

第1章 UML简介

- 了解面向对象机制与可视模型
- 了解图形说明类型
- 了解UML框图类型
- 用可视模型开发软件

本章介绍UML（统一建模语言，Unified Modeling Language），这是最广泛使用的面向对象系统的建模方法。本章首先介绍可视模型及其如何构造应用程序，然后介绍UML的历史与现状，最后介绍开发过程及模型的作用。

UML由几种不同框图构成，本章简要介绍这些框图，后面的章节还将详细介绍每个框图。

面向对象机制简介

面向对象是软件行业的新术语。各个公司纷纷采用这个新技术，将其集成到现有应用程序中。事实上，大多数当今开发的应用程序都是面向对象的。什么是面向对象呢？

面向对象机制是另一种观察应用程序的方式。利用面向对象方法，把应用程序分成许多小块（或对象），这些对象是相互独立的。然后可以组合这些对象，建立应用程序。可以把它看成砌砖墙。第一步要建立或购买基本对象（各种砖块）。有了这些砖块后，就可以砌出砖墙了。在计算机领域中建立或购买基本对象后，就可以集成起来，生成新的应用程序。

面向对象机制的一个主要好处是可以一次性地建立组件，然后反复地使用。就像砖块可以重复利用盖城墙、盖房子、盖太空飞船，基本面向对象的设计和代码可以重复用于会计系统、库存系统或订单处理系统。

这种面向对象机制与传统开发方法有什么不同呢？传统开发方法集中考虑系统要维护的信息。用这种方法时，我们要向用户询问他们需要什么信息，然后设计保存信息的数据库，提供输入信息的屏幕并打印显示信息的报表。换句话说，我们关注信息，而不是关注信息的作用或系统的功能。这种方法是数据为中心的，多年来用它建立了成千上万个系统。

以数据为中心的模型适合数据库设计和捕获信息，但用来设计商业应用程序就有问题。一个主要问题是系统要求随着时间不断变化。以数据为中心的系统可以方便地处理数据库变化，但很难实现商业规则变化和系统功能变化。

面向对象机制的开发正是要解决这个问题。利用面向对象机制，我们同时关注信息与功能。因此，我们可开发密切关注和适应信息与功能变化的系统。

要实现灵活性带来的好处，只能通过设计好的面向对象系统。这就要求了解面向对象的基本原则：包装、继承与多态。

包装

在面向对象系统中，我们将信息与处理信息的功能组合起来，然后将其包装成对象，这称为包装（encapsulation）。另一种理解包装的方法就是把应用程序分解成较小的功能组件。例如，我们有与银行帐目相关的信息，如帐号、结余、客户名、地址、帐号类型、利率和开户日期。我们还有银行帐目的功能：开户、销户、存款、取款、改变类型、改变客户和改变地址。我们将这些信息与处理信息的功能包装成帐目对象。结果，银行系统对帐目的任何改变就会在帐目对象中实现。它是所有帐目信息与功能的集合。

包装的另一好处是将系统改变的影响限制在对象内。可以把系统看成水，所需的改变看成大石头。一块石头投进水里，会溅起一片水波纹，它们经过湖面、冲向岸边、振荡并与其他水花碰撞。事实上，有些水花会溅上岸边和湖外。换句话说，一石击水引起了巨大的波浪。如今我们要包装湖水，将其分成较小的水体，并围起来。然后，扔出一块石头，仍然水花四溅，但只能限制在围起来的小范围内。因此，通过包装湖水，我们限制了击石引起的波浪，如图1.1。



图1.1 包装：湖水模型

下面要把这个包装思想用于银行系统。最近，银行管理层决定，如果客户在银行有信用帐号，则可以用信用帐号作为支票帐号的透支额。在无包装系统中，我们要用类似散弹猎枪的方法进行影响分析。我们并不知道系统中所有取款功能用在哪里，因此要到处寻找。找到之后，我们要根据这个新要求进行分析。如果我们水平很高，则可能发现系统中80%的取款功能。而利用包装系统，则不必用散弹猎枪方法进行分析。我们只要查看系统模型，寻找取款功能包装在哪里。找到帐目中的取款功能后，只要在对象中一次性地作出所要求的改变，就万事大吉了。从图1.2可以看出，只要改变Account类。

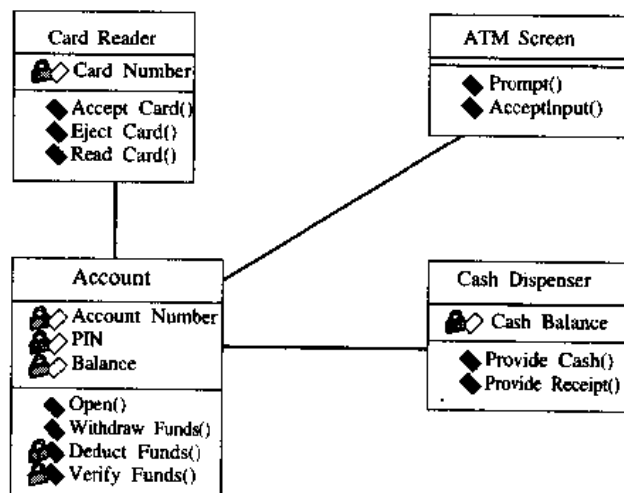


图1.2 包装：银行模型

与包装类似的概念是信息隐藏 (information hiding)。信息隐藏就是不向外部显示对象细节。对于一个对象，外部就是对象之外的一切，包括系统其他部分。信息隐藏提供了与包装相同的优势：灵活性。第6章将详细介绍这个概念。

继承

继承 (inheritance) 是第二个基本面向对象的概念，它与小约翰继承的万贯家财毫无关系，而与你的鼻子像你父亲的鼻子有关。在面向对象系统中，继承机制可以根据旧对象生成新对象。子对象继承父对象的特性。

自然界中有许多继承的例子。哺乳动物有几百种：狗、猫、人、海豚等等。每种动物都有一些共性和个性，如有毛发、热血、哺乳。用面向对象术语，哺乳动物 (mammal对象) 有一些共同特性。这个对象是狗、猫、人、海豚等等的父对象。狗对象继承mammal对象的特性，还要有一些狗对象自己的特性，如转圈跑和流口水。面向对象机制借用了自然界中的继承概念，如图1.3，因此可以对系统采用相同的概念。

