开放实现与计算反射技术综述*)

A Survey of Open Implementation and Computational Reflection Technology

唐亚哲　李增智　张　鹏

(西安交通大学电信学院系统结构与网络研究所　西安710049)

Abstract Open implementation and computational reflection have emerged as flexible programming techniques and structured ways of achieving program adaptabilility and reusibility. In this paper, we introduce the basic concepts about them , present a survey of related works and give some deep analyzes.

Keywords Open implementation, Computational reflection, Metaobject protocol

1. 引言

传统上软件开发遵循这样的原则:模块应提供表达其功能的接口,同时应隐藏其实现细节。这个原则被非正式地称为"黑盒抽象",是软件设计的一个基本方法学原则,也是其他研究方向比如可移植性、重用、组件软件、软件工程方法、标准式设计等的基石。

但是近年来随着软件重用技术和软件可扩展性技术的发展,人们对这样一个软件基本准则提出了疑问。主要的疑问是:模块的最佳实现策略在不知道模块的具体使用环境的情况下是不能确定的。也就是说,用户(模块的使用者)而不是模块的设计者通常知道实施方案的最佳策略。但在黑盒抽象原则下,设计者过早地确定了实现策略,并且在黑盒中固化,用户不能改变这些策略,因而往往不能满足用户的需求。

让我们看一个典型的例子,假设有一个模块实现了对集合的封装,提供给用户的基本接口有四个:构造集合(MakeSet);插入集合(Insert);是否在集合中(IsIn);从集合中删除(Delete)。用户使用这个模块时,先构造一个集合,然后可以进行插入、判断和删除操作。模块设计者已经将集合的存储等实现细节固化,用户不能修改。但是用户在使用集合时,有不同的应用情况,比如集合的大小不同,不同的操作的频率,有的用户主要的操作是判断是否在集合中;有的用户的主要操作是插入。对于这些不同的应用情况,仅仅用一种固化的集合实现方式肯定不能满足所有的用户需求。典型的问题是:确定的一种集合实现方式可能适合于有100个元素的集合却根本不适合于有10000个元素的集合,问题主要出在性能上。施乐公司帕洛阿尔托研究中心(Xerox PARC)的 Gregor Kiczales 等人在研究了这个例子之后,给出了他们的解决方案[1]。主要的思想是模块将一些实现细节向用户开放,即用户可以控制模块的一些实现策略。比如在上述例子中,改进的模块接口有五个:构造集合1(MakeSet);构造集合2(MakeSet(usage));插入集合(Insert);是否在集合中(IsIn);从集合中删除(Delete)。其中,构造集合1与前面的相同,而构造集合2使用了一个字符串参数,用户可以使用这个参数把自己的使用模式告诉集合的实现,Gregor Kiczales 给出了一个例子(图1)。图1中,左边是集合实现提供的接口(功能),右边是用户的使用实例。可以看出,集合封装的实现方式可以根据集合元素个数以及插入、删除和判断是否为集合元素等操作的频率进行自适应,目的是在任何一种情况下给用户提供较好的运行性能。

```
makeSet(usage)
makeSet()
insert(item, set)
delete(item, set)
isIn(item, set)
```

```
makeSet("n=10000,
insert=lo,delete=lo,isIn=hi")
makeSet("n=5,
insert=hi,delete=hi")
```

图1　开放集合实现及应用

上述例子集中体现了开放实现(Open Implementation)的思想。开放实现对于软件模块的重用、软件的自适应以及软件系统的灵活性、可扩展性都具有重要的意义。本文对开放实现技术做一简单介绍。

2. 开放实现与计算反射基本概念

2.1　开放实现

开放实现就是软件模块有能力自适应或改变它们的内部实现以满足不同的用户需求,从而提高软件模块的可重用性和灵活性[1]。开放实现是一个比较高层次的概念,所有对模块功能实现细节的开放都可以称为开放实现。在理解开放实现时,有三个基本概念:(1)映射选择(Mapping Dilemmas);(2)映射决策(Mapping Decision);(3)映射冲突(Mapping Conflict)。在老的黑盒抽象原则下,模块设计者将除接口之外的所有实现细节都隐藏,现在看来并不是非常可取。模块在向用户提供一定的功能(接口)时,应该考虑是否开放一些至关重要的影响模块性能的因素。这些因素就称为映射选择因素。也就是说,对这些因素,模块设计者必须做出一个选择,即从若干个映射高层次功能到低层次功能的实现方式中的一个选择。我们把模块设计者采用的解决映射选择的决策称为映射决策;在这个决策中出现的冲突称为映射冲突。从前面例子可以看出,模块的实现必须部分地开放,允许用户控制一些实现策略。

