元知识与反射原理

蔡希尧　陈　平　（西安电子科技大学）

摘　要

本文从元知识概念引出元计算和实现元计算的主要途径——反射机制。对反射的原理以及用面向对象的程序设计语言(OOPL)来实现反射进行了概括性讨论，并介绍了作者用基于编译的C++语言在计算反射方面所作的工作。

1、什么是元知识？[1],[2]

知识可以分成层次。元知识是指关于知识的知识。同样，关于理论的理论称为元理论，关于推理的推理称为元推理，关于计算的计算则称为元计算。

在人工智能和计算机科学的研究过程中，人们发现需要有元知识，需要有表示元知识、推导元定理的能力，需要有关于计算的计算。

在推理系统中，规则是知识，通常是针对特定域的。如何找出某个域的推理规则，还需要一种知识，这种知识不同于规则所包含的知识。一个表示系统持有某种知识，但对它们进行修改，则需要另外的知识，例如知识的结构如何？规则怎样表示？解题采用什么方法？这两个例子告诉我们，把规则所含的知识和寻找规则的知识区分开来，把表示系统持有的知识和修改表示的知识区分开来，是很必要的；换句话说，把描述一个域的语言和变元，同用以表征描述活动的语言和变元加以区分，是很必要的，也是非常重要的。

在计算系统中，为了有效地利用资源，适应变化，要求对计算过程加以控制。这种动态控制，应当是系统结构和环境同应用程序之间双向的交互作用基础上实现的，这就需要关于计算的计算。

一个知识系统，对同类的问题如果使用相同的推理步骤，不可能改善系统的性能。用户常常可以通过成功或失败的经验，发现带有普遍意义的事实，可以把它们做为以后对这类问题进行推理的引理，而不必从头开始。一般说来，检验一个事实是引理的实例，比证明这个事实要简单方便。

对于元知识的兴趣，几乎已经波及到人工智能的所有领域，包括知识表示、问题求解、专家系统和演绎数据库等方面，而推理控制则是提倡使用元知识最强烈的领域。在计算系统方面，由于面向对象程序设计走向实用，关于计算的计算也日益受到重视，并在迅速发展。

一个系统使用元知识，有许多好处，主要有：(1)在完成一个新的任务时，可以借助于已有的经验。这是一个有效的自我修正的形式。(2)在执行一个任务之前，可以利用元知识预计它的计算行为。这是一种自我估价的形式。(3)可以克服经典逻辑的限制，在不同的或变化的状态下进行推理。这就为非单调逻辑、省缺逻辑等提供了一种可能的工具。(4)在计算系统中，可用来实现对程序的动态修改。

2、知识表示对面向对象程序设计语言(OOPL)的要求

OOPL近年来日益受到人们的重视，它

不仅是功能很强的程序设计语言，而且也是知识表示的良好工具。用OOPL表示知识，兼备了语义网络和框架的优点。组成OOPL的要素是：

(1)对象(objects)：是一实体，是OOPL的基本数据结构。对象的规模可大可小，能适应不同应用的要求，它的值、变量和操作可赋予多态性。

(2)方法(methods)：相当于普通程序设计语言中的过程，当一个对象收到消息时，将执行操作。

(3)消息(messages)：它用来进行对象和对象之间，或对象与外部世界之间的通信，这一通信方式称为消息传递。

OOPL有许多和现有的语言不同的特点，它们是：

(1)类的继承性。具有相同特性的对象组合在一起构成类(class)，一个类或多个类的特性在定义新类时被再次使用，称为继承性。一个类可以从它的父类继承特性，而它的特性又可以被子类所继承。类的继承性是OOPL中的资源共享机制，建立在集合的概念之上。

