应用对象过程图扩展 UML 建模环境*

于明钊[1], 刘文印[2], 张宏江[2], 李元香[1]

[1](武汉大学 软件工程国家重点实验室,湖北 武汉 430072);
[2](微软亚洲研究院,北京 100080)
E-mail: mingzhao_yu@263.net
http://research.micorsoft.com/users/wyliu

摘要: 统一建模语言 UML 存在着面向对象建模的不足,例如,用多个模型表示系统,模型的一致性难以保证.UML 的工作主要集中于面向对象的分析和设计阶段,在实现阶段并没有做过多的努力等等,给系统设计带来了不便.提出了在 UML 建模环境中引入对象过程图(object-process diagram,简称 OPD),为多个模型的一致性提供了统一的标准,提高了程序文档的可理解性和可维护性,使得扩展后的 UML 具备从面向对象分析到程序设计的完全可视化的描述能力.

关 键 词: 对象过程图;对象过程方法;Rose 扩展接口
中图法分类号: TP311 文献标识码: A

统一建模语言 UML 在继承和扩展了面向对象建模的优点的同时,也存在着一定的面向对象建模方法的缺陷.例如,通过多个模型表示系统,模型之间的一致性很难得到保证;UML 的工作主要集中于面向对象的分析和设计阶段,在实现阶段并没有做过多的努力;UML 中对象之间只能通过消息传递机制来通信等等,这些都给系统设计实现带来了不便.

本文通过在 UML 建模环境中引入对象过程图(object-process diagrams,简称 OPD)从而扩展了 UML 建模环境.

OPD 对 UML 建模环境的扩展主要体现在以下两个方面:

(1) 设计方面.图1表示软件的颗粒度图.在软件系统的分析、设计到实现的过程也是软件颗粒度逐渐变小的过程.从图1可以看出,UML 及其建模工具所支持的最小软件颗粒度为“方法/操作”[1,2],这与实现还有很大距离.而 OPD 中代码到 OPD 视图是一一对应的,从而使其支持的最小软件颗粒度的大小为“语句”[3].这样,通过 OPD 编写的程序文档将会提高程序文档的可理解性和可维护性.

(2) 模型构造方面,OPD 的一个重要特点是用单一模型代替了多模型,使得无论设计人员还是维护人员都不必通过在许多个模型图之间的转换来理解系统,同时也为多个模型之间的一致性提供了标准.与 UML 相比,OPD 有更强的扩展和收缩能力,能够通过多个层次来理解系统,使对系统的理解从最高层逐步推进到系统的实现层,使程序设计和维护更具有层次性.

本文第 1 节简单介绍了 OPD 方法.第 2 节介绍了 UML,OPD 与程序代码之间的转化.第 3 节给出了系统结构框架和系统实例.第 4 节总结全文.

* 收稿日期: 2000-11-22; 修改日期: 2001-07-19

作者简介: 于明钊(1979－),男,湖北老河口人,硕士生,主要研究领域为软件工程,演化计算;刘文印(1969－),男,吉林榆树人,博士,研究员,主要研究领域为模式识别,软件工程;张宏江(1960－),男,河南叶县人,博士,高级研究员,主要研究领域为多媒体计算;李元香(1962－),男,湖北武汉人,博士,教授,博士生导师,主要研究领域为软件工程,演化计算.

