第 4 卷第 1 期
1993 年 2 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 4, No. 1
Feb., 1993

人工智能软件中的面向对象程序设计

王怀民　陈火旺

(国防科技大学，长沙 410073)

OBJECT—ORIENTED PROGRAMMING FOR AI SOFTWARE

Wang Huaimin and Chen Huowang

(*National University of Defense Technology, Changsha* 410073)

Abstract　In the paper, we propose three levels of understanding for object-oriented programming (OOP): philosophy level, methodology level, and Language/environment level. We consider "object as intelligent agent" based on knowledge representation hypothesis. Along with the view, we discuss the methodology and architecture of OOP in AI software. ROOT, which is an AI programming Language and environment developed by our group, reflects the understanding discussed in the paper.

摘要　面向对象的程序设计(OOP)以多种面貌广泛地出现在 AI 系统中. 面向 AI 的 OOP 模型与结构依赖于对 AI 中对象的理解，以及相应的 AI 软件开发方法. 本文提出在三个层次上，即世界观的层次上，方法论的层次上以及语言与环境的层次上，理解 OOP. 我们根据 AI 中知识表示假设，将 AI 软件中的对象规定为智能代理者(intelligent agent)，并在此基础上讨论了 AI 软件的 OOP 方法学和结构. 最后我们介绍反映上述思想的人工智能语言及其环境 ROOT.

§1. 引　言

被称为八十年代结构程序设计的面向对象程序设计(OOP)不仅在软件工程(SE)领域产生了重大影响，而且已经成为基于知识的人工智能(AI)系统及其开发环境(简称为 AI 软件)的重要范型[1,3,11,14]，在 AI 软件中以多种形式表现出来：

直接用 OOP 语言设计 AI 软件　一些 AI 软件，例如基于知识的智能控制与制造系统，直接利用 Smalltalk—80，C++等 OOP 语言设计实现. 其特点是知识以状态的形式分布在控制过程的各个环节，程序的过程性代替了知识的形式表示.

本文 1990 年 11 月 16 日收到，1991 年 2 月 8 日定稿. 本课题由国家 863 高技术发展计划资助. 作者**王怀民**，1988 年获硕士学位，1992 年 9 月获博士学位，主要研究领域为软件工程、人工智能语言及其环境. **陈火旺**，教授，博士导师，主要研究领域为软件工程、人工智能、计算机科学理论.

建立在函数/逻辑语言上的OOP环境 函数范型的符号处理语言LISP是具有较长历史和丰富环境的AI语言,人们自然希望在LISP上建立OOP环境. Flavors, LOOPS和ObjVLisp是这一思想的典型体现[8,2]. 这一工作的核心是在LISP环境上定义有关对象定义,方法定义和消息传递等支持OOP的函数,从而在LISP上建立OOP的外壳. 同样在具有元级描述机制的Prolog上也可以类似地建立OOP外壳[10].

建立在抽象数据类型(ADT)上的OOP ADT与OOP的许多共性使人们希望建立两者的联系. J. A. Goguen等的工作Fooplog就是试图在ADT中引入反映状态变化的属性和反映继承关系的继承说明以扩展ADT建立OOP[5].

OOP作为多范型AI环境的一部分 一种观点认为多种范型并存在的AI语言与环境是一种合理和现实的选择. 知识系统开发工具KEE就是一种集知识的规则表示,框架表示和OOP于一体的集成环境[4].

OOP仅作为AI软件的整体结构 这种OOP强调AI软件的模块成分是对象,整体结构是面向对象结构. 而对象的内部结构不追求统一的纯OOP原则,可以是规则、逻辑、函数等范型的组织形式. Orient84/K[6],NEXPERT OBJECT等工作反映了这一思想.

OOP方法在AI软件中的多样性不仅说明了OOP在AI中的广泛应用,而且反映了人们对OOP不同层次的理解,以及在AI中不同深度的应用. 下面,我们首先在三个层次上一般地讨论对OOP的理解,然后依据知识表示假设提出对象作为智能代理者的OOP观点和程序设计方法. 最后介绍基于这一观点的AI语言和环境ROOT的OOP结构.

§2. 关于OOP不同层次的理解

人们对OOP的认识反映在面向对象的世界观,面向对象的方法学和OOP语言等不同层次上.