开放实现技术的几个典型应用实例:

1)虚拟存储—基本功能很简单:一块存储器地址,可以读写。映射选择包括怎样映射虚拟地址到页面,怎样映射页面地址到物理地址。典型的映射冲突发生在数据库系统,假定数据

*)国家863项目资助(863-511-946-008);陕西省自然科学基金资助(99X18)。唐亚哲　讲师,博士研究生,主要研究方向为网络管理和软件进化。李增智　教授,博士生导师,主要研究方向为网络管理。张　鹏　博士研究生,主要研究方向为分布式对象和软件 Agent。

库系统采用的虚存实现策略是 LRU(Least Recently Used,最近最少访问)算法。在数据库管理系统完成内存一部分的一次遍历,同时完成对另一部分的随机访问的情况下,系统的性能将会很差。

2)并行编程语言—基本功能是提供自然的方式表达并行计算的一个操作。三个映射选择是:怎样分布操作到若干个物理 CPU 上,怎样分布数据,怎样进行同步。通常的映射冲突出自矩阵的布局上。例如编译器将矩阵 A 和 B 的第 I 行分布到同一个 CPU 上时,矩阵加法将不需要通信开销,但矩阵乘法将需要大量的通信。

3)通信—基本功能是数据流。映射选择包括消息缓冲、重发策略、同步和顺序限制。映射冲突发生是因为不同的应用需要不同的 QoS。例如:发送工资表的应用程序需要所有的包正确到达,甚至可以容忍大的延迟。而发送 MPEG 图像数据的应用,少量的丢包并不十分重要,重要的是低延迟。

在很多情况下,开放实现看来是必需的。问题在于如何设计开放的模块,模块使用者(用户)如何使用开放的模块优化性能?如何使模块的开放更加有组织、有规则,而不是胡乱开放?即怎样以一种原则性的方式给用户提供选择决策的能力,同时不引起新的更大的问题。Gregor Kiczales 等人认为[1],黑盒抽象的优点是它提供给程序员一个简单、容易理解的接口,并且把实现细节隐藏在接口之后。开放实现并不是对黑盒抽象的彻底否定,而是在尽可能保持黑盒抽象优点的基础上,在性能、可重用性等方面的一个折衷。

2.2 计算反射技术

为了更好地利用和控制开放实现,Gregor Kiczales 提出了反射模块(Reflective Module)的概念[1],主要思想是:(1)既然必须开放,映射选择不可避免,我们至少要将用户对这些问题(映射选择的因素)的控制同用户对基本功能的控制区分开;(2)借鉴计算反射技术方面的一些进展来完成这样的分开。应用这种技术,系统可以提供一个元接口允许用户检查和修改基本用户接口。

我们首先来回顾一下计算反射技术。计算反射技术起源于人工智能,是 MIT 的 Brian Smith 首先提出来的[2]。主要是描述一个自我感知(Self-Awareness)的计算过程。基本原理是系统只有具有自我感知、自我控制的能力,才能不断适应变化的外部环境。一个反射程序有能力考察自己的运行过程,并且改变它。该技术后来逐渐应用到程序设计语言、并行系统、操作系统、网络和分布式系统以及软件工程方面。

计算反射技术的一个基本思想是 Meta-Level 和 Base-Level 的定义和区分。Base-Level 是处理应用领域问题的级别(也就是我们使用通常程序设计语言设计程序的级别);Meta-Level 是处理 Base-Level 处理如何进行的级别。相关的概念还有:检查(Introspection)和干预(Intercession)。检查指查询或检查自身状态和结构的过程(正如 Java 语言所提供的);而干预指干预特定语言结构解释或运行行为的过程,根据反射技术应用的时间不同,还可以分为运行时反射和编译时反射。

反射技术常用的实现方式是 OO 的,称为元对象协议 Metaobject Protocol[3]。意思是把 Self-representation 的实现用 OO 框架来进行,例如:Java 语言中的 java.lang.reflect.* 中的类是对 Java 程序中类和对象以及方法等语言元素的描述类。

将反射技术应用到开放实现上,就产生了反射模块的概念。反射模块不仅提供基本的功能,并且提供再协商以确定如何提供基本功能的功能。我们把这两种功能分别称为 Base 和 Meta 功能。相应的有 Base 接口和 Meta 接口。Gregor Kiczales[1]建议对映射决策的访问只是通过 Meta 接口来进行。