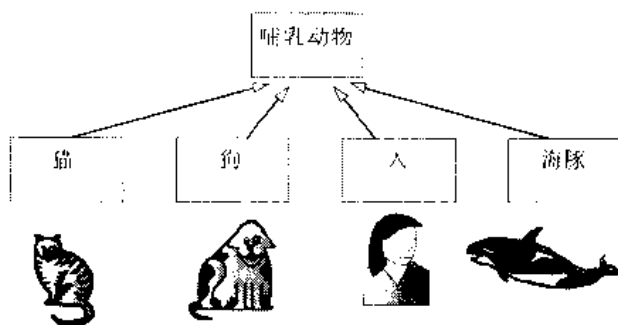


图1.3 继承：自然模型

继承的主要好处之一是易于维护。发生影响所有哺乳动物的改变时，只要改变父对象，子对象自动继承这个变化。如果所有哺乳动物突然变成冷血，只要改变mammal对象，猫、狗、人、海豚和其他子对象自动继承哺乳动物新的冷血特性。

在面向对象系统中，窗口是继承的一个例子。假设一个大系统有125个窗口。一天，客户要求所有窗口显示放弃消息。在没有继承的系统中，我们要吃力地到每个窗口中作出改变。而在面向对象系统中，所有窗口只要从一个父窗口继承，这时只要到父窗口中一次性作出改变即可。所有窗口自动继承这个变化，如图1.4。

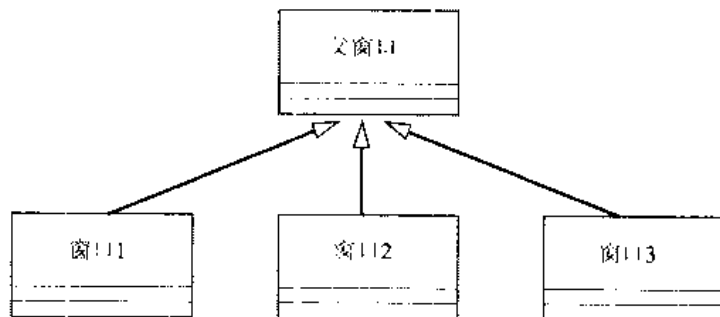


图1.4 继承：窗口模型

在银行系统中，可以对不同类型的帐目使用继承。假想银行有四种帐目：支票、存款、信用卡和存款证明。这四种帐目有一定的相似性。每个帐目都有帐号、利率、所有者等等。因此可生成父对象account，保存所有帐目的共同特性。子对象则在继承特性的基础上增加自己的特性。例如信用帐目还有信用额度和最少付款额，存款证明还有到期日期。对父对象的改变影响所有子对象，而对子对象进行改变则不会影响其他子对象和父对象。

多态

面向对象的第三个原则是多态（polymorphism）。多态的定义是多种不同形式、阶段或类型发生的事，表示特定功能有多种形式或实现方法。和继承一样，多态也有自然界中的例子。比如让对方说话，人可能说“你好”，狗会汪汪叫，猫会咪咪叫，但也可能就是不理你。

在面向对象系统中，就是特定功能有多种实现方法。例如，我们可能要建立一个绘图系统，用户要画线、圆或者矩形时，系统发出绘图命令。系统中有许多形体，各有不同的绘图功能。因此，用户要画圆时，调用圆对象的绘图命令。利用多态，系统运行时确定要画的形体类型。而如果没有多态，则绘图功能的代码可能如下：

```
Function Shape.drawMe()  
{  
    CASE Shape.Type  
        Case "Circle"  
            Shape.drawCircle();  
        Case "Rectangle"  
            Shape.drawRectangle();  
        Case "Line"  
            Shape.drawLine();  
    END CASE  
}
```

利用多态，则只要对所画对象调用drawMe()函数，命令如下：

```
Function draw()  
{  
    Shape.drawMe();  
}
```

每个形体（线、圆、矩形等等）用自己的drawMe()函数画图。

和面向对象的其他原则一样，多态的好处之一是易于维护。如果应用程序要画一个三角形，在非多态情形中，就要给Shape对象加一个新的drawTriangle()函数，Shape对象的drawMe()函数也要修改成适应新形体的类型。而利用多态，则生成新的三角形对象，用drawMe()函数绘图。启动绘图操作的draw()函数根本不必改变。

何谓可视化建模

如果要扩建房子，你可能不会买一大堆木材，慢慢钉成所要的样子，而会按照蓝图规划和构造之后再着手扩建。这样的扩建能更持久，而不会因为一场小雨把一切冲得粉碎。

软件中的模型也一样，是系统的蓝图。蓝图可以帮你规划要作的补充，模型可以帮你规划要建的系统。这就可以保证系统设计良好，要求得到满足，系统能在要求改变时站得住脚。

收集系统要求时，把用户的业务需求映射到开发小组能理解的要求。最终你要利用这些要求产生代码。通过将要求映射为代码，可以保证代码满足这些要求，代码也能方便地回溯要求。这个过程称为建模。建模过程的结果就是可以跟踪从业务需求，到要求、到模型、到代码的过程及其相反的过程，而不会在这个过程中迷路。

可视化建模将模型中的信息用标准图形元素直观地显示。标准对实现可视化建模的通信功能至关重要。可视化建模的主要目的就是用户、开发人员、分析人员、测试人员、管理人员和其他涉及项目人员之间的通信。利用非可视信息（文本）也能进行通信，但人类毕竟是百闻不如一见，通过图形比通过文字更容易理解事情的结构。利用系统的可视化建模，可以在几个层次上显示系统如何工作。我们可以建模用户与系统间的交互，可以建模系统对象间的交互，甚至可以建模系统之间的交互（如果需要）。

建立模型后，可以向所有感兴趣的方面显示这个模型，让他们对模型中的重要信息一目了然。例如，用户可以通过模型直观地看到用户与系统间的交互，分析人员可以看到系统对象间的交互，开发人员可以看到要开发的对象和每个对象的任务，测试人员可以看到对象间的交互并根据这些交互准备测试案例，项目管理人员可以看到整个系统及各部分的交互，而信息总管可以看看高层模型，看看公司的各个系统如何相互交互。总之，可视化建模提供了向各有关方面显示系统计划的强大工具。

Booch、OMT与UML

可视化建模的一个重要问题是用哪种图形标注方法表示系统的各个方面。这个标注方法应能向各有关方面传达意图，否则模型就用处不大。许多人都有自己的可视化建模图注方法，最常用的图注方法有Booch、对象建模技术（OMT）和统一建模语言（UML）。Rational Rose 98i支持这三种方法，但UML是大多数公司采用的标准，是ANSI和OMG等部门采用的标准。