1)面向对象的世界观

OOP被广泛接受的一个重要原因是面向对象的世界观,更接近于人们对现实世界的自然感受. [9]对此作了如下概括:"OOP的基本哲学是使程序尽可能反映人们正在考虑的现实世界的那一部分,这样常常易于对程序的理解,以及对程序所描述事物的全面把握. 因为人们是遵循现实世界的发展认识世界的,越接近于用这种方法思考程序设计,就越容易设计和理解程序."OOP是将"程序的执行看成模拟现实世界(包括人的精神世界)某一部分行为的物理模型.""一个物理模型涉及一组对象,对象由属性和动作序列刻划."

尽管人们对OOP的特点有各种各样的概括,但对OOP世界观的认识基本上是一致的.

2)OOP方法学

OOP方法学是OOP世界观在程序设计方法上的具体反映,它与OOP语言的性质紧密联系在一起. 根据[13]的观点,对象=属性+方法. 对象之间通过消息传递建立联系. 对象通过一个方法的执行响应一个消息. 一组具有共性的对象组成一类,而类之间又构成了继承关系.

按照上述OOP的图景,OOP方法包括:确定系统中涉及的对象和类;确定对象,以及类之间的联系;完成对象中属性和方法的设计.

由于OOP具有信息隐蔽,数据抽象,动态约束和继承等特性,因此人们也说:

OOP方法=抽象数据+动态约束+继承

由于基于OOP的软件具有模块性,强壮性,可理解性,可重用性,可扩展性,开放性等一系列优点,OOP在SE领域被广泛采用.

3)OOP语言

OOP方法学体现在OOP语言中.一般认为,对象,类,消息,方法和继承是OOP语言重要的概念.尽管人们普遍认为没有绝对标准的OOP语言,但反映在Smalltalk－80中的语言概念成为设计和衡量OOP语言的相对标准.我们把它归纳为以下原则:一切皆为对象;对象是属性及其操作方法的有机构成;对象是系统的活跃成分;对象是类的实例,实例关系和继承关系是对象之间的静态关系;消息传递是计算的唯一形式,也是对象之间动态联系的唯一形式,方法是消息的序列.

实际上,包括Smalltalk－80在内的许多OOP语言都没有绝对恪守上述原则.因为这些原则不仅给OOP语言的实现技术和实现效率带来许多问题,而且其简单统一的倾向也与面向对象的世界观产生了冲突.由于OOP源于一种世界观,而不是一种局部具体的理论或方法,因此OOP在SE,DB(数据库)或AI中的表现形式由于领域问题的不同而变得多种多样.

§3. 从AI的角度看OOP

传统AI研究遵循了称之为知识表示假设(Knowledge representation hypothesis)的途径.Smith将这一假设归纳为[12]:任何机械的表现智能的过程(process)由以下结构成分组成:a)人们作为外部观察者,将被处理过程作用的知识当作命题自然地表示出来.b)独立于这些知识的外部语义属性,形式地引发表现知识意义的行为,这些行为与知识有着实质性的因果联系.

知识表示假设规定智能代理者(intelligent agent)由知识表示结构和作用于此结构上的处理过程组成.知识表示结构与相应的处理过程密不可分,互为存在.知识表示结构的意义通过处理过程的行为体现.简单地讲:智能代理者＝知识表示结构＋处理方法.

与对象的一般结构相比较,智能代理者正是对象在AI软件中的一种具体解释.我们认为把对象作为智能代理者对于复杂的AI软件设计具有十分重要的意义.

第一代AI软件可以说是一种单个智能代理者结构的AI系统,这种软件结构的问题是知识表示结构与一种统一的知识处理方法以隐含形式相联系.知识处理方法的单一性及其在系统实现中的"固化"使人们在AI软件的设计中产生一种错觉:AI软件设计的核心是知识表示结构的设计,这种结构必须为这种单一的推理机制理解.也就是说,知识表示结构是第一性的.E. Feigenbaum甚至把这类专家系统称之为知识传输系统(knowledge delivery system).

对象作为智能代理者意味着:

1)AI软件不再是单智能代理者系统,而是由一组协同工作的对象构成.

2)AI软件设计不再是单纯的知识表示结构的设计,而是对象的设计.每个对象内部不仅涉及知识表示结构的设计,而且更主要的是具有决定意义的特殊的知识处理方法的设计.

3)AI软件的执行不再是简单重复性的问答.对象作为AI软件中能够自主进化的动态实体,可以根据自身的行为和整个系统的状态调整对象内部的知识表示结构,显示简单的成长过程.