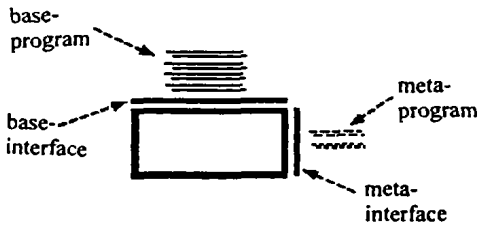

图2 开放模块接口

图2是文[1]中的图示。反射模块提供了两个接口:一个基本接口(Base Interface)和一个元接口(Meta Interface)。在黑盒抽象模式下,用户只需编写应用程序调用模块基本接口,完成相应的功能;在开放实现模式下,用户不仅要编写基本应用程序,调用基本接口完成功能,而且要调用模块的元接口,监控模块实现基本功能的方式。这样的实现方式,不仅体现了开放实现的要求,也体现了将实现细节的控制与功能控制相分离的要求。使得我们在引入开放实现的情况下,没有增加太多的复杂性,并且最大限度地保留了黑盒抽象的优点。

3. 相关工作及进一步研究方向

开放实现技术在程序设计语言、OS 和分布式系统等方面得到了较广泛的应用。这些系统的共性是:有要求开放的需求,系统需要灵活性、扩展性和广泛的重用。

3.1 程序设计语言

传统的程序设计语言基本是封闭的,程序员按照程序设计语言提供的接口,操纵一系列内置的数据结构以及通过一系列内置的控制结构指定控制逻辑来完成编程。

程序设计语言中最新的功能强大的开放实现手段是 MOP(MetaObject Protocol)。MOP 使用 OO 编程概念,提供范围控制(scopecontrol)以及更好的渐进性(incrementality)。典型系统有:CommonLoops[5]、CLOS[6]、Open C++[7]、ABCL/R[8]等。主要思想是:源程序映射(编译、链接、运行)到低层,即编译和运行支持的每一个方面都可以由一些对象或对象组控制,这些对象(组)有定义完好的协议。这些对象被称为元对象(Metaobject),因为它们是关于程序映射的对象,而不是关于程序基本问题域的对象。用户程序员可以用特殊的对象替换一个或多个 Metaobject 来影响程序特定部分实现。

未来进一步的研究方向有:研究设计更好的开放实现编程语言,使用户在可控的条件下访问功能强大的元接口(Meta Interface)。即既要提供足够的开放,又要保证语言和程序的完整性和效率。

3.2 OS

操作系统的开放需求由来已久,操作系统的设计者们已经在开放实现方面有了很多实践经验。最初的操作系统可扩展性设计主要是钩子(Hook)的使用[9],但是钩子实施困难,并且应用范围很窄。因此,后来的研究逐渐转移到了开放实现技术。操作系统主要涉及三方面内容:计算、存储、输入/输出,随着并行环境的逐渐增多,操作系统中的映射选择主要是来自于资源分配,因而操作系统开放实现的重点在资源分配和性能优化上。

典型的开放操作系统有:Apertos[11]系统等。最新的研究

方向有开发支持元接口的操作系统服务，用户可以通过这些接口定制这些服务；开发可扩展操作系统内核，支持元接口，用户可以使用这些接口更有效和更安全地进行资源管理。

3.3 网络和通信系统

传统网络系统出于性能的考虑，实现方式是黑盒的，但是分布式系统的快速发展和网络高速互连能力逐渐显出了传统黑盒方法的不足。不断增长的物理平台、数据访问模式以及QoS需求意味着在许多情况下，应用程序只有自适应它的底层通讯机制与它的特定需求才有可能得到可接受的性能。因此，基于开放实现的网络与通信系统已经提上了日程。

这类系统典型的应用是进程间通信。进程通信的基本功能是把一些字节流从一台机器传送到另一台机器上。映射选择有：数据格式、通信协议的选择、同步/异步机制、消息缓冲和重试机制、同步和保序约束、拥塞控制优先级和调度策略等。

这就要求协议栈要自适应应用的需求，以获取更好的性能和效率。应用帧技术 ALF[12]（Application Level Framing）允许应用指定网上传输的帧（单元）的大小，因此就可以直接处理这些帧，而不是等待协议栈重新装配成应用数据单元。要应用 ALF 技术，应用程序需要对传输协议做更多的选择以及对相关同步和重试策略进行控制。这正是开放实现的思想。另外一个思想是将协议栈的某些功能从操作系统内核级提升到用户地址空间[13]，以提供更多的安全性、范围控制和开放性。

