面向对象的方法学

郭 江 (北京航空航天大学软件工程研究所)
廖越虹 (北方交通大学计算机系)

摘要:本文从方法学的角度出发,讨论了面向对象的分析(*OOA*)、设计(*OOD*)、实现(*OOI*)。在此基础上,详细讨论了面向对象开发的七个步骤,最后给出了面向对象软件开发的图形表示。

一、引言

现在已经提出了许多软件开发模型,如瀑布模型、增量开发模型、空间开发模型等,它们对软件危机问题的解决都有一定的帮助。目前的面向对象的技术则是以封装、继承、数据抽象,动态束定为特征来解决软件可靠性、软件生产率,以及软件可维护性等方面的问题。

面向对象的方法学就是使用面向对象的技术来进行软件系统的分析(OOA)、设计(OOD)、实现(OOI)的方法学。尽管这里我们将软件的开发分成了三个部分,但实质上它们是交织在一起的。

面向对象的方法如功能分解一样,也是采取分而治之的方法,只是分解的指导思想有所不同而已。

在面向对象的系统分解中,可以将系统看作是对象或对象类的集合。高层的分析和设计主要目标就是确定这些对象以及它们所提供的服务。提供的方式是对象关系的客户/服务器模型,其中对象间是通过传送信息的方式来进行交互的,并通过调用对象来实现一个处理过程。客户/服务器模型的使用导致将系统描述成"责任驱动"式的。

面向对象的技术使详细设计,包括过程实现和数据结构的确定尽量推迟到开发过程的后一阶段来进行。因此基于对象表示的系统就可以有更大的灵活性,换句话说,这样就可以很容易地实现上层的改变而无需改动系统设计。因此,对象的开发就可以集中于数据抽象而不是在对象说明中确定数据结构。

尽管面向对象的程序开发是由底向上的,但程序开发和编码之间的差别与基于过程的系统生命周期没什么两样。但是在后面的阶段,单独的代码模块(称为类)内部仍需要高层的功能分解和自顶向下的系统设计开发工具,如数据流图(DFD)。另外一些图形工具在 OO 系统生命周期的不同阶段也很有用,如对象关系图(ORG)、客户/服务器图,继承表(chart)及合作图等。

由于 OO 实现的目标之一是开发通用的类并将之放入类库之中,因而同时考虑自顶向下的分析和由底向上的设计就能导致最键壮的软件系统。因此在面向对象的方法学中使用这种自顶向下,由底向下和中间两分的混合模式是最自然的了。

二、系统分析和设计的方法学

在面向对象的方法学中,问题空间对象的分析直接映射到解空间的对象上,如果也和实现使用相同的语言环境,那么第二次也可直接映射到代码,实际上,系统设计和系统实现使用相同的语言环境可以认为是面向对象方法的极大优点之一。

面向对象的程序设计环境强调从 OO 设计阶段就开始实现阶段,在某种意义上讲高层的自顶向下的设计,可以通过确定库中现存的类,使用继承派生和构造新类的由底向上的方法来实现。这样自顶向下的设计和由底向上的构造并发进行,如图 1 所示。

下面简要叙述 OOA,OOD 的开发方法

1.OOA

OOA 的主要目的就是自上而下地进行分析,即将整个软件系统看作是一个对象,然后将这个大的对象分解成具有语义的对象簇和子对象同时确定这些对象之间的相互关系,在将对象分解成子对象集的过程中,同时概