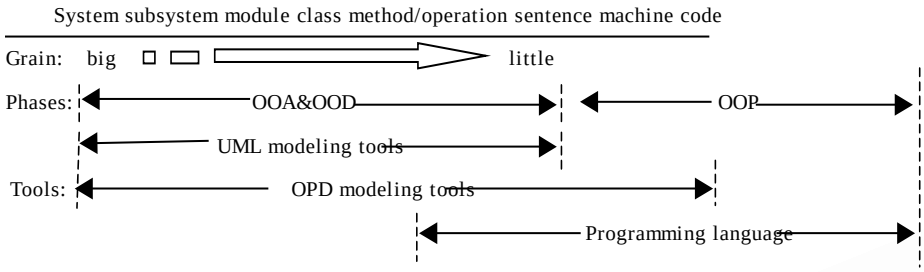


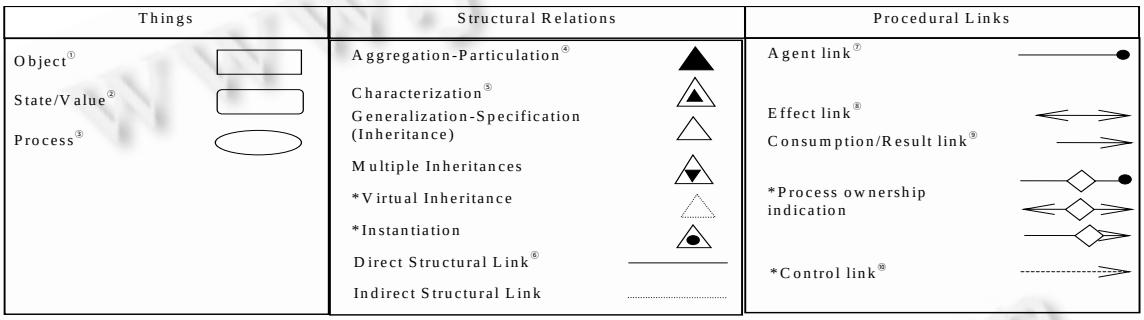
Fig.1 Illustration of software grains degree

图1 软件颗粒度与几个工具的处理能力

1 OPD 方法

OPD 最初作为对象过程方法(object-process methodology,简称 OPM)的图形工具被提了出来.与 UML 一样,OPM 也是一种系统分析设计语言,它把系统的静态和动态模型构造融合在单个模型框架中,消除了多个模型之间的一致性问题,从而使模型更容易理解.

OPD 中的描述符号如图2所示[2].后面,我们简单地介绍了这些符号的意义和用法.



①对象,②状态,③过程,④聚合,⑤特征,⑥直接结构链接,⑦工具调用,⑧变更调用,⑨消耗/产生调用,⑩控制链接.

Fig.2 Implementation-augmented OPD symbol set

图 2 OPD 中的符号

1.1 事 务

在 OPD 中,对象和过程同样被看做是构成系统事务的基本元素,系统由过程和对象组成.对象是一直存在的实体;过程是暂时存在的实体,它的存在至少依赖于一个对象.从设计和实现的角度,对象是以某种数据结构形式存在的变量,而过程则是对变量的操作或方法.OPM 通过 OPD 来表述系统对象和过程以及它们之间的联系.

对象类是拥有相同属性和行为的对象的抽象,它的概念等同于面向对象方法中的类.与 SmallTalk 相似,在 OPM 中,对象类也可以被看做是一个对象,从而使对象类成为一个相对的概念,而不是一个绝对的概念.

事务状态是事务运行到某一特定时刻时可能的值的集合.

OPD 中事务的一个很重要的特征是事物的迭代和分层能力.在复杂系统的设计中,OPD 通常通过多个层次的视图来表示,这种由顶向下的多层视图使得系统的理解和维护更加容易.

1.2 关 系

对象之间的关系分为聚合关系、特征关系和继承关系.聚合关系是用来描述两个对象之间的包含关系,如关系数据库和其中的表;特征关系是指特征、属性或一个操作与其所描述的对象之间的关系;与面向对象的设计方法中继承的概念一样,继承关系又可以分为继承、虚拟继承和多态继承.实例关系是类和实现它的对象之

间的关系.间接结构连接表示两个事务之间有一个或多个事务关系被忽略,这是一种很有用的关系,因为由于结构关系具有传递性,为了防止视图过于繁杂,两个事务之间的一些事务并不需要被指明.

过程和对象之间的关系分为变更调用、消耗和产生调用及工具调用.工具调用是对象作为参数被调用,对象存在于过程发生之前,对象在过程执行过程中状态不发生改变;变更调用则是对象被调用并且状态发生了改变;消耗调用指的是对象在过程执行中被删除;而产生调用则是对象在过程执行中被生成.