人类认识和处理问题的一般过程是从一个认识基点出发曲折的探索过程.AI软件的开发

作为一种智力投入也应立足于一个认识基点.我们把这一认识基点规定为OOP结构.

OOP结构包括环境中已经建立的类以及类与对象的构造方式.

1)环境中的基本类

OOP环境中的基本类(记为SCLASS)是OOP结构的基础素质.SCLASS中汇集了一个阶段成熟的和常用的知识处理方法.SCLASS的设计与实现是人对OOP结构最初的智力灌输.SCLASS应具有非构造性和开放性的特点.非构造性是指SCLASS中的方法是不可分解的基本方法,用它们可以定义同一层次的其它方法,但其自身不可被同一层次的其它方法定义.开放性是指SCLASS中的方法允许被修改和丰富.SCLASS是OOP结构与其支持环境(即实现环境)相联系的窗口.

2)结构方式

类和对象的定义与结构方式是OOP结构的成长素质.类以及对象之间的关系包括一个静态关系,即继承结构;一个准静态关系,即实例化结构;和一个动态关系,即消息传递.

· 继承结构

对继承结构的认知意义人们强调的侧重点不尽相同.这里我们强调:子类对父类的继承是:1)子类对父类的信息共享,2)子类对父类一次丰富或细化.我们用规则(R1)表示继承结构.(R1)说如果A是类,那么新定义成分Def与A构成以A为父类的新类B.B中的Def是对A的丰富或细化.

$$\frac{\text{Class} \quad \text{A}}{\text{Class} \quad \text{B}:<\text{A},\text{Def}>} \qquad \text{(R1)}$$

图1(a)是子类B对父类A的引用关系图,图1(b)所示的继承链被解释为从B_0到B的不断逼真的开发过程.继承结构的起点通常是SCLASS.

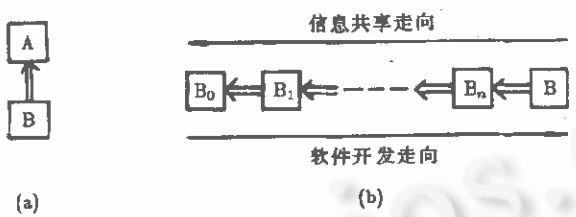

图1 类的继承关系

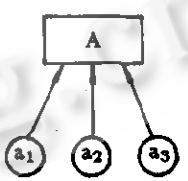

图2 实例化关系

· 实例化结构

类是一组对象的抽象,对象是类的实例化.我们用规则(R2)表示实例化关系.(R2)表示如果A是类,则A的实例a是对象.

$$\frac{\text{Class} \quad \text{A}}{\text{Object} \quad \text{a of A}} \qquad \text{(R2)}$$

图2是对象a_1,a_2,a_3对其类A的引用关系图.这里我们强调两点:(1)实例化结构是从一般到具体的结构.抽象是SE的重要原则,但是实例化结构不是刻划抽象行为的机制.我们只能由类A通过实例化机制得到具体的对象a_1,a_2,a_3,还不能由a_1,a_2,a_3通过某种抽象机制得到A.到目前为止OOP乃至整个AI技术还没有比较成熟的描述抽象过程的机制.OOP结构仅提供了描述抽象成分的语言设施——类.类的设计仍然依赖于人的智力投入.(2)按照纯统一的OOP语言原则,OOP结构中要求存在最一般的(元)类,OOP环境中所有的类和对象均

源于这个类的实例化.尽管一些OOP结构追求这种理想,但我们注意到:a)实例化过程不能反映从具体到一般的发现过程;b)如果把最一般的(元)类看成最抽象的事物,那么这种(元)类是不存在的;c)人们能够实现的最一般的(元)类实际上是类的定义模式[2].我们主张在对象作为智能代理者的观点下,类仅作为对象的静态抽象描述;类的定义通过OOP环境支持;类不再作为对象被元类的实例化生成;允许类似变量类型说明的静态实例化说明.

• 消息传递

消息传递是OOP结构中,对象之间协同工作的动态联系方式.我们主张知识表示结构不能独立成为对象,因此,对象内部的知识处理不被解释为消息传递行为.

