面向对象程序的类信息的抽取规则①

黄 捷，古 辉

(浙江工业大学 计算机科学与技术学院，杭州 310023)

摘 要：通过计算机自动实现对程序的理解是目前国内外研究的热点内容。以面向对象程序为研究对象，提出了一种从面向对象程序中抽取类信息的规则，利用该规则可以将程序中的类以及其它程序信息抽取出来，并以 UML 类图形式表示出来，进而为计算机自动实现对程序的理解奠定基础。在本文中描述了产生 UML 类图中的各类规则，包括了类、属性、操作、对象、关系、继承、关联和接口等。最后介绍了该规则的一个应用实例。

关键词：程序理解；逆向工程；信息抽取；UML 表示方法

Object-Oriented Programming Classes of Information Extraction Rules

HUANG Jie, GU Hui

(College of Computer Science & Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract: In domestic and foreign research, using the computer to understand the program automatically is the focus. This paper, object-oriented program for the research, proposes object-oriented programming classes of information extraction rules. This method can extract the type of information from the program, then, represents these information by class diagrams, thus lays the bais for the computer to understand the program automatically. In this paper, we specify the rules for generating UML class diagrams, which include classes, attributes, operations, objects, relationships, inheritance, associations interfaces and so on. Finally, we introduce an application of the algorithm instance.

Keywords: program comprehension; information extract; reverse engineering; UML representation

1 引言

在软件维护及再工程中，需要对遗留软件系统实施逆向工程，软件逆向工程的基本原理是抽取软件系统的主要部分而隐藏细节，然后使用抽取出的实体在高层上描述软件系统。理解面向对象系统的关键就是理解其类图的结构[1]，并从类图中挖掘出系统的关键设计特征[2]。而 UML(Unified Modeling Language)统一建模语言,是用来对软件系统进行可视化建模的一种语言，可用于对面向对象软件产品进行说明、可视化、和编制文档。如果对程序进行逆向工程后得到的是 UML 类图的形式，将有利于程序的理解。本文提出了一个抽取规则，以面向对象程序代码为输入，对面向对象程序进行逆向获得其类信息，并以 UML 类图形式表示出来。

2 类信息抽取规则

类信息抽取主要有包含九个步骤。

(1) 识别结构事物:结构事物通常是指模型的静态部分，描述概念或物理元素。这些主要分为类，抽象类，模板类和接口。为了区分结构事物，定义规则 1～规则 4：

规则 1：识别代码中的关键字 class，并以 UML 的图表示，如图 1 所示。

① 基金项目:浙江省基金(Y1080189)
收稿时间:2010-08-23;收到修改稿时间:2010-09-21

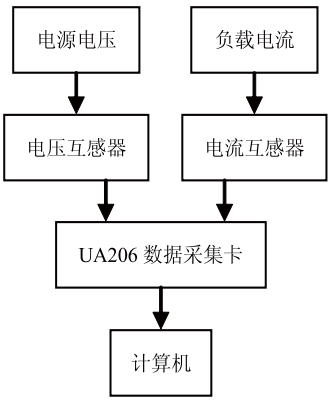

图 1　类在 UML 中的表示

表 1 中定义了属性和行为的英文简单表示符号。

表 1　属性和行为的英文简单表示符号

关键字	简写符号表示
Private	-pvt_var_name
Public	+pub_var_name
Protected	#prot_var_name
Derived	/der_var_name
Abstract	abst_var_name
Static	$stat_var_name
Package	~package_name

规则 2：抽象函数能用以下规则进行识别：(1)类名前有关键字 abstract 或者；(2)这个类没有实例或者；(3)它至少包含一个纯虚函数。规则 2 规定了抽象类的映射规则。为了保证程序的结构性和简单性，在面向对象程序中接口主要通过抽象类来实现，抽象类和方法透露了设计者的意图。

规则 3：模板类可以通过下列规则经行映射(1)可以通过代码中的关键字 template；(2)它是一个类模板从指定的代码实例化的结果。在面向对象程序代码中，类模板，是对一批仅仅成员数据类型不同的类的抽象。因此，模板是用于实现高效的代码基础，是可重复使用和扩展的。本文提出的规则 3 指定了模板类到 UML 类图的映射规则。