OPD 中使用菱形的方块来表示类或对象与过程之间的拥有关系.

1.3 流控制

OPD 使用至上而下的时间线和用过程之间联系表示的数据流来表示程序流程.对于程序转移语句使用控制链接来表示.控制链接连结一个过程或对象的状态到另一个过程,从而实现流控制.

使用 OPD 可以方便地实现程序的分支和循环控制.对于分支控制,可以根据控制对象状态数目的多少分别表示 IF-THEN-ELSE 和 SWITCH 语句;对于循环控制则可以根据控制对象的状态决定退出循环或进行下一次循环.

图3给出了一个 OPD 视图的一个例子,从中可以看到 OPD 符号的意义和用法.在后面的系统实现中,我们将进一步看到 OPD 在可视化面向对象程序设计中的强大功能.

```
Class Class8 {
  Class5& Process(const Class1& Object1,
                  const Class2& Object2,
                  Class3& Object3
                  ) ;
} ;
class C6 : virtual Class8 {} ;
class C7 : virtual Class8 {};
class Class4 : C6, C7 {} Object4 ;
Class5& Object5
        = Object4.Process(Object1,
                          Object2,
                          Object3) ;
```

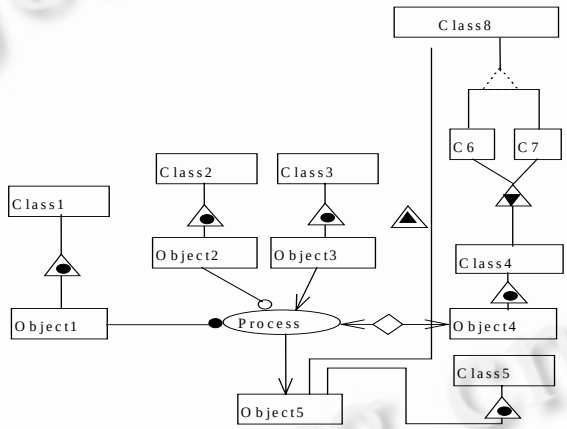


Fig.3 An implementation OPD

图 3 C++程序段和相应的 OPD 表示

2 UML,OPD,代码三者的转化

UML,OPD,代码三者的关系如图 4 所示.通过 UML 建造模型,利用代码生成机制生成代码,同时由代码和 UML 生成 OPD.用户对 OPD 进行编辑,产生完整的程序编码,即可视化程序设计阶段.对代码或 OPD 的改变会使 UML 模型发生改变,为了保持模型和代码的一致性,UML 模型也应作相应的修改.

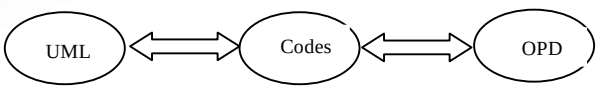


Fig.4 The relationship among UML, OPD and codes

图 4 UML,OPD,代码关系图

我们将 OPD 分为详细视图(detailed diagrams)和概要视图(summary diagrams),由 UML 模型直接生成的 OPD 是概要视图,概要视图对应的最小软件颗粒度为方法/操作,详细视图包括软件颗粒度从复合语句到语句阶段代码生成的 OPD.对详细视图的修改,只会影响到代码的变化,并不会改变 UML 模型;对概要视图的修改,则会使 UML 模型和代码同时发生变化.

对 OPD(OPM)和 UML 的建模能力的比较见表 1[4~6].在实际设计时,我们采用的是 OPD 的一个子集,对过程对象关系和对象之间通信能力加以限制.基于 UML 本身针对面向对象语言设计,这种限制并没有减弱 OPD 在实现面向对象设计方面的能力.

由前面对 OPD 的介绍可知,OPD 与程序代码之间存在着对应关系,代码都可以用 OPD 来表示,OPD 也都能够转化为代码的形式.问题的关键是怎样保证 OPD 与 UML 的一致性.