Booch方法是按其发明者Grady Booch命名的，他是Rational软件公司的首席科学家。他著有儿本关于可视化建模的需求与好处方面的书籍，并开发了表示模型各个细节的图注方法。例如，图注中把对象画成云图，表示对象几乎无所不包。Booch图注方法还用各种箭头表示对象之间的关系类型。图1.5是用Booch图注方法表示的对象和关系。

OMT（对象建模技术）图注方法来自James Rumbaugh博士，他著有多本关于系统分析与设计的书籍。在《系统分析与设计》一书中，James Rumbaugh博士介绍了模型系统在实际组件（称为对象）中的重要性。他提供的OMT方法有许多追随者，Rational Rose和Select OMT等行业中标准软件建模工具都支持这个方法。OMT使用比Booch更简单的图形表示系统。图1.6是用OMT图注方法表示的对象和关系。

统一建模语言（UML）图注方法是Grady Booch、Dr. James Rumbaugh、Ivar Jacobson、Rebecca Wirfs-Brock、Peter Yourdon和许多其他人员集体智慧的结晶。Jacobson是个学者，他的著作包括在事务包“使用案例”（use cases）中捕获系统要求。第3章将详细介绍使用案

例。Jacobson还开发了系统设计方法OOSE（Object Oriented Software Engineering，面向对象软件工程），着重用于分析。Booch、Rumbaugh和Jacobson是Rational软件公司的三剑客，进行了UML的标准化和改进。UML符号与Booch和OMT图注符号相似，并引入了其他图注方法的元素。图1.7显示了UML图注的样本。

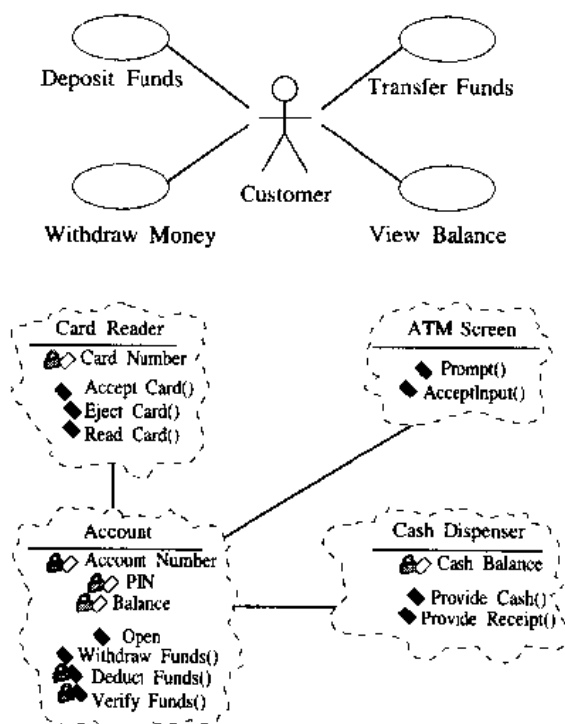


图1.5 用Booch图注方法表示的对象和关系

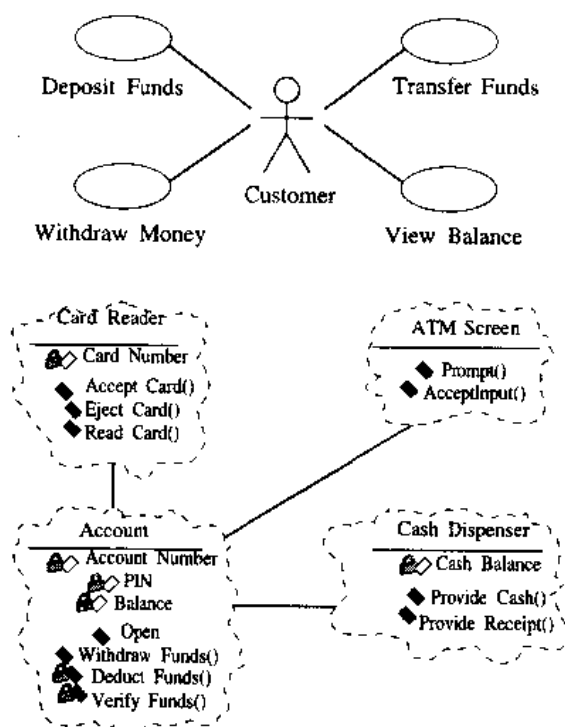


图1.6 用OMT图注方法表示的对象和关系

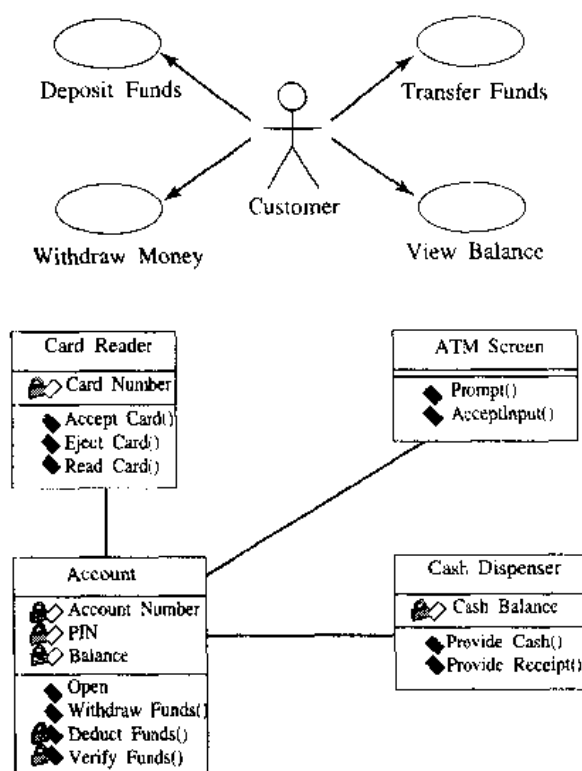


图1.7 UML图注的样本

1993年开始汇集UML的方法，三剑客Booch、Rumbaugh和Jacobson开始引入其他方法中的思想。直到1995年后期，正式推出了0.8版的Unified Method。1996年，经过改进和变革的Unified Method成为统一建模语言Unified Modeling Language。1997年UML 1.0经过认可并提交给对象技术组织（Object Technology Group），许多软件开发公司开始采用UML。最后，1997年11月14日，OMG将UML 1.1作为行业标准。