规则 4： 在代码中识别关键字 struct。它是一个非面向对象的构造，但是在 C++中还是大量的用到。由于结构不属于面向对象程序设计，一般不会有人提出相应的映射规则，因此这块是有所缺漏的。当程序设计者想声名一个主要是由 public 类型成员组成的类时，用结构代替类的声名，将方便理解。因此本文中提出了规则 4 对结构类型进行映射。

(2) 识别行为事物:行为事物是 UML 模型的动态部分，描述了跨越时间和空间的行为。主要是指交互关系，交互关系是由一组对象之间在特定上下文中，为达到特定的目的而进行的一系列消息交换而组成的动作。为了识别行为性事物，定义规则 5：

规则 5：交互中组成动作的对象的每个操作都要详细列出，包括消息、动作次序(消息产生的动作)，连接(对象之间的连接)。在 UML 中消息画成带箭头的直线，通常加上操作的名字，如图 2 所示：

图 2　消息

由于行为性事物属于 UML 模型的动态部分，所以这里就不详细讨论这些问题，本文中提到的只是其在 UML 类图中的表示。

(3) 识别分组事物：分组事物是 UML 模型中组织的部分，可以把它们看成是个盒子，模型可以在其中被分解。总共只有一种分组事物，称为包(package)。包是一种将有组织的元素分组的机制。结构事物、动作事物甚至其他的分组事物都有可能放在一个包中。与组件(存在于运行时)不同的是包纯粹是一种概念上的东西，只存在于开发阶段。为了识别包，定义规则 6：

规则 6：通过代码中的关键字 package 或者.h 的头文件来识别。在面向对象代码中我们使用包，头文件或者命名空间来将元素成组，它们简化了面向对象代码的维护和重用。包体现了功能的聚合服务，命名空间允许面向对象程序的可重用，头文件用来连接一个程序，这样使得包，头文件还有命名空间成为逆向工程的基本元素，因此，规则 6 指定了包，头文件还有命名空间到 UML 类图的映射规则。

(4) 识别注释事物：UML 模块中解释部分称之为注释事物。映射注释采用规则 7：

规则 7：识别代码中的注释部分，例如//.....或者/*....*/注释是用来说明和评价模块中的任意元素，使用注释能更好的理解程序并具有灵活性，因此为了更好的了解程序设计，逆向工程的注释必须在设计文档中表示出来，本文的规则 7 规定了相应的注释映射规则。

(5) 识别对象:为了从面向对象程序代码中识别出

对象，定义规则 8：

规则 8：在代码中找出那些将一个类名作为其数据类型的类的实例。跟类的表示方法一样，不同的是实例名下面要加下划线。同时还能识别代码中所有独立的函数，该函数在 UML 中用虚线矩形表示。根据规则 8 我们将图 3 左边的代码表示成右边类图的形式。

在任何的面向对象程序中都是通过对象来处理过程的，所以对象是代码的主要部分。对象为编程提供了灵活性并且为面向对象编程环境提供了更高层次的抽象。因此，逆向工程中的对象在程序理解中起到了重要的作用。本文中的规则 8 就指定了对象的映射方法。

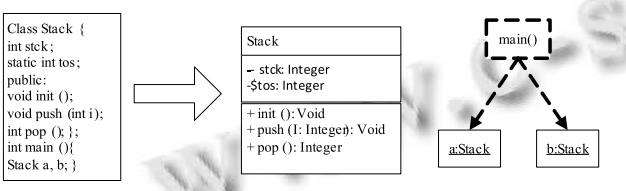

图 3 类中对象的抽取

(6) 识别依赖关系：依赖是一种相互制约而又相互依赖的关系，即一个类的改变可能会影响到另一个正在使用该类的类的关系。由于依赖关系不能很明显的在代码中体现出来，所以过去的研究者觉得没有太大的必要去讨论这个问题。但是依赖关系在程序理解中确实起着重要的作用，反映一个类的改变如何影响到另一个类，因此定义了规则 9。规则 9：针对类的依赖关系，我们可以通过一个类中包含了另一个类，当被包含的这个类发生变化的时候，这个类也跟着相应变化这种关系来识别，在 UML 中用带虚线的箭头来表示依赖关系。根据图 4 左边框中的代码，我们有两个类 Filmclip 和 Channel。根据规则一可以将它们表示出来，同时 Filmclip 类中的一个公共函数 playon 使用了一个 Channel 类的实例 C 作参数，因此根据规则 9 我们将图 4 左边的代码表示成右边类图的形式。