Table 1 The comparison of modeling ability between OPM (OPD) and UML

表 **1** OPM (OPD)与 UML 建模能力比较

	UML	OPM(OPD)
The way to describe system①	Meta model	Single model
Relationship between process and object②	Process belongs to one object③	Be treated as two things④
Communication between object⑤	Message communication⑥	Many ways⑦
Ability to describe status	Strong	Weak

①描述系统途径,②过程对象关系,③过程隶属于一个对象,④不存在隶属关系,⑤对象通信,⑥消息,⑦多种方式.

UML 用例图(use case)是 UML 其他方法的基础.迭代开发系统中的用例图往往也是通过多个层次表现出来的,从而使同样有层次性的 OPD 与用例图之间建立了一种对应关系,当由代码生成 OPD 概要视图时,每一层的用例图对应生成该层的 OPD,建立这种对应的基础是用例图正是描述系统执行者与用例之间的关系.这种关系很容易转化到 OPD 中对象与过程之间的对应.

在由 UML 和代码生成 OPD 的过程中,首先通过 UML 用例生成 OPD 的基本框架,进一步的实现则是通过代码与 OPD 的对应关系来完成.对代码分析过程中,将代码用树形结构表示[7],图5给出了一段排序程序段的树形表示,对于整个程序同样采取这种方法,根据代码树及 OPD 和代码之间的对应关系从而生成了多层次的 OPD.

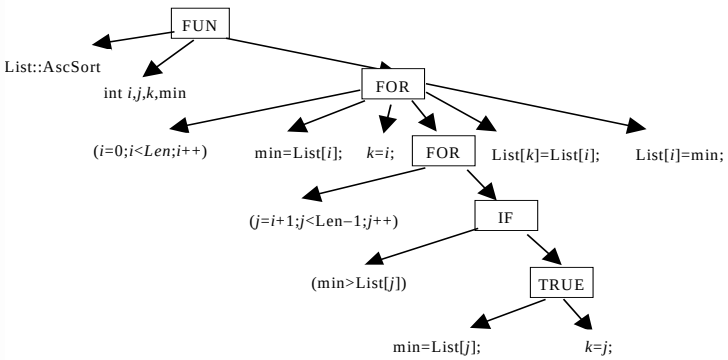


Fig.5 Illustration of a tree express of a function

图 5 AscSort 树的表现形式

在 OPD 的编辑修改过程中,只有对概要视图的改变,才会影响到响应 UML 的改变.由于前面生成的多层 OPD 与 UML 用例图是对应的,因此对某一层的 OPD 的改动将只会影响到该层 UML 视图.

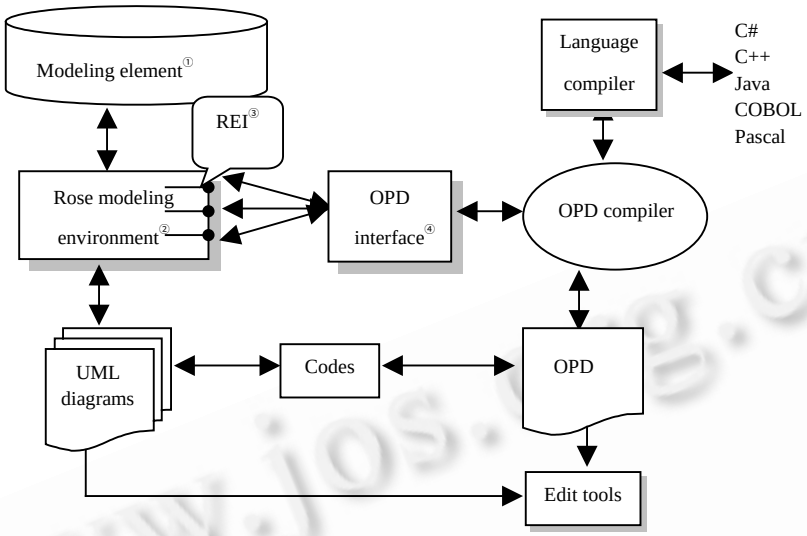
OPD 到代码和 UML 的转化,实际上就是如何在 OPD 建立在整体模型的语义机制和 UML 建立在动态与静态模型的语义机制之间建立一种对应.这个过程实际上是生成 OPD 的逆过程.通过 OPD 和代码的对应,可以方便地实现 OPD 到代码的转换,进一步可以通过代码到 UML 的逆向生成机制实现到 UML 的转化.

