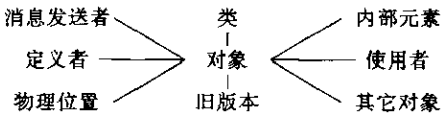

图1

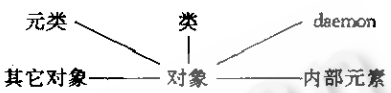

图2

我们定义一个对象的自身由它的内部结构和它的所有(局部)关系组成.在一般的OO模型中,关于一个对象的计算信息,如它的类对象,它的元素一般是该对象可见的,也即在它的方法中能访问到这些关系(当然,修改这些关系可能要引起某种检查),而对有些关系,如它的内部物理结构,它的访问者,消息发送者,它自己的物理地址在一般的语言如Smalltalk,C++中是不可见的,它们蕴含在省缺的实现结构中,是一个对象不能访问到的.显然,要使一个对象能对这些自身信息进行判断,必须把这些关系通过某种结构来明显地表示出来,一个具有反射功能的对象就是能利用它的内部或局部环境信息,进行分析判断和决策并能影响这些信息的对象.

要明显地表示一种信息,在程序设计语言中有两种方法,一是通过一个对应的语法成份,如一个变量,函数或过程来表示出这种信息,如Smalltalk中的伪变量'Self'和'Super'、Objvlisp中的函数'class-of'、C++中的'this',这些变量或函数由系统提供实现,另一种方法是用一个通用的过程通过结构或关系参数来访问相应的计算信息,实际上ObjVlisp中的函数iv-value,CLOS中的函数'slot-value'均是这种过程,它通过一个属性名或槽名来访问到一个对象的元素、属性或一种关系值,显然,后一种方法提供了用户扩展一个对象的结构关系的途径.而在第一种方法中,这些所谓的反射术语[5]是固定的.

用什么方法来表达一个对象的计算信息,以及在加入反射后怎样保持系统的原有性质则是设计计算反射系统时必须要考虑的.

3 ROL中反射对象的结构

在OOPL模型中加入对象的反射结构和行为,必须与OO原有的结构和风格相容,这包括基于类,类作为对象,(多)继承,封装性等OOPL的风格.这一节就前面列出的几种关系介绍在ROL模型中的实现结构.

3.1 类提供的信息

任一个对象的类作为该对象的模板提供了这个对象的共享的结构和行为的描述,由于类在ROL中是一个对象,故可以通过访问自己的类来获得关于自身结构描述、iv类型、访问者、功能和行为描述等信息,另外,在ROL中,元类提供了这个对象的物理实现细节.这些类提供的信息可以通过class-of函数(得到一个类名)、iv-value函数(访问对象的实例

变量)、Self 局部变量以及属性名访问到.

3.2 对象本身的信息

一个对象通过其逻辑结构或属性名联结着有关值或其它对象,在任何一个 OO 语言中,这个对象的方法体均可以访问到这些属性值,这种能力是面向对象程序设计的主要基础之一.

3.3 动态运行环境

能够高效地提供一个对象被激活时的许多有关局部动态信息,构筑了对象能在自己的行为中作动态判断,从而使该对象的行为更智能,更强壮(robust)的基础,在 ObjVlisp 中这种动态环境包括变量 self、selector 和 classes,分别表示当前哪个对象被激活,被激活运行的方法名和此对象的类的当前超类序列(用于方法体中过程 runsupers),在 ROL 中我们不仅提供了上述动态环境,而且还提供了两种重要的环境因素 sender 和 caller,sender 提供了消息发送者,caller 为 sender 发消息时运行的方法名. sender 和 caller 为动态统计和生成一个对象的 cluster、对象之间的对话式通信等能力提供了基础. 在以后的版本中,我们拟扩充 sender 和 caller 为函数 sender—of,caller—of.

3.4 daemon 对象

一个对象除了涉及上面讲的信息之外,还有许多"杂"信息,包括上面提及的,设计者、使用者及权限、与其它对象的关系和限制、自身的结构限制、版本及处理等等. 这些计算信息的管理不能由系统提供一些变量或过程来表示,因为这些信息的管理是很复杂的,而且处理方式可能需要由用户自己决定,另外,这些信息如果由类或一个对象本身来处理,则有可能引起类或这个对象的概念混乱,即 Maes 所谓的对象的问题域和计算域信息的混杂造成的对象语义不清晰.

为了管理这些信息,我们依照 Maes 给出的模型为每个需要处理上述(部分)信息的对象引入了一个对象,Maes 称之为元对象(meta—object),而在 ROL 中称为 daemon(精灵)对象. 一个 daemon 所管理的对象称为它的 client(顾客),目前 daemon 管理的内容有:管理并负责 client 版本;统计并生成 client 的簇集(cluster);控制对 client 的访问(实现 view 功能);控制 client 的结构以及和其它对象的关系;重解释 client 对某些消息的行为;保存 client 的一些特殊属性和方法;代表远程 client 在本节点的行为等等.

上述 4 种对象自身信息的表示可以由图 2 描述.

4 daemon 对象与封装原则

daemon 的引入使得对象的环境信息既可以集中地管理,且这种复杂管理的行为基本不会使全局控制流更复杂,并且为用户修改、扩展一个对象的反射信息和行为提供了便利的条件,这是因为 daemon 也是一种对象,为某个类的实例,从而享有其它对象的同样性质.

但这种模型虽然解决了 Maes 所谓的信息混乱问题,另一方面却和对象的封装原则相抵触. daemon 概念的引入需要对象存储关于问题域的信息,daemon 存储和控制对象的诸信息和行为,但由于问题域的信息和对象的计算信息(特别就对象的使用者而言)往往是分不开的,或不细加区分,故对象和其 daemon 的区分给用户在某种程度上同样引入了混乱,而

且造成了对象封装性的破坏,因为封装性原则要求一个对象的内部信息结构存储和实现均对用户透明,用户只知道对象呈现给它的一个界面.