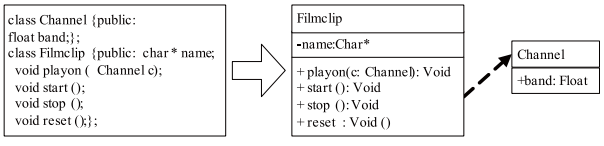

图 4 类中依赖关系的抽取

由于依赖关系不能很明显的在代码中体现出来，所以过去的研究者觉得没有太大的必要去讨论这个问题。但是依赖关系在程序理解中起到了重要的作用，因为它能正确的显示一个类的改变如何影响到另一个类。在逆向工程中，必须纳入依赖关系以防失去任何包含重要信息但是抽象层次低的，因此我们指定了规则 9 的映射关系。

(7) 识别关联关系：关联是一种结构关系，表现为一个对象能够获得另一个对象的实例引用并调用它的服务。这些关联关系由规则 10～规则 12 确定。

规则 10：关联可以这样被识别，即当有两个类 A 和 B，A 中含有 B 的一个实例，B 中也含有 A 的一个实例，在 UML 中我们将用一条实线连接这两个类来表示这种关系。

关联还支持一些修饰关系，如：名字、角色名、多重性。但是这些不能直接从代码中获得，因此这些完全依赖与设计者的角度，所以设计者可以鉴于那些语义类来适当的选择这些修饰关系。关联关系是作为衡量系统文件之间亲近关系的一种度量，同样也可以用它来寻找数据的共享，因此在逆向工程中包含映射关联关系的算法是非常重要的。

规则 11：发现聚合：聚合是关联的一种形式。通常在定义一个整体类后，再去分析这个整体类的组成结构，从而找出一些组成类，该整体类和组成类之间就形成了聚合关系。例如有两个独立的类 A 和 B，而 B 类中包含了这个 A 类，于是我们就称 B 类是 A 类聚合。

在 Kollman 的文献[3]也提到了一种类似的方法，但其只是针对 JAVA 语言的逆向工程而言。采用聚合是因为聚合类的实例可以被其它类所共享，当这些类被删除的时候，聚合类的实例将不被删除，因此聚合关系在很多有非常重要的应用，应尽可能的表现出来。聚合可以帮助程序应用者去更好的了解系统。

规则 12：发现组合：组合也表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期。一旦整体对象不存在，部分对象也将不存在，部分对象和整体对象之间具有共生死的关系。例如有两个类，A 和 B，B 类控制着 A 类的生存期并且 B 类中含有 A 类的实例，于是就称 B 是 A 的组合。聚合和组合的区别在于：聚合关系是“has-a”；聚合关系表

示整体与部分的关系比较弱，而组合比较强;聚合关系中代表部分事物的对象与代表聚合事物的对象的生存期无关，一旦删除了聚合对象不一定就删除了代表部分事物的对象。组合中一旦删除了组合对象，同时也就删除了代表部分事物的对象。

(8) 识别泛化：泛化是一类 AKO 的关系，是类元的一般描述和具体描述之间的关系，具体描述建立在一般描述的基础之上，并对其进行了扩展。具体描述与一般描述完全一致所有特性、成员和关系，并且包含补充的信息。泛化由规则 13 确定。

规则 13：发现泛化和继承：通过识别是否代码中在进行类的声名时包含了其他的类名来发现泛化和继承。首先定义派生类的类名，然后使用‘:’符号，最后是父类或基类的类名。在面向对象程序设计的时候，基类是首先被声名的，然后再是定义派生类。

其实软件开发的一个主要过程就是组件泛化的过程，我们在编程的时候喜欢用继承，是因为它可以在各种应用中被重用，当这些代码已经经过测试，并记录在文档里，可以节省用户大量的时间和精力。继承根据父类的个数可以分为简单继承或者多重继承。如果一个类只有一个父类，这就是简单的继承关系；如果存在多于一个父类的情况，那这个类就采用了多重继承的方式。