UML框图

利用UML可以开发几种不同的可视框图，表示系统的不同方面。Rational Rose支持开发这些模型的大部分，包括：

- Use Case框图
- Sequence框图
- Collaboration框图
- Class框图
- State Transition框图
- Component框图
- Deployment框图

这些模型框图表示系统的不同方面。例如，Collaboration框图显示对象间为完成某种系统功能所需要的交互。每个框图有一定的用途和使用对象。

Use Case框图

Use Case框图显示使用案例（表示系统功能）与角色（表示提供或接收系统信息的个人或系统）间的交互。图1.8是自动柜员机（ATM）系统的Use Case框图。

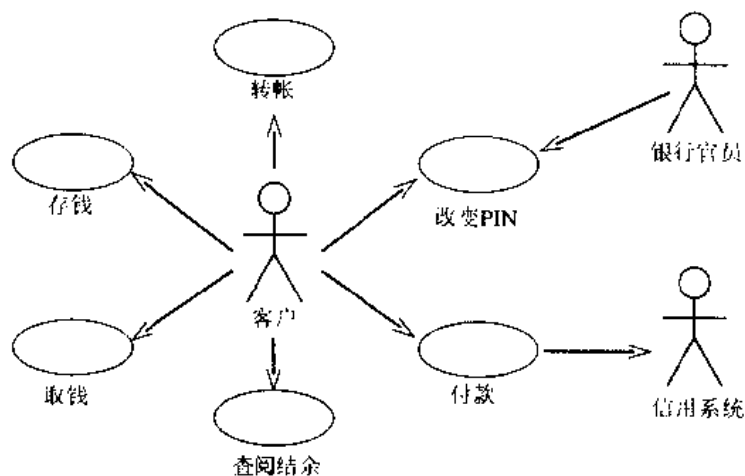


图1.8 自动柜员机（ATM）系统的Use Case框图

Use Case框图显示使用案例与角色间的交互。使用案例表示从用户角度对系统的要求，因此表示系统功能。角色是系统的主体。这些框图显示哪个角色启动使用案例，并显示角色何时从使用案例收到信息。这个Use Case框图显示自动柜员机（ATM）系统使用案例与角色间的交互。实际上，Use Case框图可以演示系统的要求。本例中，银行客户启动几个使用案例：取钱、存钱、转帐、付款、查阅结余和改变PIN。其中几个关系值得继续讨论。银行官员也可以启动改变PIN使用案例。由付款到信用系统有一个箭头。外部系统可能是角色，这里信用系统是个角色，因为它是在ATM系统之外的。箭头从使用案例到角色要表示使用案例产生一些角色要使用的信息。这里的付款使用案例向信用系统提供信用卡付款信息。

大部分信息可以从Use Case框图看到。这个框图显示系统的总体功能。用户、项目管理员、分析人员、开发人员、质量保证工程师和任何对系统感兴趣的人都可以浏览这个框图，了解系统的功能。

Sequence框图

Sequence框图显示取钱使用案例中的功能流程。例如，取钱使用案例有几个可能的程序，如取美元、想取而没钱、想取而PIN错等等。取20美元的正常情形（没有想取而没钱或想取而PIN错等问题）如图1.9。

这个Sequence框图显示了取钱使用案例的过程流程。框图顶部显示了涉及的角色，上例中显示客户角色。系统完成取钱使用案例所需的对象也在框图顶部显示。每个箭头表示角色与对象或对象与对象之间为完成所需功能而传递的消息。关于Sequence框图要说明的另一点是，它只显示对象而不显示类，类表示对象的类型，见第5章介绍。对象是特定的，Sequence框图不只显示客户，而且显示Joe。

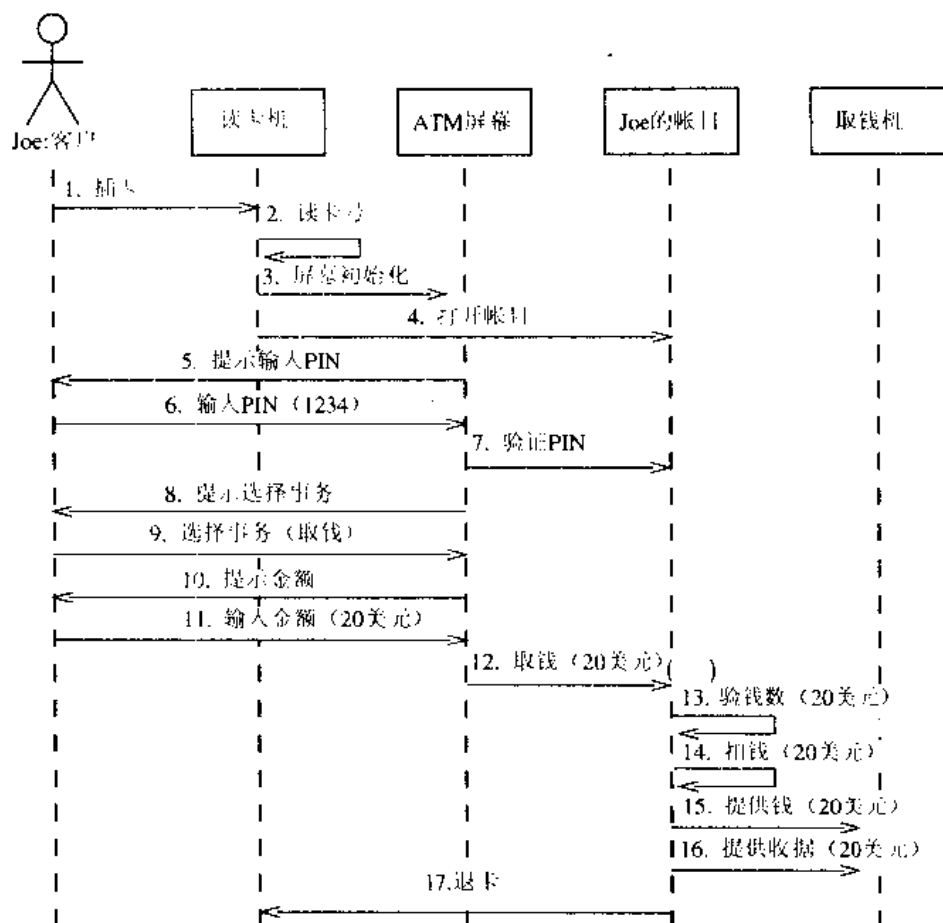


图1.9 Joe取20美元的Sequence框图

取钱使用案例从用户将卡插入读卡机开始，读卡机对象表示为框图顶部的矩形。然后读卡机读卡号，打开Joe的帐目对象，并初始化ATM屏幕。屏幕提示输入PIN，Joe输入PIN（1234），然后屏幕验证PIN与帐目对象，发出其相符的信息。屏幕向Joe提供选项，Joe选择取钱。然后屏幕提示Joe输入金额，他选择20美元。然后屏幕从帐目中取钱，启动一系列帐目对象要完成的过程。首先，验证帐目中至少有20美元。然后，它从中扣掉20美元，再让取钱机提供20美元现金。Joe的帐目还让取钱机提供收据。最后，它让读卡机退卡。