3 系统构造框架及实例

在系统实现过程中,我们以 UML 开发环境 Rose 为基础,加入 OPD 建模的功能,Rose 采用了 COM 组件技术,提供 Rose 扩展接口 REI(Rose extensibility interface),供人们进行二次开发[8].REI 是 Rose 脚本语言,只有通过 REI 接口才可以存取 Rose 应用程序提供的某些功能和建模元素信息.图6给出了系统结构图.下面的部分以一个图形识别框架系统的设计实现来具体描述系统功能.

设计目的是为了建立一个用于图形识别的、垂直的、可重用的基类.图形识别是工程图纸解释领域的一个

基本问题,系统可以识别出通过扫描仪扫描的原始工程图纸中的各种工程标记,如注释字、直线、弧、箭头、阴影等.



①模型元素,②Rose 应用程序,③Rose 扩展接口,④OPD 接口,⑤OPD 支撑工具.

Fig.6 System framework

图 6 系统结构图

首先,做出 UML 模型图.图 7、图 8 是 UML 的类图和交互图,这里,为识别专门建立了一个类.在类 RecognizerOf 的说明中,给出了短化线弧、字符框、阴影区域识别的 3 个例子.

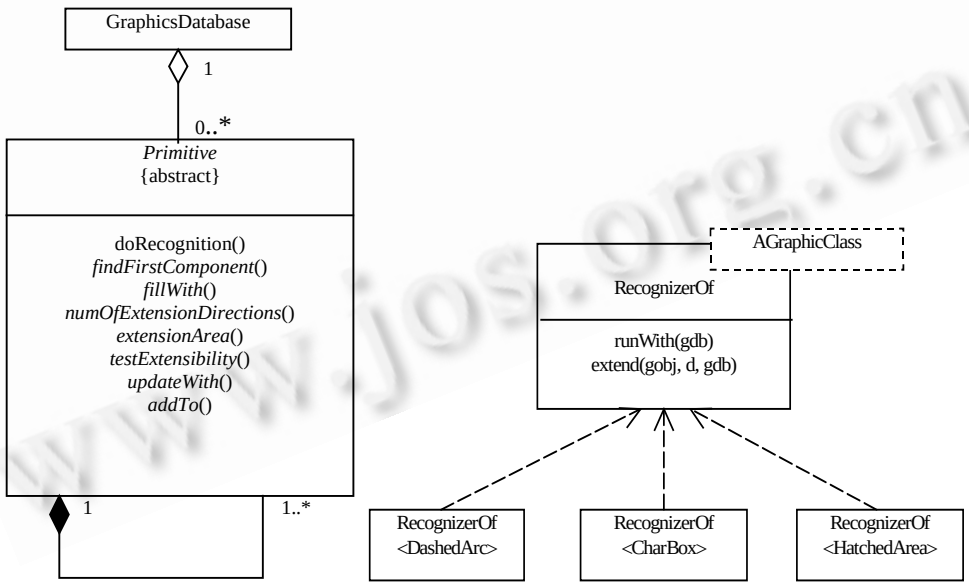


Fig.7 The class diagrams of graphic recognition

图 7 图形识别框架系统类图

在 UML 模型设计好之后,UML 的自动生成代码的正向变换机制会自动生成代码框架.编码完成之后,系统对编码进行分析,将编码用树形结构表示,生成 OPD 概要视图,进而根据系统需要,设计详细视图,如图 9 所示.由详细视图生成代码.图10 给出了类 RecognizerOf 的编码.