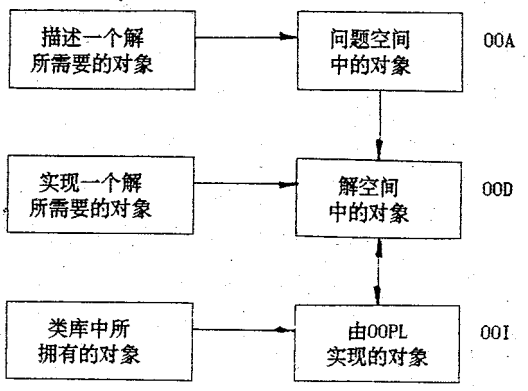

图 1　面向对象的开发模型

括抽象，这样就形成了整个系统的体系结构。为了将系统初步细化，要反复进行对象分解的过程，同时在分解过程中要确定哪些是在类库中可以直接使用的，哪些是需对类库中的类稍加修改就可以使用的，哪些是类库中没有而必须加以设计的。在面向对象的技术中，最重要的就是 OOA 技术，它是以后 OOD、OOI 的基石，为此给出其详细步骤。在 OOA 的分析过程中，对域可进行静态分析，而后进行动态分析。所谓静态分析需将问题空间对象分解成若干子对象，写出应用系统的需求说明；而动态分析则是根据应用系统的需求说明从类库中选择一些功能相近的类来构成原型，并对原型进行评测，在评测过程中可形成应用系统运行时的快照，这可以有三个阶段的时序快照，即开头，中间和结束时的快照，通过观测这些系统运行时类库中的类生成的三个阶段的对象，就可以判定该原型系统是否能很好满足用户的需求。因而面向对象的技术是天生支持快速原型法的技术，以下是 Bailin[1]提出的 OOA 的七个步骤：

(1)确定问题空间中的关键对象；

(2)确定何为主动对象何为被动对象；

(3)建立主动对象之间的数据流；

(4)将对象分解成子对象；

(5)检查所需要的新对象；

(6)在新对象下组织功能；

(7)确定适合新对象的领域；

实际上可以将第 1 步至第 3 步看作是一步，用一步来完成。而第 4 步至 7 步则反复执行，直至满意为止，这样就完成了 OOA 的任务。图 2 是 OOA 的形式表示[2]。

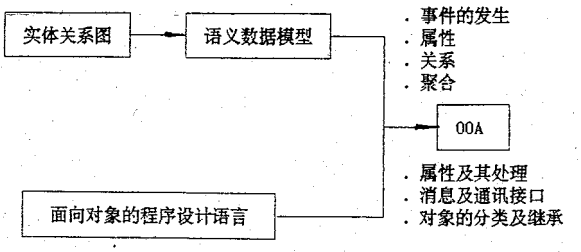

图 2　OOA 的形式表示

2.OOD

OOD 的任务是将对象及其相互关系进行模型化，建立分类关系，解决问题域中的基本构建。因此，OOD 要去查询类库，对那些类库中没有的，需要重新设计或派生的类要进行新的分析和设计，给出这些新类的形式化算法或描述性算法。这样 OOD 实际上是承上启下的一个环节，。即要利用下层(即类库)已有的部件，又要对分析中产生的需求加以完成。

对 Booch[3]提出的五个阶段稍加修改就很适合于 OOD 了。因此有如下六个步骤：

(1)确定对象及其属性；

(2)确定影响对象的操作；

(3)建立对象的可视性；

(4)用客户／服务器和继承机制来建立对象之间的关系；

(5)建立接口；

(6)实现每个对象。

从这里可以看出系统设计和代码设计已经变得界限不清了。这就是既需要面向对象系统设计来在高层抽象上分析系统，又需要认识到用面向对象的程序设计语言进行设计的最佳方式是：利用库来派生新对象以便由底向上地加以实现。这些库类是设计对象或对象类的代码版本，一般而言，类是用于描述解空间对象和设计对象的抽象数据类型的编译时描述的。

自顶向下的分析和由底向上的类设计可看作是整个面向对象软件生命周期的基石，因此必须是并发或至少是交替进行的，为了进行重用，经验证的类必须看作是设计阶段的一部分，而不仅是实现阶段的一部分。

这个分析阶段和类设计与实现有直接的联系。对象

的分解可以在实现阶段用分类、聚集和概括的过程来确定。这样就可用数据流图来描述，而其中的节点则变成了对象(即实现阶段的类)。

三、面向对象的系统开发

前面论述了 OOA、OOD 的方法，下面将较详细地讨论面向对象的系统开发，系统开发过程分成以下 7 个步骤：

(1)确定面向对象系统的需求说明；

(2)确定对象(实体)及其可提供的服务(接口)；

(3)用所提供的服务和所需的服务来建立对象之间的交互；

(4)使用较低层的 EDFD／IFD 将分析阶段溶进设计阶段；

(5)使用类库来由底向上地进行设计；

(6)根据需要引入层次的继承关系；

(7)对类进行聚集和概括抽象。

1.确定系统的需求说明

这一阶段是系统的高层分析，用于确定系统的对象以及其所提供的服务(这类似于系统的功能)。在这一阶段，主要使用 OOA 技术来进行工作，产生的结果是面向对象的需求说明(OORS)，包括时间细节、硬件使用、代价评估及其它有关的文档。这里可以使用 Goad 和 Yourdon【4】。shlaer 和 Mellor[5]提出的 OOA 技术。