(2)代理机制。这也是OOPL中的资源共享机制，但建立在原型的概念之上。在同类对象中，可任选一个做为原型，别的对象可以共享原型的特性，还可以有自己的其他特性。

(3)消息传递。这是OOPL中通信的唯一方式。

(4)密封性。每个对象严格定义其外部接口，尽可能少地暴露实现的细节，用户只能通过外部接口使用对象。此外，为继承者提供另一接口，使它们可以取得有关的信息。

(5)动态联编。在程序运行时可以加入对象，层次也可改变。

OOPL的这些特点，可用以较自然地表示客观世界，表示知识。知识表示包括定义概念，与概念相关的特性，各个实体之间的关系，OOPL可以很好地满足这些要求。一方面是在类的概念中嵌入了集合，对象被划分为集合，并形成等级，这对知识表示很有利。另一方面，依附于每个对象的属性，可用以表示关系，这也是知识表示所需要的。但是，OOPL需要为更好地表示知识加以扩充。(1)嵌入于类的概念中的集合如果是显式的，其他种类的有用的集合可以被引入。(2)关系也希望是显式的，这就是要给出对于这些关系的系统中对象的状态。(3)单重继承性常常是不够的，知识系统要求多重继承，而且有异常关系存在。有异常的多重继承问题是很难的[3]。(4)现有的面向对象的系统中，对象往往是局部性的，但在许多实际的场合，要求它们是全局性的。

对OOPL加以扩展以适应知识处理的要求，最直接的途径是在OOPL中引入反射功能，而OOPL也是反射功能最容易实现的一种语言。对象所具有的优良特性，能够扮演许多角色，它不仅可以用来对域中的问题进行计算，也可以对如何实现这类计算进行计算。对象既能表示域中的事物，也能表示自身，例如它自己是什么时候建立的，满足哪些限制条件。下面我们将介绍在OOPL中引入反射功能的原理。

3、反射原理

对于计算系统中的反射的定义，不同作者有不同的提法，但基本方面是一致的，参考[4]和[5]，我们给出以下的定义："在计算系统中，反射是关于并作用于计算系统本身的推理过程，这一活动涉及访问并部分地改变整个系统，影响自身的计算。反射作用的目的在于动态地修改计算系统的内部结构，使得计算更为有效。反射系统是因果关联的"。

按照定义，我们知道一个反射计算系统具有自我表示的能力，这种自我表示使系统能够回答自身的问题并支持作用于自身的动作。一个程序设计语言把反射作为程序设计的一个基本概念，并提供设施以显式地处理反射计算，则称这个语言具有反射结构。

建立一个反射结构，是把某个计算域D_n中隐含的实体和另一个计算域D_{n+1}加以有效的联系，这就有了两个层次的计算域，我们把D_n称为基级，D_{n+1}称为元级。对于更高的层次来说，D_n和D_{n+1}都可以做为基级；对于更低的层次来说，它们都可以做为元级。

大多数已有的反射系统都是在解释的基础上建立的。设域D_n中的对象用语言L_n写成，所使用的解释程序是I_n，它是用语言L_{n+1}写成，对应的解释程序是I_{n+1}。在通常的不具有反射功能的计算系统中，L_n和L_{n+1}是不同的两种语言。在有反射功能的系统中，事实上有一个用同一语言L构成的无穷链，称为反射塔，它的解释程序I，用与L不同的另一语言L′写成。当系统在D_n域工作时需要反射，转移到高一级的D_{n+1}级进行计算，如在D_{n+1}域仍然需要反射，则转移至更高一级D_{n+2}，依此类推，直到找出一个反射函数，它不再引用任何其他反射函数。

Ferber把现有的反射模型分成两大类[6]：(1)结构反射模型：所关心的主要是在实现对象时类和元类的使用。在纯粹类的模型中，每个实体是单个类的实例，类是元类的实例。这一模型可以扩展OOPL的静态部分，即对象的结构方面。(2)计算反射模型：此模型所依据的事实是每个对象有它自己的元对象，后者表示对象的结构和处理消息的方法等原来是隐含的信息。使用元对象在OOPL中建立反射能够定义一些原来系统中所没有的特征，如多重继承，对属性访问的demons，排错工具等等。

定义一个反射结构要解决的主要问题有：(1)对语言中的实体——对象和消息，它们的元级应具有的结构和行为。元级设施要能够把程序当做数据来操纵，可以显式地调用解释程序和编译程序，有创建新的控制结构的能力，能够描述自身，可以控制实体的分配和它们的生命周期。(2)要确定因果关联的实现，这就是在元级怎样描述消息的处理和方法的寻找。(3)什么时候系统将转移至元级，使用元对象和元通信。