(9) 识别实现：实现关系将一种模型元素（通常是一个类）与另一种模型元素（通常是一个接口）连接起来，其中接口只是行为的说明而不是结构或者实现。在 UML 中，实现关系存在于两个类之间，其中一个类必须实现或者应用，它的行为被另外一个类指定。那个指定行为的类称为供应商，应用的类称为客户。实现关系可以包含接口和类，接口指定了行为，子系统应用了这种行为。实现关系由规则 14 确定。

规则 14：发现实现或者接口：实现可以通过下列规则来识别：代码中是否存在关键字 interface 或者它是否是代码中任意被定义为 public 类型且是纯虚函数的类，如果是在这种情况下该类被其它类所衍生，那这个基类肯定就是接口。

UML 通过在类名上加<<interface>>或者在类的边上附加上一个圆来表示实现关系。

使用接口的主要目的就是可重用性，它在重复使用已有代码的同时使事情简单化。例如，当应用只使用一些类的某些函数，定义接口是非常合理的解决方式。

3 抽取规则应用

在本节中将运用本文的类信息抽取规则来构建类图。

图 5 是一段 C++代码，这段 C++代码包含了 6 个类，一个命名空间和一个 main 函数，将作为抽取过程的输入。通过该抽取过程我们可以把代码中的所有类(Company, Bank, Bankaccount, person, Customer, passwd)识别出来并根据规则 1 将其以 UML 类图的形式表示出来。然后根据规则 6 将命名空间 Security 表示出来。接下来识别在 main 函数中的两个对象 mycust 和 b，同时根据规则 8 将其表示出来，在 Bankaccount 类中构造函数 Bankaccount（）使用了 Customer 类型的参数，而 Customer 中也有 Bankaccoun 的实例，因此根据规则 10 可以判断这是一种关联，Customer 类在其定义语句的后面有 Person 类，Bank 类在其定义语句后也出现了 Company 类，因此根据规则 13，这是一种继承关系。经过所有规则的识别，我们将最后结果通过画图模块生成了 UML 类图，如图 6 所示。

```
namespace Security{
class Passwd {
    private:  char* pwd;
    public:   Passwd ();
              Passwd (char * p);
        void setPasswd (char * p);};}
class Person {
    protected: char * nm;
    public: Person (char * nm);
            Person ();
            char * getName ();
            virtual void print ()=0; };
        class Bankaccount
        void check ();
class Customer: public Person {
        private:  static int len;
              Bankaccount * b;
        public: friend void check();
        Customer (char * nm);
        static int getpasswdlen ();
        void print ();
        void checkPasswd ();  };
class Company {
        protected  float prft;
        public: Company (float pr);
        Company ();
                Float getProfit (); };
class Bankaccount {
        private: float tot;
                 Customer hld;
        public: Bankaccount(Customer c);
                 void deposit (float amt);
                 float getTotal ();  };
class Bank: public Company {
                 private: Bankaccount * actlst;
                 public:  Bank (float prft);
        void addAccount(const Bankaccount&b);  };
int main (){
                 Customer mycust ("xyz");
                 Bank b (1000);
                 b.addAccount(Bankaccount(mycust));
                 Return 0;  }
```

图 5　一段 C++代码

利用该类信息抽取规则进行类信息抽取的结果表明，该规则能够将面向对象程序的结构映射成 UML 类图的形式，进而为实现程序结构的可视化表示奠定基础。由于在大型的面向对象的程序系统中，类的设计、类与类之间的关系十分复杂，即使得到了整个系统的类图还是难于理解，运用本文的方法，可以将映射过程中产生的程序信息按一定的

（下转第 149 页）

IDC 报告[9]称：智能视频分析系统在中国的市场普及率还未达到 5%，随着安防发展的不断加快，人们的安全防范意识不断增强，将会对智能视频分析提出更高的要求，正因为这些，智能视频分析产业将会有很好的发展前景；未来的发展重点在于建立智能视频分析监控产品的标准、加强产品的稳定性和准确性[10]；优化出更有针对性的压缩算法和有效的存储检索机制；尤其是对基于目标的自适应规则的描述表示以及增强智能分析的规则,实现多种规则的组合应用方面还需要进一步深入细致的研究。