2.确定对象

在分析和高层设计阶段，必须确定对象、属性以及它们所提供的服务。这里确定了功能特征，但并不需要说明怎样具体实现，因而可使用面向对象的图来描述，而且可以用映射现实世界对象的方法来确定对象，但是，这一层上的抽象并不必分解对象和寻找更基本的对象表示，这些更适于放在详细设计阶段，而在这里还可以使用对象字典来作为确定对象的手段。对象的确定不应孤立在需求说明和分析以及设计阶段，因为设计好的对象类将会反复使用。因此对象和最终类的确定还要定义会影响其它对象的操作，以及它所能提供的服务。这样就必须确定显示的接口。

3.建立对象之间的交互

在这一阶段，由于用所需的服务和所能提供的服务来建立对象之间的交互(类似 Bailin【1】的 EDFD 的数据流)，因而使用类似 EDFD(实体数据流图)和实体关系图(ERD)的图形表示来描述对象之间的交互特别有用，在这时也许更合适的是与 DFD 等价的面向对象的 IFD(信息流图)而不是 EDFD。这里的“信息和消息及消息参数有关。因为在一般情况下，数据包含在对象内部，而不像在功能分解技术中由 DFD 描述的那样流动。这里也可以使用交互式的对象浏览工具来追踪对象的层次结构和对象的关系。这样也就可以检查对象的属性和例程。

4.使用较低层的 EDFD／IFD 来将分析阶段溶进设计阶段

随着分析阶段溶进设计阶段，就需要使用低层的 EDFD 和 IFD 来描述对象的细节，从这一阶段起就要考虑由底向上的设计方法了，这里，重用以前的设计成分，即类来设计是 OO 策略的一个重思想。在一些语言中，类是嵌入在其它类中的，在一个纯面向对象的生命周期(见图 1)中，特别是系统分析、设计和实现都用同样语言的生命周期，是否表达嵌入类的决策将反映正在使用的语言，在 Eiffel 中，嵌入是不可能的。而是由类去调用其它类，这样类包含一个给定的属性(即抽象数据)，就是对这个定义了抽象数据类型实现的类的一个对象的简单调用。在图形化表示的设计层上，对象仍是单独保留的。但是，在顶层上，高层抽象的对象将由单个的表示块来表示。

5.使用类库来由底向上地设计

这一阶段，要并发地进行由底向上的设计。这里可以使用实体关系图、EDFD、信息流图(IFD)来描述分析好的对象的更详细的内部结构。而对象本身是类库中更基本的对象使用客户／服务器和抽象的概念来构造的。库本身也包含了以前的应用系统中所建立的类。同时这阶段也可以开始一些底层类的实现工作(编码加测试)。

6.根据需要引入层次的继承关系

随着在详细设计中确定越来越多的对象，就需要对类集进行重新评价，反复分析是否需要新的父类和子类，建立继承关系图。这一个过程就是开发一个对象的逻辑结构而且不能丢失对象。因此这一阶段就可以提供一个结构良好的层次关系，以使将来的项目开发可以重用这个结构而无需重新设计继承图，这对软件系统将来的维护也有极大的好处。这个阶段是在类开发的簇模型概括阶段进行的。

7.对类进行聚集和概括

对类进行聚集和概括，就需要反复考虑所描述系统的 EDFD 和 IFD。这一阶段的原型化能导致将原型的评价反馈回来以便对需求文档进行修改，并进一步开发特定的类。尽管这和传统的生命周期模型相反(这样的反馈本质上是从传统生命周期的一端到另一端)，但在面向对象的技术面前就变为可行的了，而且还能提供一个更可靠、健壮和有用的软件系统。

在这一阶段确定和开发的系统类要经历另一个阶段的开发，即遵循簇开发模型来概括，在这一阶段，组成部分要继承开发，直到它足够通用和健壮，这样就能将之放到成分库中。为将来的维护以及以后项目的开发所利用，为长期的利益做了准备。