这样，Sequence框图用Joe取20美元的例子演示了取钱使用案例的全过程。用户可以从这个框图看到业务过程的细节。分析人员从Sequence框图可以看到处理流程。开发人员看到需要开发的对象和这些对象的操作。质量保证工程师可以看到过程的细节，并根据这个过程开发测试案例。Sequence框图对项目的各方面人员都有用。

Collaboration框图

Collaboration框图显示的信息与Sequence框图相同，但Collaboration框图用不同方式显示这个信息，具有不同作用。图1.9的Sequence框图对应图1.10的Collaboration框图。

在这个Collaboration框图中，对象表示为矩形，角色用简图表示。Sequence框图演示的是对象与角色随时间变化的交互，而Collaboration框图则不参照时间而显示对象与角色的交互。

互。例如，本框图中我们看到读卡机让Joe的帐目打开，Joe的帐目让读卡机退卡。另外，直接相互通信的对象之间画一条直线。如果ATM屏幕与读卡机直接相互通信，则其间画一条线。没有画线的对象之间不直接通信。

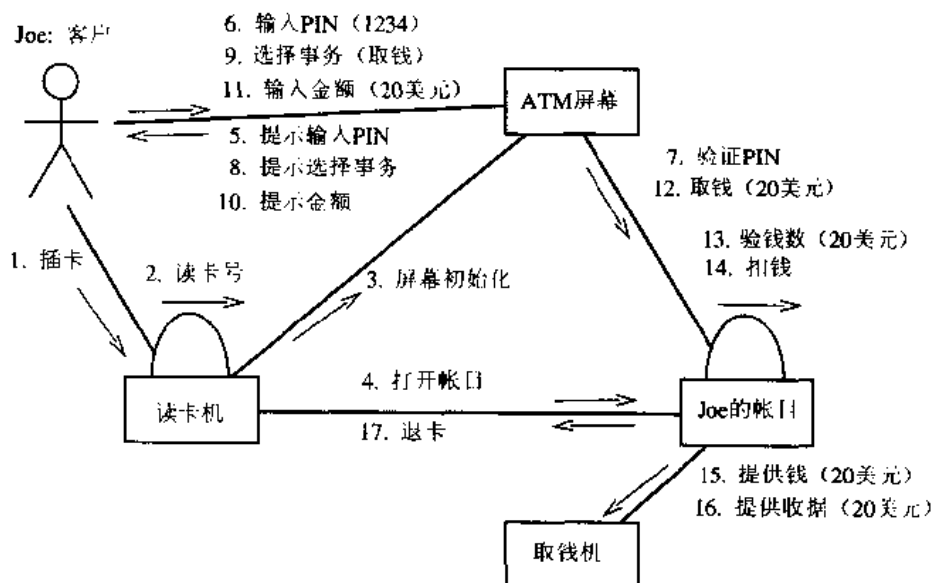


图1.10 Joe取20美元的Collaboration框图

因此，Collaboration框图显示的信息与Sequence框图相同，但Collaboration框图具有不同的作用。质量保证工程师和系统建筑师用Collaboration框图显示对象间处理过程的分布。假设Collaboration框图是星形的，几个对象与一个中央对象通信，则系统建筑师可能认为系统对中央对象依赖太强，系统建筑师可能重新设计对象，更均匀地分配处理工作。这种交互是很难在Sequence框图中看到的。

Class框图

Class框图显示系统中类与类之间的交互。类是对象的设计图，见第5章。例如，Joe的帐目是个对象。帐目是Joe的支票帐目设计图，是一个类。类包含信息和处理信息的功能。帐目类包含客户PIN和检查PIN的功能。Class框图中的类是对Sequence框图或Collaboration框图中每种对象生成的。图1.11演示了取钱使用案例的Class框图。

上述Class框图显示了实现取钱使用案例中类之间的关系。这是用四个类完成的：读卡机、帐目、ATM屏幕和取钱机。Class框图中每个类用方框表示，分成三部分。第一部分是类名。第二部分是类包含的属性，属性是与类相关联的信息。例如，帐目类包含三个属性：帐号、PIN和结余。最后一部分包含类的操作，操作就是类提供的功能。帐目类包含四个操作：打开、取钱、扣钱和验钱数。

连接类的直线显示类之间的关系。例如，帐目类连接ATM屏幕类，因为两者要直接相互通信。读卡机与取钱机不连接，因为两者不进行通信。另一个有趣之处是有些属性和操作的左边有小锁图标。锁头表示专用属性和操作。专用属性和操作只能在包含该专用属性和操作的类中访问。帐号、PIN和结余是帐目类的专用属性，而扣钱和验钱数是帐目类的专用操作。

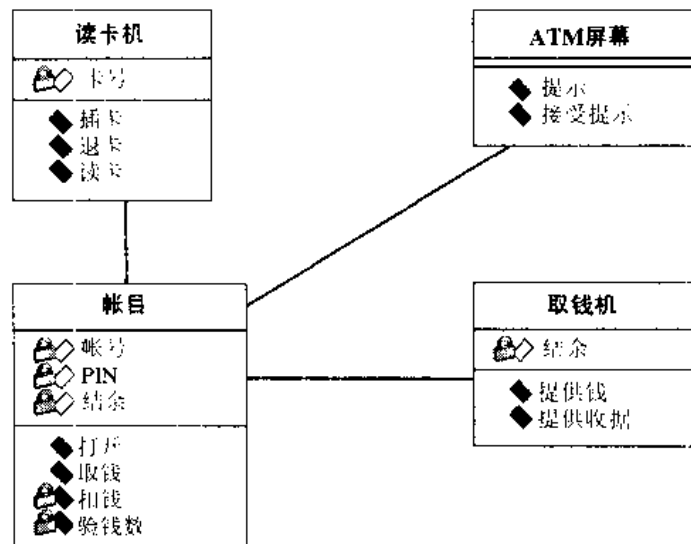


图1.11 ATM系统的取钱使用案例的Class框图

开发人员用Class框图开发类。**Rose**之类的工具产生类的框架代码，然后开发人员用所选语言填充代码细节。分析人员用Class框图显示系统细节。建筑师也用Class框图显示系统设计。如果一个类包含太多功能，则建筑师可以从这个Class框图中看出问题并将功能划分到多个类中。如果要相互通信的类之间没有建立关系，建筑师和开发人员也能从Class框图中看出。Class框图可以显示每个使用案例中类的相互作用，也可以显示整个系统或子系统。