一个反射系统的功能，随应用目的和设计要求而有所不同，一般可以做到：(1) 对实现的描述，如实体的分配，废址的收集，增量式扩展，系统的再实现等。(2) 对语言中的实体实施监控，对编码进行观察和分析，以及动态修改。(3) 实现某些程序设计的工具，如排错，接口形成等。(4)自我再组织，系统行为的学习，一致性和效率的增强等。

4、实现方法

(1)使用类实现反射[7],[8] 类是OOPL的基本成份，它描述一组事物的共同特征，整体地代表了这类事物，属于类的事物可以共享类所描述的特征，对外则屏蔽了内部结构和实现细节。可以把类看做是OOPL中的一个对象，它决定其他一组对象的结构和行为，这些对象称为类的实例。OOPL对于类所管理的对象有以下的约束：(1) 特征属性能用可测度的量来表示，如变量的值等。(2)和特征属性相关的动作是可计算的。

在OOPL中，类的构造一般包括以下几个部分：(1)类的名；(2)属性结构：分为专用的类型、变量和操作的定义和公共的类型、变量和操作的定义；(3)和其他类的关系；(4) 必要的其他描述，如自动验证的前置条件和事后条件等。

从反射的观点来看问题，类可以被看做是它的实例的局部描述子，它以变量集定义实例的共同结构，以方法集定义它们的共同行为。

一个类的实例也是类，则称这个类为元类。元类的特性可以被它的类所继承。所以，联系于元类的关键概念是它们可以描述其他的类，包括这些类的结构和行为两个方面。由于元类有描述类的行为的能力，因此，它们可以控制类的实例。

从结构的观点来看，元类可以定义持有嵌入于一个类之中的信息所需的所有实例变量，这种嵌入于类的信息有类的名，实例变

量描述，为消息传递的方法辞典，以及有关继承的信息。元类的这些特点和能力，形成一种程序设计的风格。

Smalltalk语言最早考虑了元类的设置。为了展示每个对象的结构和行为并使之可以访问，就需要使用元对象去控制每个对象，包括对于对象的表示和修正，Smalltalk为此设置了元类。首先是在Smalltalk-76中，用元类来表达类的行为，使之能够处理消息传递。在Smalltalk-80中，每个类有一对应的元类，后者的方法辞典定义了类的规程，例如一个对象的创建是通过发送消息new给它的类来完成，类这时作为接收者，发送给它的消息分配一个新的实例，此实例的变量被初始化为nil。但Smalltalk-80中的元类还很少被用来控制或描述类的结构。

除了类的结构以外，元类还可以描述类的行为。因此当任何消息发送给类的时候，系统将在元类中找到方法。元类这一描述类的行为的能力是多方面的，它能够提供实例生成的规程，对于这一消息的描述通常是以一基本的初始化作用于对象分配的结果之上而组成的。因此，分配必须被元类所描述，使之能控制实例的实现。元类控制类的初始化：要生成一个类C，发送一个消息给它的元类CC，这一消息触发了CC的类所描述的分配方法，接着应用CC所描述的初始化方法。在初始化的时候，类将使用继承信息计算它的实例的结构和行为。此外，元类还可以给实例方法的编译优化以提示。Cointe为Smalltalk-80的元类计算提出了五条公设[7]，并给出了若干例子，可资参考。

(2) 使用元对象实现反射[5],[8] 在这一方法中，每个对象有一个对应的元对象。在对象中含有域的信息，在元对象中含有反射信息，描述对象的基本行为，即对象如何处理消息的方法。对象和元对象之间的通信没有标准的规程。元对象也是一个对象，它们可以有自己的元对象。Maes在KRS语言的基础上，定义了3-KRS语言，以实现计算反射[5]。3-KRS的特点是：(1) 将对象级和反射级分开，每个对象都有一个元对象，后者有指针指向前者。(2) 每个实体都是一个对象，包括实例、类、方法、元对象、消息等等，这样，3-KRS的各个方面都是可反射的。(3) 3-KRS的自我表示是完全的，一致的。(4) 自我表示能在运行时被修改，因而影响运行时的计算。

Ferber把使用元对象实现反射的方法分成两种不同的模型[6]，一种是使用接收者类做为元对象，当接收者或接收者类需要时，转移至元级。另一种是使用一个称为META-OBJECT的特定类的实例做为元对象，当接收者有一相关的元对象时，转移至元级。Ferber提出这两种模型是为了使用以类和实例为基础的OOPL语言以实现元对象反射，而Maes的3-KRS语言不是以类为基础的OOPL。