在 OPD 的编辑修改过程中,只有对概要视图的改变,才会影响到相应 UML 的改变.由于前面生成的多层 OPD 与 UML 用例图是对应的,对某一层的 OPD 的改动将只会影响到该层 UML 视图.这种改动首先通过 OPD 生成程序代码,进而通过由代码到 UML 的逆向生成机制来实现.

4 结论与展望

本文的主要研究成果是定义了 OPD 对 UML 进行扩展,并对基于 Rose 的扩展建模进行了可行性分析.主要体现在以下几点:

(1) 在可视化程序设计领域上,引入了概要视图和详细视图,充分发挥程序树形层次化结构的特点,使程序可以灵活地进行收缩和展开,极大地提高了程序的可理解性和可维护性.

(2) 将 UML 和 OPD 相结合,使得扩展后的 UML 具备从面向对象分析、设计到程序设计完全可视化的描述能力.

(3) 将 OPD 引入 UML 降低了设计阶段上的软件颗粒度,使得设计到程序设计阶段变得相当平滑.同时仍保持原有 UML 的风格:语言无关性和可视化等特点.

(4) 由于 OPD 的引入,增加了设计的可理解性和可维护性,从而使软件设计人员得到更大的解放.

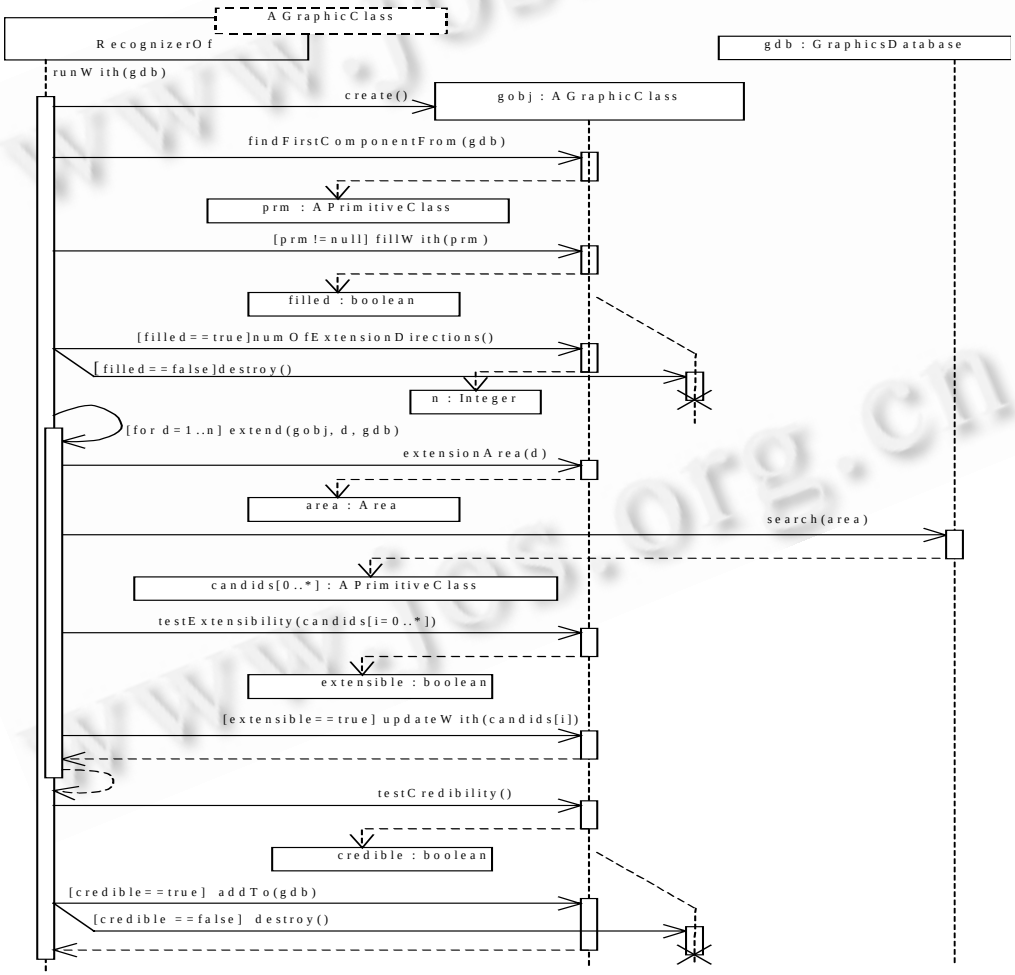


Fig.8 The interactive diagram of graphic recognition

图 8 图形识别框架系统交互图

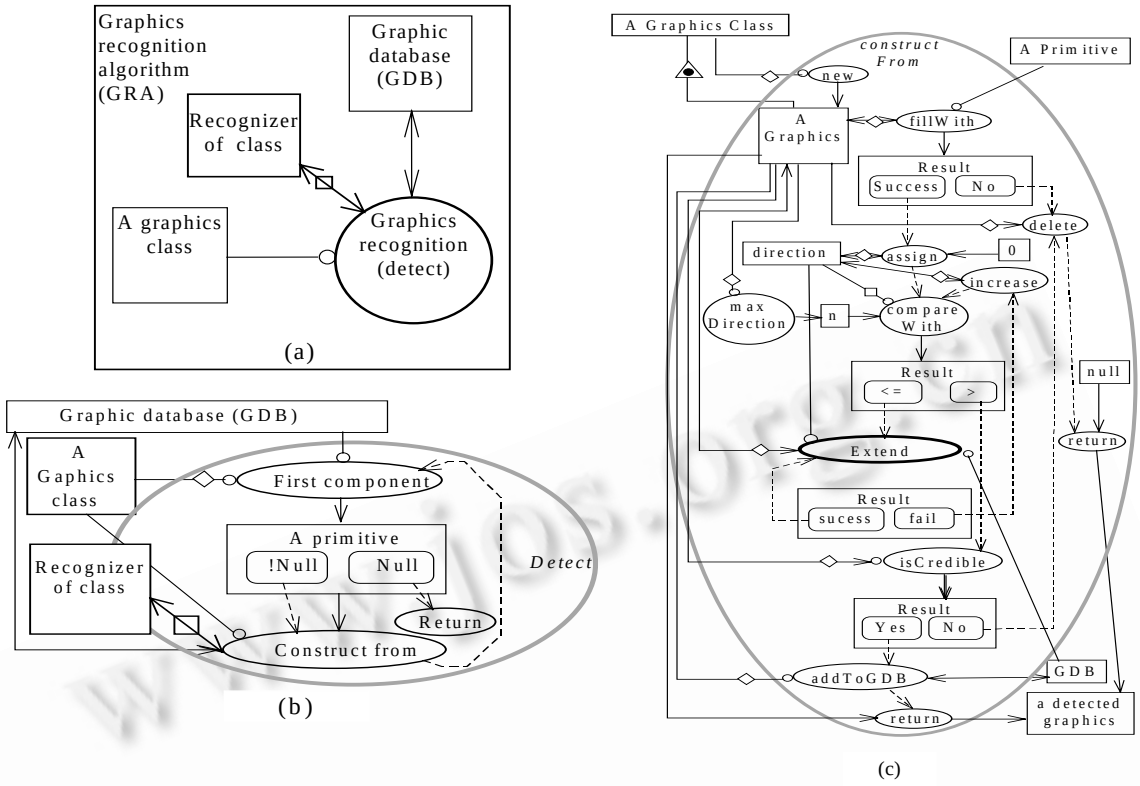


Fig.9 The OPD diagrams of graphic recognition

图 9 图形识别框架系统 OPD 图

```
emplate <class AGraphicClass>
        class RecognizerOf{
        RecognizerOf() {}
          void runWith(GraphicDataBase& gdb) {
                              while (1){
                          AGraphicClass* gobj = new AGraphicClass();
           Primitive* prm = gobj->findFirstComponentFrom(gdb);
                    if (prm == null)    return;
                       if (!gobj->fillWith(prm)) continue;
                           for (int d=0; d<=gobj->numOfExtensionDirections(); d++)
                     while (extend(gobj, d, gdb));
                        if (!gobj->isCredible())    delete gobj;
                        else gobj->addTo(gdb)
           }
           }
            boolean extend(AGraphicClass* gobj, int direction, GraphicDataBase& gdb) {
           Area area = gobj->extensionArea(direction);
           PrimitiveArray& candidates = gdb.search(area);
           for (int i=0; i<candidates.getSize(); i++) {
           if (!gobj->extensible(candidates[i]))    continue;
              gobj->updateWidth(candidates[i]);
              break;
           }
           if (i<candidates.getSize())    return true;
           return false;
             }
        };
```