State Transition框图

State Transition框图提供了建模对象各种状态的方式。Class框图提供了类及其相互关系的静态图形，而State Transition框图则可以建模系统的动态功能。

State Transition框图显示对象的功能。例如，银行帐户可能有几种不同状态，可以打开、关闭或透支。帐户在不同状态下的功能是不同的。State Transition框图可以显示这个信息。

图1.12显示了银行中帐户的State Transition框图。

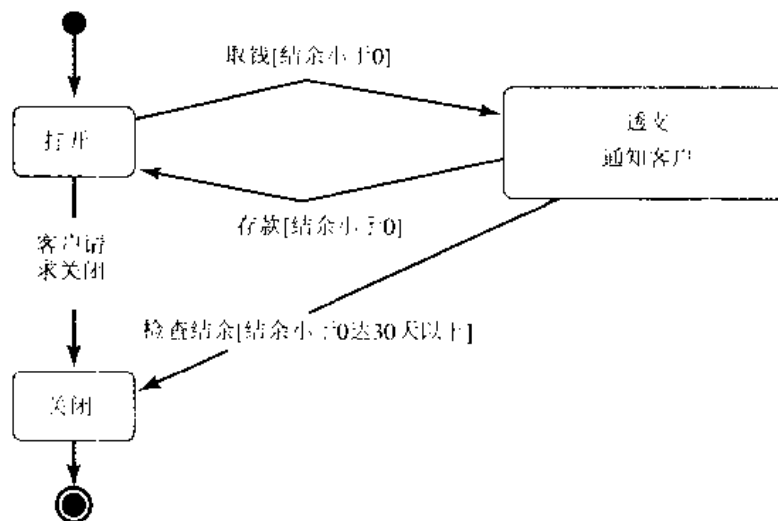


图1.12 银行中帐户的State Transition框图

在这个框图中，我们看到了银行帐目的不同状态，并看到了帐目如何从一种状态变到另一种状态。例如，帐目打开而客户请求关闭帐目时，帐目转入关闭状态。客户请求是事件，事件导致一种状态过渡到另一种状态。

如果帐目打开而客户要取钱，则帐目可能转入透支状态，这发生在帐目结余小于0时，框图中显示为[结余<0]。方括号中的条件称为保证条件，控制过渡能不能发生。

有两种特殊状态——开始状态和停止状态。开始状态在框图中用黑点表示，显示对象首次生成的状态。停止状态用牛眼表示，显示对象删除之前的状态。在State Transition框图中，只有一个开始状态，可以没有停止状态，也可以有多个停止状态。

对象处于特定状态时，可能发生某种事件。例如，帐目透支时，要通知客户。对象处于特定状态时发生的过程称为操作。

State Transition框图不是对每个类生成，只用于复杂的类。如果类对象有多种状态，每种状态中的表现又大不相同，则可能要对其生成State Transition框图。许多项目根本不需要这种框图。如果生成这种框图，则开发人开发类时可以使用这种框图。

State Transition框图仅用于文档。从Rose模型产生代码时，没有任何代码是从State Transition框图的信息产生。但Rose插入件可以在实时系统中根据State Transition框图产生执行代码。

Component框图

Component框图显示模型的物理视图。Component框图显示系统中的软件组件及其相互关系。框图中有两种组件：执行组件和代码库。

在Rose中，模型中的每个类映射到源代码组件。一旦生成组件，就加进Component框图中，然后画出组件间的相关性。组件间的相关性包括编译相关性和运行相关性。

图1.13是ATM系统的Component框图。

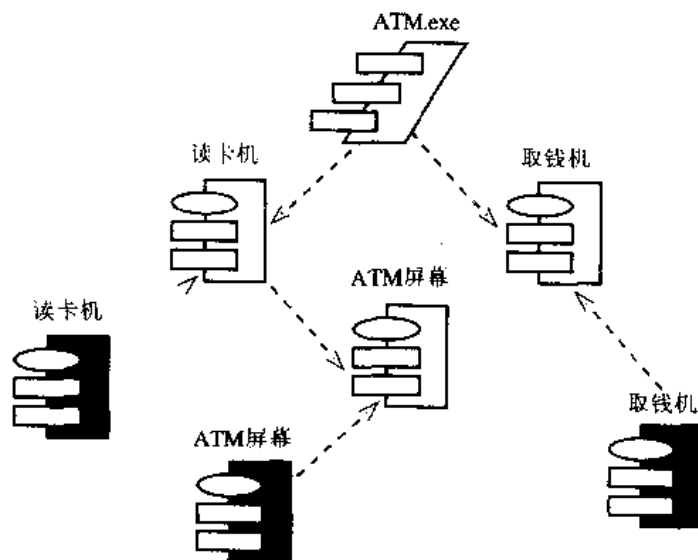


图1.13 ATM系统的Component框图

这个Component框图显示ATM系统中的客户机组件。这种情况下，小组决定用C++建立系统。每个类有自己的.CPP和头文件，因此框图中每个类映射自己的组件。例如，ATM屏幕类映射ATM屏幕组件。ATM屏幕类还映射第二个ATM屏幕组件。这两个ATM屏幕组件表示ATM屏幕类的头和体。阴影组件称为包规范（package specification），表示C++中ATM屏幕类的头文件（.H）。无阴影组件也称为包规范，这个包规范表示C++类的头文件（.H）。组件ATM.exe是个任务规范，表示处理线程。这里的处理线程是个可执行程序。

组件用虚线连接，表示组件间的相关性。例如，读卡机类与ATM屏幕类相关，即必须有ATM屏幕类才能编译读卡机类。编译所有类之后，即可生成执行文件ATMClient.exe。