以类为基础的OOPL用来实现计算反射时，人们比较自然地想到把对象的类做为元对象，这和Cointe的结构反射中所采用的方法是类似的。从行为的角度来看，一个对象接收一个消息，和它的类接收一个处理消息的消息是等效的。Ferber用OBJVLISP语言来实现他所提出的模型。在以类为元对象的模型中，所得的主要结果是：(1) 一个类的每个实例共享相同的消息解释程序。(2) 消息解释的修改可以通过替换元类来达到，不过要十分小心，否则很容易导致系统的不一致。(3)不能在元对象中记录对象的个体特性，如统计信息，输入消息的历史等。这一限制是很严重的，因为元对象的一个最重要的方面是能够存贮它们所指的对象的特定信息。以META-OBJECT类的实例做为元对象的模型，接近3-KRS的反射模型，具有许多优点：(1) 对象的元对象容易修改。(2) 一个对象能被它的元对象所监控，记录所发生的事情，甚至于某些改变的决定。(3) 可以通过META-OBJECT的子类生成来定义新的处理消息的方法。

在[6]中，Ferber还提出第三个计算反

射模型，称为元通信模型。这一模型的基本思想是每一通信都是一个对象，把对消息的解释做为它本身的责任。在以上的三种模型中，Ferber本人认为使用元对象是最好的选择。

5、基于编译的反射

现有的文献介绍反射问题时，绝大部分都是以解释为基础，这一方面有历史的原因，同时也是由于基于编译的反射在实现上要比基于解释的反射更加困难，但并不是不可能的。要实现基于编译的反射，需要在系统的因果关联能力与系统的效率、可靠性、以及可移植性之间作出适当的折衷。我们的研究结果表明[9]、[10]，如果类的层次结构设计加以一些合理的约束，就可以使支持动态联编的OOPL具有基于编译的反射能力，允许用户进行反射程序设计。

我们所使用的语言是C++[11]，根据反射程序设计的要求，提出了一种以C++为基础的反射结构[9]，支持用户以类的层次结构来定义反射层次。一个类c的实例可以共享同一个元对象，条件是：（1）该元对象从属的类mc是c的直接子类（这是就mc直接继承c）；（2）c是由反射结构提供的基本类-top的子类；（3）c的一个或多个成员函数c::m被定义成虚拟形式。满足这些约束条件的程序可以在我们所提供的反射机制支持下实现反射计算。

以类c和mc为例，对应于c::m的元操作可以定义在mc::m之中。这里，因果关联通过成员函数来实现。利用C++的继承性和虚拟函数以及相应的动态联编等特性的支持，我们实现了一种动态导向机制，使得任何对c::m的调用可被隐式地导向至mc::m，以激活相应的元操作（如跟踪、统计调用频度、测试执行时间、设置/制定被反射对象的状态等）。这种导向关系可以显式地动态切换。在非导向状态下，c::m的执行与mc::m脱离关联。这一反射结构还支持多反射层次，即mc的实例也可以有元对象，依此类推。我们用Advantage C++1.1M4在386微机上实现了上述反射结构，详细情况见[9]和[10]。

利用上述的反射结构，在编译的条件下，所取得的主要结果是：（1）对反射机制本身的反射实现。在实现了反射机制的一个基本核以后，用反射方法扩充实现其他的反射机制，从而使反射机制不仅被样本程序，而且被它本身测试与验证。（2）在C++中引入了对象标识ID本身就是对象的概念。ID除了标识对象以外，还提供了强类型下与类型无关的元对象的支持，而且ID本身也可以被反射。（3）利用反射机制和ID，在C++中引入组合对象和对象集合类型两种新概念。

和基于解释的反射相比，基于编译的反射明显地具有运行效率高的优点；虚拟函数的动态导向使得原来非反射程序的操作实现不需修改就可以追加反射；C++的强类型化和基于C语言的特征使系统可靠性和对于已有软件资源的利用等方面均优于基于解释的反射系统。基于编译的反射系统也有不如基于解释的反射系统的一些方面，最主要的是由于本原特性的限制，基于编译的反射很难以语句为单位实现因果关联，自我表示的信息只能在修改编译程序以后才能达到基于解释的反射系统那样的完整程度。