4. 技术分析

4.1 开放实现的本质

从前面的论述可以看到，开放实现的本意非常简单，就是把实现细节向用户开放，使用户参与到低层的实现中，来优化和扩展低层的实现。集合例子集中体现了开放实现的这个根本目的。我们认为，开放实现对于软件开发以及其他领域更为本质的思想是它的实现手段—分而治之。将功能的实现和对功能实现的控制分开考虑和处理，不仅达到了实现细节开放的目的，而且使这样一个细节开放的过程遵循一种可控的清晰的实现模式。因此才可以说，开放实现建立在黑盒抽象上，在保留黑盒（Base Level）的基础上增加了白盒（Meta Level）。

4.2 开放实现与反射的关系

开放实现与反射技术天生就有联系，反射技术的本质是自我感知，开放实现的本质是实现细节开放，反射技术显然是开放实现的理想方式。系统将自我感知的一些方面呈现出来，由使用者去监控就是开放实现。从某种程度上讲，开放实现分而治之的本质与反射技术有不可分割的联系。从概念上讲，开放实现更普遍一些，反射技术更技术一些，例如：反射式语言其实就是开放的语言系统，反射式操作系统就是开放的操作系统。

4.3 开放实现的表现形式

可以看出，开放实现最主要的表现形式是性能优化和扩充。性能优化是开放实现最初的动力和目的。用户通过元接口对功能的实现细节进行控制，例如选择最合适的算法、最合适的调度策略或者最合适的存储方式等，通过用户将运行环境（包括数据模式，使用模式等）和程序实现有机地结合，来达到性能的最大优化。

近来，开放实现的扩充功能也得到重视。当系统允许用户提供自己的实现细节时，就是扩充。研究内容有功能的增、删、改。即把程序也当作数据，可以进行类似于数据库一样的增删改基本操作。

总结 当前，主流面向对象程序设计语言 Java 也引入了反射机制，用户可以利用这些机制对 Java 解释器进行适当的定制，完成其他语言很难完成的功能[14]。这说明开放实现与反射技术已经融入到程序设计语言、软件开发的主流技术中。

本文介绍了开放实现和反射技术的基本概念和研究现状，分析了这两种技术对软件开发方法的影响。可以看出，开放实现与反射技术是对传统黑盒抽象的合理扩充，对未来软件的开发将会发挥非常重要的作用。

参考文献

1 Kiczales G, et al. Foil For The Workshop On Open Implementation. Available at: http://www.parc.xerox.com/spl/projects/oi/workshop-94/foil/main.html

2 Smith B C. Reflection and semantics in a procedural language: [Ph. D. thesis, Technical Report TR-272]. Laboratory for Computer Science, MIT, 1982

3 Kiczales G, et al. Metaobject protocols: Why we want them and what else they can do. Object-Oriented Programming: The CLOS Perspective, The MIT Press, Cambridge, MA, 1993. 101～118

4 Maeda C, Lee A, et al. Open Implementation Analysis and Design. Available at: http://www.parc.xerox.com/csl/groups/sda/projects/oi-at-parc/ourpapers/oiad.pdf.

5 Bobrow D, et al. Commonloops: Merging lisp and object-oriented programming. In: Proc. of the Conf. on Object-Oriented Programming: Systems, Languages, and Applications, 1986

6 Bobrow D G, et al. Common lisp object system specification. Sigplan Notices, 1988, 23(Special Issue)

7 Chiba S, Masuda T. Designing an extensible distributed language with a meta-level architecture. In: Proc. of the 7th European Conf. on Object-Oriented Programming, LNCS 707, 1993. 482～501

8 Watanabe T, Yonezawa A. Reflection in an object-oriented concurrent language. In: Proc. of the ACM Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), San Diego CA., ACM, Sep. 1988. 306～315

9 Bershad B N, et al. SPIN -- An Extensible Microkernel for Application-specific Operating System Services. In: 6th European SIGOPS Workshop, 1994

10 Patterson R H, Gibson G A, Satyanarayanan M. A Status Report on Research in Transparent Informed Prefetching. Operating Systems Review, 1993, 27(2): 21～34

11 Yokote Y. The Apertos Reflective Operating System: The Concept and Its Implementation. In Object-oriented Programming Systems, Languages, and Applications '92, ACM SIGPLAN Notices, ACM Press, 1992, 28(Oct): 414～434

12 Clark D D, Tennenhouse D L. Architectural considerations for a new generation of protocols. In: Proc. of the ACM Symposium on Communications Architectures and Protocols, Sept. 1990. 200～208

13 Thekkath C A, et al. Implementing network protocols at user level. IEEE/ACM Transactions on Networking, 1993, 1(5)

14 Sun Corp. Java Core Reflection. Available at: http://java.sun.com/products/jdk-/1.1/docs/guide/reflection/