本文在总结目前国内视频分析应用领域的现状后，结合现有的大型地铁项目，力求基于以上各种因素，提出了将嵌入式深度集成技术以及目前全新的 H.264 编解码技术应用在的视频分析系统中，很好地解决了目前普遍采用的视频分析系统架构存在的问题，在结构和技术上对目前的智能视频分析系统进行了优化，分析给出了较好的解决方案，对未来的智能视频分析系统的发展有一定的借鉴价值。

1 中国投资咨询网.2009-2012 年中国安防行业投资分析及前景预测报告.中投顾问,2009.3:16−43.
2 GB50157—2003 地铁设计规范.北京:中国计划出版社,2003.
3 深圳市地铁有限公司.深圳地铁一号线续建工程综合安防系统技术要求,2007.4.224−380.
4 深圳市赛为智能股份有限公司.深圳地铁一号线续建工程综合安防系统技术需求分析,2007.224−458.
5 International Telecommunication Union, ITU-T H.264 standardized documentation, 2005.3.
6 毕厚杰.新一代视频压缩编码标准——H.264/AVC.北京:人民邮电出版社,2005.21—67.
7 魏芳,李学明.H.264 中整数余弦变换和周期量化的原理与分析.计算机应用研究,2004:26−28.
8 李洛,张剑.基于整数变换的 H.264 标准量化过程.计算机应用研究,2006,(5):31−33.
9 Gantz JF, Reinsel D, Chute C, et al. The expanding digital universe: A forecast of worldwide information growth through 2010. An Internet Data Center (IDC) White Paper, sponsored by EMC, 2010.
10 中投顾问.2010-2015 年中国安防行业投资分析及前景预测报告,2010.31−165.

（上接第 54 页）

规则存储在数据库[4]中，根据需要关联和组合产生更具针对性的局部程序信息，大大提高程序理解的效率。

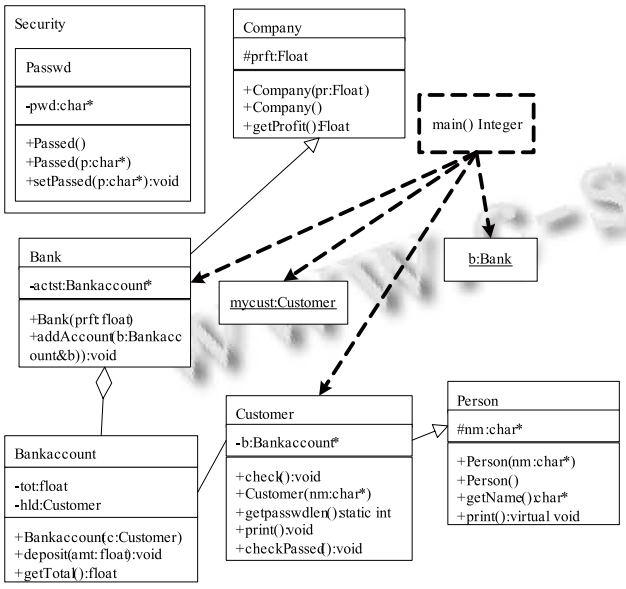

图 6 C++代码到 UML 类图的映射

参考文献

1 Canforaharman G, Penta DPM. New Frontiers of Reverse Engineering. International Conference on Software Engineering, 2007:326−341.
2 Sutton A, Maletc JI. Recovering UML class models from C++: A detailed explanation. Information and Software Technology, 2007,(49):212−229.
3 Kollman R, Gogolla M. Application of UML Associations and their Adornments in Design Recovery. Stuttgart, Germany: Proc. of Eight Working Conference on Reverse Engineering (WCRE'01), 2001: 81−92.
4 Grant SPE, Chennamaneni R, Reza PH. Towards Analyzing UML Class Diagram Models to Object-Relational Database Systems Transformations. Innsbruck, Austria: Proc. of the 24th IASTED International Conference on Database and Applications, 2006: 129−134.