ATM例子有两个处理线程，因而有两个执行文件。一个执行文件是ATM客户机，包括取钱机、读卡机和ATM屏幕。一个执行文件是ATM服务器，包括帐目组件。图11.4显示了ATM服务器的Component框图。

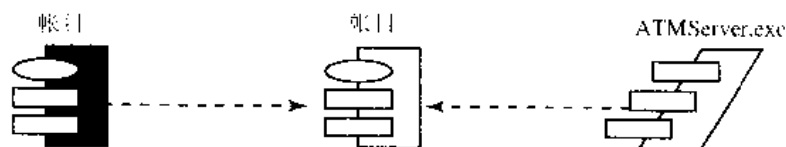


图1.14 ATM服务器的Component框图

从上例可能看出，根据子系统 and 执行文件的个数，系统可以有多个Component框图。每个子系统是一个组件包。一般来说，包是对象的集合。这里，包是组件的集合。ATM例子有两个包：ATM Client和ATM Server。第3章“Rose基础”部分将详细介绍包。

编译系统的人员要使用Component框图。Component框图显示组件应以什么顺序编译，框图还显示编译时会生成哪些运行组件。Component框图显示类与实现组件之间的映射。这些框图还启动代码生成。

Deployment框图

Deployment框图是我们要介绍的最后一种框图。Deployment框图显示网络的物理布局和各种组件的位置。在ATM例子中，ATM在不同设备（或节点）上运行许多组件。图1.15显示了ATM系统的Deployment框图。

这个Deployment框图显示了系统的主要布局。ATM客户机执行文件在不同地点的多个ATM上运行。ATM客户机通过专用网与地区ATM服务器通信。ATM服务器执行文件在地区ATM服务器上执行。地区ATM服务器又通过局域网与运行Oracle的银行数据库服务器通信。最后，打印机与地区ATM服务器连接。

因此，这个框图显示了系统的物理设置。ATM系统采用三层结构，分别针对数据库、地区ATM服务器和客户机。

项目管理员、用户、建筑师和部署人员通过Deployment框图了解网络的物理布局和各种组件的位置。项目管理员通过这个框图与用户沟通系统的布局。部署人员用Deployment框图进行部署规划。

所有这些框图从几个方面描述系统。在第3章“Rose基础”部分，我们将更详细地介绍每个框图，介绍其在Rational Rose中如何产生。我们还将Rational Rose中练习生成和使用

用这些框图。介绍Rose的细节之前，还要介绍软件开发项目的另一方面——过程。尽管本书不是方法论或过程方面的书籍，但我们希望读者熟悉利用UML框图进行开发的过程。

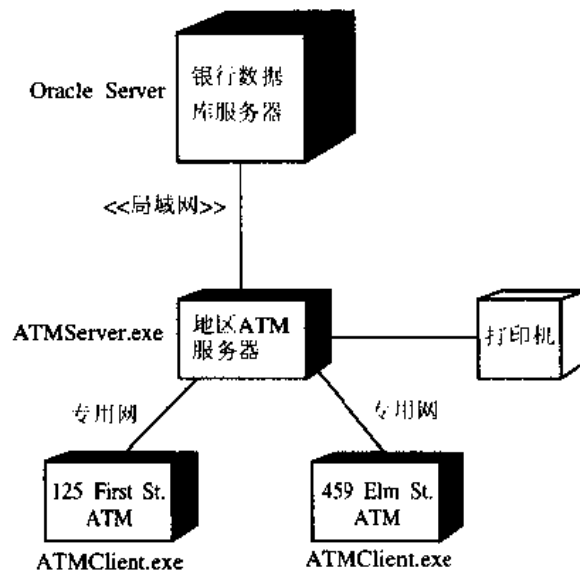


图1.15 ATM系统的Deployment框图

可视化建模与软件开发过程

软件开发可以采用多种途径。项目遵循不同类型的开发过程，包括流水式过程和面向对象过程等，各有利弊。本节不打算介绍要用哪个过程，但会介绍针对可视化建模的过程，这只是一个概述。

长期以来，软件开发采用流水式过程。在这种模型中，我们要分析需求、设计系统、开发系统、测试系统和部署系统。显然，这个过程是不可逆的，水不能倒流。这个方法已经在成千万个项目中采用，但用得并不那么纯粹。流水式过程的一个主要缺点是要一步步回溯。在流水式过程中，我们要先确定所有系统要求，为此要与用户认真讨论，详细研究业务过程，然后要与用户敲定我们所列出的大量要求，即使他们还没有阅读这些要求。如果运气好，这个分析阶段可能确定了80%的要求。

然后就要设计了。我们坐下来，确定系统的结构。我们要确定程序放在哪里、要用什么硬件才能得到满意的性能。在这个过程中，可能会出现一些新问题。然后我们又要回过头与用户讨论这些问题，从而引出新的要求，又要重新进行分析。这样来回几次后，我们转入开发阶段，开始编写系统代码。

编码过程中，我们发现有些系统设计决策是无法实现的。因此要退回设计阶段，重新研究问题。编码完成后，要开始测试。测试过程中，我们发现要求不够详尽，解释有不当之处。这样又要回到分析阶段重新研究要求。

经过一段时间，我们终于做完这个系统，要交给用户。由于经过了较长时间，公司业务已经在这个过程中发生了变化，用户不太热情地说：“这是我原先要的东西，但现在又满足不了要求了”。这么一折腾，项目可能就要折腾上十年了。

看看这种情形，你会怀疑自己干这行到底对不对，能不能有更好的办法？是业务变化太快？是用户沟通太少？是用户不了解项目小组？是小组没有遵循某个过程？这很难讲。业务变化很快，我们作为软件专业人员必须赶上。用户沟通太少是因为他们已经习以为常。问一个三十年工龄的会计人员就像问你如何洗澡，太习惯了，很难描述。另一个问题是用户不了解项目小组？项目小组提供了流程图，建立了大量要求文本，但用户并不一定能理解。有没有办法呢？可以用可视化建模。最后，小组用的是流水线过程，可惜方法的规划与执行是两回事。