从这里可以看出 OO 方法学是类设计、系统设计一步接一步进行的。尽管这里将开发过程分为七步，但实际上正如在前面所提出的：OOA、OOD 和 OOI，三者是相互交织在一起的，即系统的设计将会影响到类的设计和实现，而类的设计和类的使用反过来又影响着系统的分析和结构的设计，因此这七个阶段是密切相关的。

由于类已不足以描述空间的对象，因而提出了簇概念。簇模型是 Meyef[6]提出的，作为密切相关的类组的生命周期，主要确定了三个阶段。第一是系统设计人员写的说明，然后是设计和实现(像 Eiffel 一样的一个语言中的过程)，而最后是验证和概括。注意该模型适合于软件类而不适合于软件系统。类的说明是系统说明提精而产生的，并且尽可能详细地描述了类的服务和语义，最好用抽象数据类型的形式说明来进行描述，如图 3 所示。

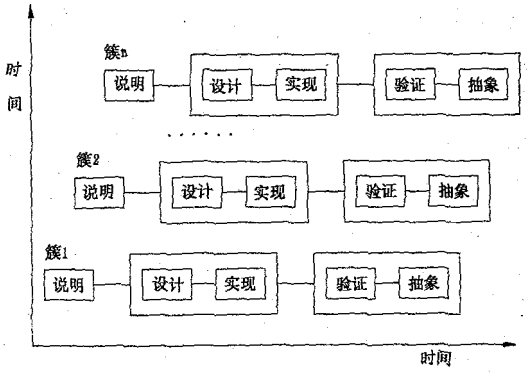

图 3　簇的生命周期

四、面向对象软件开发的图形表示

1.面向对象生命周期的表示

在软件开发生命周期的总体结构中，面向对象的图形表示必须考虑隐含的高度重叠和反复，因此用"喷泉模型"比用修订后的瀑布模型更适合。首先，它提供了面向对象软件生命周期中各阶段的图形表示，并清晰地表明了面向对象技术的反复和重叠。其次，由于成功软件项目的基础是它的需求分析和说明，因而这个阶段放在图的底部。生命周期就因此而向上发展，仅在必要的维护时才落回来。这就有效地将生命周期回复到一个较低的层次。这样的模型用于特定的环境之中就会形成合适的软件生命周期(如原型化、OOPL)。

该系统不应仅看作是系统的生命周期，也应看作是许多相互独立的类的周期。在类开发和系统说明之间可以更容易地作交互式的修改，因而不再需要在系统生命周期的早期冻结整体的系统需求说明——这就是面向对象方法学的优势之一。因此，由于一个面向程序的开发本质上作为一个类(通常独立开发)的交互系统，所以生命周期模型可以更准确地用于单个类的开发周期而不是系统，但这种方法一般是并发地用于类组即簇的开发。这样图中所表示的模型既可直接用于簇又可用于单个的类。通过明确地引入聚集和概括阶段，就可以修正特定情况下的类。以便它们具有足够的普遍性来用于各种应用中而不仅是原来开发它们的狭窄领域。这就需要较大的努力而不是短期一次性的设计与实现，但是从长期角度出发，构造好足够广泛的库以后，将能极大地减少总体系统的开发时间和工作量，这是在面向对象设计环境中强调代码重用的结果。如果在总体框架中使用自顶向下的面向对象的系统分析，而且在高层设计中采用由底向上的方法来开发类，那么就能很好地达到这个目标。

簇模型在软件生命周期中的重要性是很显然的。它的出现是系统说明不断出现分支和模块不断提精的结果。图 4 综合了系统的概念和簇生命周期怎样形成的结果，并表示了需求以及实现阶段的成长过程，图中的单个的类或类簇各自经历了自己的簇和喷泉生命周期的反复。这个图的优点在于动态表示了系统随着用户和分析员知识的不断增长而发展，而且还可以将该图结合到总体的生命周期模型之中。由于用户的需求在不断变化，