§4. ROOT的OOP结构

ROOT[15,16,17]的设计反映了上节讨论的OOP观点.我们把ROOT的OOP结构概括为五个方面:1)对象作为AI系统中活跃的智能代理者;2)对象=知识单元+单元上的处理方法;3)知识单元是被方法处理的成分.它本身不能孤立地成为对象;4)类是对象的静态抽象描述,类本身不是对象.类与对象之间的静态关系是实例化关系,类与类之间的静态关系是继承关系;5)对象之间的动态联系是消息传递.对象内部的方法执行是条件重写驱动的大粒度计算反射.

图3表示了ROOT的面向对象的程序设计结构.

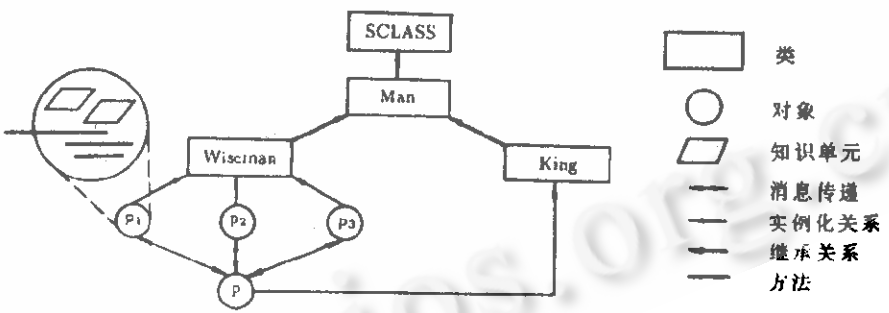

图3　ROOT的OOP结构例示

对象的结构与类的结构一致.类的说明包括父类说明,相关说明,知识单元说明,界面说明和方法说明,其语法形式为:

```
<declaration of class>::=Class<class name>:
{
<declaration of superclass>
<declaration of dependence>
<declaration of unit constant>
<declaration of unit variable>
<declaration of interface>
<declaration of method>
}
```

其中知识单元说明和方法说明是类说明的核心.一个知识单元是一组断言的集合.

<definition of unit>::=<unit name>:<unit>

<unit>::={<assertion>+(;)}

断言的一般形式为带不等式条件重写规则:

$$l: \quad tl_1=tr_1,\cdots,tl_n=tr_n,sl_1!=sl_1,\cdots,sl_m!=sr_m : r$$

其中l,r,tl_i,tr_i,i=1,…,n,sl_j,sr_j,j=1,…,m为一阶项.其意义为:如果条件$tl_i=tr_i$,(i=1,…,n),$sl_j\neq sr_j$,(j=1,…,m)成功,则l可以等价地归纳为r.我们把Horn子句$h:-p_1,\cdots,p_n$作为形如:$h:-p_1=true,\cdots,p_n=true:true$的重写规则的缩写形式.知识单元说明包括单元常量说明和单元变量说明.一个单元变量可作为实例变量被一个单元实例化,可被强制式的原语方法作用.

方法同样用上述形式的带不等式的条件重写规则定义.但方法与知识单元中的断言不同.方法说明是知识处理过程的描述.如果说知识单元中的断言是对目标论域的描述,那么方法是对如何利用知识单元中的知识进行问题求解的描述.因此方法是关于知识单元及断言的元级程序设计.知识单元中的断言在方法中被表示为断言模式表.原语方法通过知识单元名操作指定的知识单元,图4是ROOT中方法与知识单元之间的元级程序设计结构.

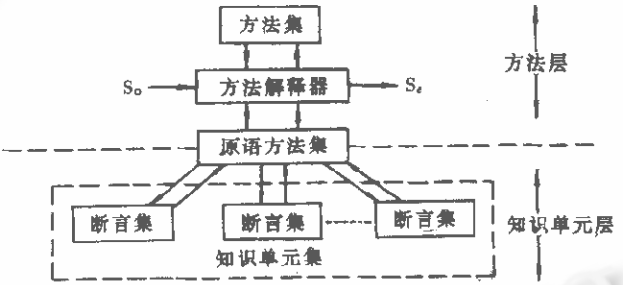

图4　方法的元级结构

消息的执行是引发另一个对象中一个方法执行.而原语方法(即SCLASS中实现的方法)的执行是反射到单元层直接完成与原语方法相联系的计算.从知识处理的意义上讲,方法的作用在于:1)使用原语方法处理知识单元;2)对原语方法的执行结果作出反映.因此,我们说方法的执行是条件重写驱动的计算反射过程,由于方法体中出现的原语方法和消息的执行不再是方法层上的条件重写,方法的执行不是方法层上严格的重写.