上述问题的根源在于 daemon(或元对象)的引入是基于把信息分成问题域和计算域的出发点.我们在本文不打算讨论这种做法是否合适,为了保存 ROL 中 daemon 的对象性,我们引入了一组设计协议来探讨解决上述问题的方法,这组协议的主要内容如下:

A. 一个 daemon 负责控制其 client 对象的行为(指对访问和消息的反应);

B. 一个对象的 daemon 对此对象的用户是透明的;

C. daemon 的方法的激活或属性的访问可向 client 发消息或访问 client 来实现;

D. client 和 daemon 的方法如果同名按 client 的方法计算,属性也同样处理.

上述 c、d 要求一个对象被加上 daemon 时,系统应判断方法名和属性名的冲突并提醒 daemon 的设计者.

daemon 对用户的透明性使得用户在考虑涉及一个对象的信息或关系时,只要访问这个对象即可,用户不需要去判别是访问一个对象,还是去访问其 daemon.从而保证了对象的封装性.

5 反射与效率

ROL 的部分设计目标是提供一个高效、灵活、易于扩充和修改的程序设计环境,Daemon 的引入使得系统原始过程 send 函数必须判定一个对象是否具有 daemon,然后才确定具体执行哪个方法体,即是类规定的方法体还是由 daemon 重解释或定义的方法体,对属性访问函数也需要同样处理.这种动态的判定是较消耗时间的,下面主要讲 send 函数的设计.为了既方便对象动态地对自己的行为进行修改,又能高效地进行上述方法选择.我们把类规定的方法和 daemon 规定的其 client 的方法同等看待,即一起放在称为方法表(m—table)的全局数据结构中,同时类名和 client 的名字也一起放在此表的合适位置,从而 send 的算法结构如下.

a. 据方法名获得 m—table 中的入口.

b. 消息接收者是否出现在表中,如是,取相应的方法体执行并返回.

c. 取类规定的方法体执行并返回.如没有则转 d.

d. 向对象的 daemon 发送上述消息.

上述算法中与 daemon 有关的只是第二和第四步,故全局控制流比较简单,并且由于上述算法的时间消耗仍以第二步为主,故算法的时空复杂度不变,保证了 daemon 的引入不影响效率.下面是用 common lisp 实现的上述算法的程序段:

```
.....
(or
  (and (setq flag t)
       (setq position (or (position o (cadr (get—entry m—table m)))
                          (setq flag nil)
                          (position (get—method—class o)
                                    (cadr (get—entry m—table m))))
```

```
(or (apply (nth position (caddr (get-entry m-table m)))
           (if flag (daemon-of o) o) m args) t))
(send (daemon-of o) m args))
.....
```

上述程序段中 o 表示一个对象，m 是消息名，daemon－of 函数计算对象的 daemon，get－entry 访问 m－table 中的一个项，get－method－class 通过继承语义找相应的方法体. 对象属性值的访问函数 'iv－value' 也类似处理.

另外，对象的局部动态环境 self，classes，selector 和 sender 均是通过高效的参数传递方式(传地址)来实现的，故对动态运行时间的影响可以忽略不计.

结　论：本文探讨了反射的思想在 ROL 对象世界中的一种实现方案，即反射的对象的结构和实现，反射使一个对象能访问其环境信息，从而提供了设计能利用这些信息的对象的基础，反射思想和技术主要是围绕着对"自身"的理解展开，从而比仅在语言级之间实现的结构和语义联系有更广泛的含义，这一点已在前面的介绍的动态环境和 daemon 中看到，类中的描述设施作为一种环境信息是共享了类作为模板这种 OO 特性. 本文还探讨了反射引起的封装性问题和解决方案，最后给出了一种实现方案，使得时空效率类似不引入反射的系统，我们已经在 SUN 工作站上用 common lisp 实现了一个单用户的 OODB 系统，并展开了包括计算反射在内的一系列的课题的研究.

参考文献

1 The common lisp object system: an overview ECOOP'87 Proc., 1987.

2 Briot J P, Cointe P. The ObjVlisp model: definition of a uniform, reflective and extensive object oriented language. in Advance in AI－2, 1987.

3 Zdonicki S B. Language and methodology for object oriented database environment. Proc. of the 6th Hawaii System Science Conference, 1986.

4 Jacques Ferner. Computational reflection in class based object oriented languages. OOPSLA'89 Proceedings, 1989.

5 Pattie Maes. Computational reflection. The Knowledge Engineering Review, 1988;3(1).

6 Brian C. Smith. Varieties of self reference, theoretical aspects of reasoning about knowledge. Proc. of 1986 Conf., 19－34.

7 Pattie Maes. Concepts and experiments in computational reflection. OOPSLA'87 Proceedings, Florida, 1987.

8 奚建清，胡守仁. 计算反射和面向对象程序设计(第二部分). 计算机科学，1991(4).

9 Ronard J Branchman. The future of knowledge representation. AAAI－90 Proceedings, 1990.

THE CONCEPT STRUCTURE AND IMPLEMENTATION OF REFLECTIVE OBJECTS

Xi Jianqing and Hu Shouren

(*Department of Computer Science, Changsha Institute of Technique, Changsha* 410073)

Abstract　Within the paper the idea and some implementation technique of reflective objects in OOPL ROL are introduced systematically. The use of reflective objects is for the

complicated management in OODB and for more intelligent behaviours of objects. The problems of encapsulation and efficiency arising from the introduction of reflection are also discussed with possible solutions.

Key words Computational reflection, OOPL, OODB.