这种非集中式的面向对象的体系结构具有很大的优势。

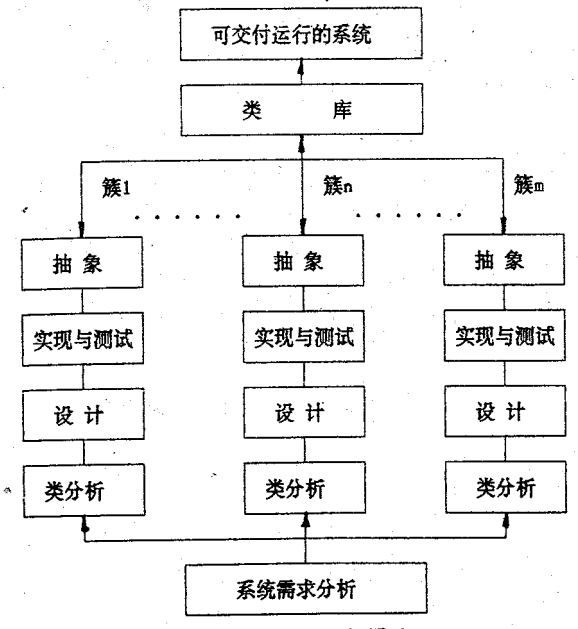

图 4 系统开发周期

类簇也经历着概要设计、详细设计等过程。系统的需求分析和设计产生了大量的簇，而簇又使程序可以在后期对类进行详细设计和实现，这样就能使在后期才进行的由底向上的方法结合到系统设计中，这种方法就使得系统需求分析和系统设计变成了一个非常容易的过程，而设计也不是第一次决策的结果，这样就比自顶向下的功能分析有更大的好处。因为总体系统综合是在系统生命周期后期进行的。因此需求方面的变化只可能引起单个簇的改动而不需要进行大规模的重新设计。

2.面向对象设计方法的图形表示

类或簇的设计和提精过程如果用图形来表示，就需要表示单个的对象类、子类以及它们的关系，特别是需要表示继承和客户／服务器机制。因此需要一种非特定语言的表示方法，对象表示应该不仅能去描述在设计层上的对象(或实体)，而且还要能描述运行时的对象类，这样就能保证表示先于实现(有点儿存在主义的味道)。由于设计对象之间的关系(也是类之间的关系)可以在生命周期的不同阶段用同样的方法来表示，因此可以用"O／C"(对象或类)来描述这种较一般的情况。这些图形表示必须具有下列性质 O／C 名字，所提供服务和可用信息属性的公用接口。Booch 图【3】稍微修改就能满足这些需要。另外还必须表示对象与对象(或类与类)之间的关系。就客户／服务器而言，O／C 的属性是抽象数据类型的属性，因此为了实现它就需要调用另一个 O／C，这样左边域的箭头就指向另一个服务器 O／C 的名字。继承关系则由左边域的箭头来指向如图 5 所示：

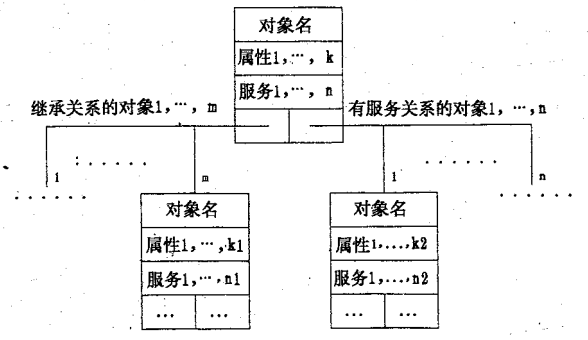

图 5 对象(类)及其之间关系的表示

对象的确定，是 Booch 方法学[3]中需要确定的第一步，尽管面向对象的术语还没有标准化，但都有一个共同的线索，其中一些与语言很有关系，一般可以确定两类对象：一类是对象的直接说明(用其所有的操作和属性来进行说明)，另一类是定义一个(对象)类作为一个模板。在类需要多个实例时总是使用后一种方式。如在 Eiffel 中，所有的对象都是类类型(Class Type)的实例(类类型本身定义为抽象数据类型的实现)。换句话，代码包含了类，而类又是在运行时创建对象实例的模板。

参考文献：

[1]bailin, " An Object- Oriented Requirements Specification Method, " ACM Commu Vol.32 NO.5, 1989

[2]郭江，"信息系统的面向对象分析"，计算机科学，Vol.19No.1, 1992

[3]booch and Grady, " Object- Oriented Development, " IEEE Transactions on Software Engineering, Feb.1986

[4]Coad and Yourdon, Object- Oriented Analysis.prentice Hall, 1990

[5]Shlaer, Sally, Mellor and Steve, Object-Oriented System Analysis. Prenticc Hall, 1988

[6]Meyer and Bertrand, Object-Oriented Software Construction Prentice Hall, 1988。