一些例子(如三个聪明人问题)表明ROOT的OOP结构有利我们开发面向智能代理者的AI软件;实现丰富的知识处理方法;模拟多智能代理者之间高级的智力行为.ROOT还可以作为分布式AI系统程序设计语言框架.ROOT V1.0已经在SUN3/260上成功实现.

§5. 结束语

基于成熟AI技术的软件产品(如基于知识的AI软件)已经广泛出现.借鉴SE的成果,研究这类软件产品的开发已经成为国际上广泛关注的工作[1,11].以孤立、静止的观点看待AI系统必然导致对AI的悲观情绪.因为人们总能从AI的局部成果中找到相对于人的局限性,从

而否定其中"真正"智能的存在.AI的希望在于将AI系统视为人—机集成系统,AI的目的在于促进人机之间更"聪明"更"和谐"的合作.其中AI语言及其环境是两者的媒介,它有两个重要作用:1)支持人对系统的智力投入——AI软件的开发与维护;2)支持AI软件的自主执行和进化.而基于AI的OOP的确为设计这种AI语言及其环境提供了十分有益的启发.

致谢:感谢高洪奎副教授在ROOT V1.0实现中给予的热情指导,感谢刘宏工程师,孙逊硕士,曹春春硕士在ROOT V1.0实现中的高效合作.同时感谢李晓毅硕士在ROOT前期研究给予的建设性合作.

参考文献

[1]Alonso, F., Mate, J. L., Knowledge Engineering Versus Software Engineering, Data & Knowledge Engineering 5 (1990), 79-91, North—Holland.

[2]Cointe, P., A Tutorial Introduction to Metaclass Architecture as Provided by Class Oriented Language, Proc. of the International Conf. on FGCS 1988, Edited by ICOT, 1988, 593—608.

[3]Firebaugh, M. W., Artificial Intelligence: A Knowledge—Based Approach, Boyd & Fraser Publishing Company, 1988.

[4]Gergoire, E., Evaluation of the Expert System Tools KEE and ART: A Case Study, Applied Artificial Intelligence 2 (1988), 1—23.

[5]Goguen, J. A., & Mesegner, J., Unifying Functional, Object—Oriented and Relational Programming with Logical Semantics, Research Directions in Object—Oriented Programming, 417—477, The MIT Press, 1987.

[6]Ishikawa, Y., Tokoro, M., Orient 84/K: An Object—Oriented Concurrent Programming Language for Knowledge Representation, Object—Oriented Concurrent Programming, A. Yonezawa & M. Tokoro(eds), 159—197, The MIT Press, 1987.

[7]Keravnou, E. T., & Washbrook, J., What is a Deep Expert System? An Analysis of the Architectural Requirements of Second—Generation Expert System, The Knowledge Engineering Review 4:3 (1988), 205—233.

[8]Luger, G. F., Stubblefield, W. A., Artificial Intelligence and the Design of Expert System, The Benjamin/Cummings, 1989.

[9]Madsen, O. L., & Pedesern, B. N., What Object—Oriented Programming may be and What it does not have to be, Proc. of European Conf. of Object—Oriented Programming, Norway, 1988, 1—20.

[10]Malpas, J., PROLOG, A Relational Language and Its Application, Prentice—Hall, 1987.

[11]Ramamoorthy, C. A., Shekhar, S., Gong, V., Software Development Support for AI Programs, Computer, 20: 1 (1987).

[12]Smith, B. C, Prologue to "Reflection and Semantics in a Procedural Language", Reading in Knowledge Representation, R. J. Brachman & H. J. Levesque(eds), 1985, 31—40.

[13]M. Stefik & Bobrow, D., Object—Oriented Programming: Themes and Variations, AI Magazine, 6:4, 1986, 40—62.

[14]Tello, E. R., Object—Oriented Programming for Artificial Intelligence: A Guide to Tools and System Design, Addison—Wesley Publishing Company, Inc. 1989.

[15]Wang Huaimin, Chen Huowang, Object as Intelligent Agent, Pro. of Int. Conf. of Youth Computer Scientists, 1991, Beijing.

[16]王怀民,陈火旺,基于重写技术和面向对象结构的AI基础语言——ROOT,知识工程进展1990,许卓群主编,35—44,1990.

[17]王怀民,陈火旺,对象作为智能代理者的面向对象程序设计,人工智能与智能计算机,1991.9.