Fig.10 The codes of class RecognizerOf of graphic recognition

图 10 图形识别框架类 RecognizerOf 的 C++编码

由于时间等原因,我们定义的 OPD 扩展 UML 还需要进一步地完善,系统的实现基本上还停留在实验阶段,还有待进一步研究和开发.我们接下去要做的工作可以归纳为:

(1) 进一步完善 OPD 扩展 UML 的定义,最终实现该图符语言的标准化,并与 UML 和 OPM 有机地联系在一起.

(2) 现阶段我们对从 OPD 到 UML 的转换,主要通过由代码向 UML 的逆向转化机制,而逆向转化机制正是 UML 的一个弱点.我们试图在 OPD 到 UML 中建立一种直接对应关系,从而直接实现由 OPD 到 UML 的转化.目前虽然已经取得了一些进展,但这种转化只是对简单的系统有效,对于复杂的系统仍然不能产生很好的效果,下一步的工作主要就是使这种转化能应用于复杂的系统之中,使系统早日商品化.

References:

[1] Liu, W., Dori, D. Object-Process diagrams as an explicit algorithm-specification tool. Journal of Object-Oriented Programming, 1999,12(2):52~59.

[2] Liu, W., Dori, D. Object-Process based graphics recognition class library: principles and applications. Software Practice and Experience, 1999,29(15):1355~1378.

[3] Dori, D. Object-Process analysis: maintaining the balance between system structure and behaviour. Journal of Logic and Computation, 1995,5(2):227~249.

[4] Zhou, Jian-wu. Research and implement of the extend of UML visual programm design [MS. Thesis]. Shanghai: East China Normal University, 1995 (in Chinese).

[5] Rational Software Corporation. UML Semantics version 1.1.

[6] Rational Software Corporation. UML Notation Guide version 1.1.

[7] Liu, Chao, Zhang, Li. Visual OO Modeling Technology——UML Tutorial. Beijing: Beijing Aeronautic Astronomical University Press, 1999 (in Chinese).

[8] Rational Software Corporation. Rational Rose 98 Rose Enterprise Edition.

附中文参考文献:

[4] 周建武.UML 可视化程序设计扩展的研究及其实现[硕士学位论文].上海:华东师范大学,1995.

[7] 刘超,张莉.可视化面向对象建模技术——标准建模语言 UML 教程.北京:北京航天大学出版社,1999.

Using Object-Process Diagram to Extend the UML Modeling Environment*

YU Ming-zhao[1], LIU Wen-yin[2], ZHANG Hong-jiang[2], LI Yuan-xiang[1]

[1](*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072, *China*);

[2](*Microsoft Research Asia, Beijing* 100080, *China*)

E-mail: mingzhao_yu@263.net

http://research.microsoft.com/users/wyliu

Abstract: UML is a powerful language for object-oriented modeling, but it has some explicit drawbacks. For example, it lacks a model check mechanism so it is difficult to keep the consistence of Meta Model. Moreover, the code building of UML is so scant in spite of the strong power on the OOP analysis and design steps. All of these bring lots of inconvenience in the system building. In this paper, the UML modeling environment is extended by using object-process diagram (OPD), through which people can set a uniform standard for multi-modeling and make the extended UML has the ability of both OOP analysis and design and visible program design.

Key words: OPD (object-process diagram); OPM (object-process methodology); REI (Rose extensibility interface)

* Received November 22, 2000; accepted July 19, 2001