6、结束语

元知识和元计算的概念是人工智能和计算机科学领域中新注入的动力。有了关于计算的计算，我们就可以动态地对计算过程加以控制和修正，大大地增强了计算的能力和适应性。反射机制是实现元计算的一种有效途径，而面向对象的语言是最容易实现反射功能的语言。所以，扩展OOPL使之具有良好的反射结构，是一个重要的研究课题。这是我们的认识和主要的结论。

参考文献

1. F. Hayes-Roth,. D.A. Waterman, and D. B. Lenat (ed), Building Expert Systems,Addison-Wesley Pub. Co.,1983,

Ch. 7, Reasoning About Reasoning, by D, B, Lenat, R. Davis, J. Doyle, M. Genesereth, I. Goldstein, and H. Schrobe

2. L. Aiello, C. Cecchi, and D. Sartini, Repre-Sentation and Use of Metaknowledge, PIEEE Vol. 74, No, 10, Oct, 1986
3. E. Sandewall, Nonmonotonic Inference Rules for Multiple Inheritance with Exceptions, PIEEE Vol. 74, No, 10, Oct. 1986
4. A. Yonezawa and T. Watenabe, An Introduction to Object-Based Reflective Concurrent Computation, Proc. of the ACM SIGPLAN Workshop on Object-Based Concurrent Programming, SAN DIEGO, Sept. 26-27, 1988
5. P. Maes, Concepts and Experiments in Computational Reflection, ACM SIGPLAN Notices, Vol. 22, No, 12, Dec. 1987
6. J. Ferber, Computational Reflection in Class Based Object Oriented Languages, OOPSLA'89 Proceedings, Oct. 1—6, 1989
7. P. Cointe, A Tutorial Introduction to Metaclass Architecture as Provided by Class Oriented Languages, Proc. of the International Conf. on 5th Generation Computer Systems, Nov. 28—Dec. 2, 1988, Tokyo, Vol. 2, 592—603
8. N. Graube, Metaclass Compatibility, OOPSLA '89 Proceedings, Oct. 1—6, 1989
9. 陈平、蔡希尧、金益民, An Approach to Introduce the Reflection to C++, Proc. COMPSAC'90, Oct. 1990
10. 陈平、蔡希尧、金益民, Object Identity is Object, A Reflective Approach in C++, Submitted to Toulouse'90
11. B. Stroustrup, The C++ Programming Language, Addison-Wesley Pub., 1986

(接第19页)

参考文献

[1] J. Y. Girard. Linear Logic. TCS, 50: 1-102, 1987.
[2] J. Y. Girard. Y. Lafont. Linear logic and lazy computation. In, Springer LNCS 250, 1987.52-66.
[3] U. Solitro. A typed calculus based on a fragment of linear logic. TCS, 68: 333-342, 1989
[4] Y. Lafont. The linear abstract machine. TCS, 59: 157-180, 1988
[5] J. Y. Girard, et al. Proofs and types. Cambrige Tracts in Theoretical Computer Science, Cambridge Universiry Press, 1989
[6] J. Y. Girard. The System F of variable types, fifteen years later. TCS, 45: 159-192, 1986
[7] R. A. G. Sleey. Linear logic, *-autonomous categories and cofree coalgebras. Contemporary Mathematics, Vol. 92, 1989. 371-382
[8] J. Y. Girard. Toward a geometry of interaction. Contemporary Math Vol. 92, 1989
[9] V. C. V. de Paviva. A Dialectica-like model of linear logic. In LNCS 389. 1989.341—356.
[10] J. C. Guzman, et al. Single-threaded polymorphic λ-calculus. In Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia, June 1990.
[11] S. Cerrito. A linear semamtics for allowed logic programs. In Proc. 5-th IEEE Symn. on Logic in Computer Science, Philadelphia, June 1990.
[12] N. Marti-Oliet et al. From Petri nets to linear logic. IN LNCS 389, 1989. 313-340
[13] Y. Lafont. Interaction nets. In Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco, 95-108, Jan. 1990.
[14] J. Y. Girard, Linear logic and parallelism, in: Proc. School on Semantics of Parallelism, IAC (CNR, Roma, 1986)