因此，一个问题是小组计划使用流水式过程，它按顺序整齐地经过项目的各个阶段，但需要回溯整个项目。是否因为规则不好？可能不是。软件开发是个复杂的过程，每个阶段要彻底解决相关问题是不可可能的。如果忽略回溯的要求，则系统会有设计缺陷，遗漏要求或者更糟的问题。但多年来我们已经学会了回溯，学会了重复性开发。重复性开发就是一遍一遍来。在面向对象过程中，我们多次进行分析、设计、开发、测试和部署的子阶段。

要在项目早期了解所有要求是不可能的。新事件不断出现，因此要通过重复规划。有了这个概念，就可以把项目看成一系列小瀑布，每个小瀑布都标志项目某个重要部分的完成，又能充分减少回溯。在项目中，我们要经过四个阶段：开始、细化、构造和交接。开始就是项目的开头，要收集信息和进行概念验证。开始阶段得出项目的行/不行决策。细化阶段要细化使用案例和作出结构性决策。细化包括分析、设计、编码和测试文件。而构造阶段则是进行大量编码。交接是系统向用户最后准备和部署的阶段。下面要介绍这四个阶段在面向对象项目中的意义。

开始

开始阶段就是项目的开头。开始阶段一开头，有人说“要是有一个这样的系统该多好”，然后有人开始研究这个思路，管理人员询问要多长时间、多少经费、项目是否可行。开始阶段要回答这些问题。我们发现项目的高层特性并建成文档，确定系统中的角色和使用案例。这里并不详细研究使用案例，只是提供一两句说明。我们还向主管人员提供一个估计。因此，如果用Rose支持项目，则生成角色、使用案例和产生使用案例框图。研究完成后，管理人员同意提供项目所需资源时，开始阶段即告完成。

项目开始阶段主要是顺序性的，而不是重复性的。其他阶段则在项目中重复多项。由于项目实际上只能开始一次，因此项目中的开始阶段只进行一次。为此，开始阶段还有另一任务：开发重复计划。重复计划描述每次要重复实现的使用案例。如果开始阶段发现四个使用案例，则可以开发重复计划如下：

- 第一次 使用案例 1, 5, 6
- 第二次 使用案例 7, 9
- 第三次 使用案例 2, 4, 8
- 第四次 使用案例 3, 10

这个计划说明首先要干什么。确定这个计划要求考虑使用案例之间的相关性，相应作出计划。如果必须有使用案例3才能让使用案例5工作，则上述计划行不通，因为使用案例3要到第四次循环时才能实现，而使用案例5则在第一次循环时要实现。因此可能要根据相关性调整计划。

开始阶段使用Rose

开始阶段要确定角色和使用案例。Rose可以建描述这些使用案例和角色，可以生成显示其关系的框图。使用案例框图可以让用户验证这个框图确实显示了系统特性。

细化

项目的细化阶段包括计划、分析和结构性设计。完成重复计划后，要对当前循环的每个使用案例进行细化。细化包括项目的几个方面如编写概念验证、开发测试案例和作出设计决策。

细化阶段的主要任务是细化使用案例。第3章“Rose基础”部分会介绍使用案例的细节。使用案例的低层要求包括使用案例的处理流程，使用案例中涉及的角色，显示处理流程的交互框图和使用案例的状态细节，以及使用案例期间可能发生的状态改变。详细使用案例形式的要求放进软件要求规范SRS中。SRS包含所有系统要求细节。

细化阶段还有其他任务，如初期估计、审查SRS和使用案例的质量、风险调查。Rational Rose可以帮助细化使用案例模型和生成程序框图与合作框图，显示图形化的处理流程。类框图显示要建立的对象，也是在细化阶段设计。

使用案例完全细化并被用户接受后，完成概念验证以减少风险并完成类框图，细化阶段即告完成。换句话说，系统设计、审查并可以让开发人员建立时，细化阶段即告完成。

细化阶段使用Rose

细化阶段有几处可以使用Rational Rose。由于细化阶段是细化系统要求，因此不能接受新使用案例模型。在细化阶段时，需以程序框图与合作框图。它们还有助于设计系统所需的对象。细化要求及系统的设计准备，以便开发人员能开始开发。这可以通过Rose生成类框图与状态过渡框图实现。

构造

构造指软件开发和测试的过程。和细化阶段一样，构造阶段也是对当前循环的每个使用案例进行。构造阶段的任务包括确定余下的要求、开发软件和测试软件。由于细化阶段的软件设计已经完成，构造时不应涉及太多设计决策。这样，项目小组就可以并行开发。并行开发就是多个开发人员可以开发软件的不同对象，直到整个系统最终能集成起来。在细化阶段，我们要设计系统中的对象及其交互方式。构造阶段只是将设计付诸实施，而不能进行新的设计决策，以免改变交互方式。自下而上建立系统模型的另一个好处是Rational Rose可以产生系统的框架代码。要使用这个特性，就要在构造阶段早期生成组件和组件框图。生成组件并画出组件间的相关性后，就可以开始产生代码。代码生成根据设计提供尽可能多的代码，但并不是说Rose能产生业务特定代码。产生的代码取决于所选的语言，但通常包括类声明、属性声明、范围声明、函数原型和继承语句。这样可以节省时间，因为编写这些代码是很繁琐的。产生代码后，开发人员即可关注项目的特定业务方面。完成代码时，要让同级开发人员检查，保证其符合标准、设计规则和可行。代码审查后，应对对象进行质量保证检查。如

果构造期间增加了新属性或函数，或对象间的交互有所改变，则应通过逆向转出工程代码在Rose模型中更新代码。第19章到第24章将详细介绍这个问题。

软件完成并测试之后，构造阶段即告结束。一定要保证软件与模型同步，软件进入维护方式时，模型非常重要。

构造阶段使用Rose

构造阶段要完成项目的大部分代码。Rose可以根据对象设计生成组件。组件框图显示组件间的翻译相关性。选择每个组件的语言后，可以产生框架代码。开发人员生成代码后，可以通过逆向转出工程代码让软件与模型同步。

交接

交接阶段将完成的软件产品交给用户。这个阶段的任务包括完成最后软件产品、最后接受测试、完成用户文档、准备用户培训。软件要求规范要根据最终改变进行更新。一定要让软件与模型同步，因为软件进入维护方式时，模型非常重要。完成项目几个月后，模型对改进软件非常有用。

Rational Rose在交接阶段的作用不大。这时软件产品已经开发。Rose是用于辅助建模和软件开发的，甚至帮助软件部署。但Rose不是测试工具，无法帮助测试计划和部署过程。这些工作可以用其他工具完成。因此，Rose在交接阶段主要用于在软件产品完成时更新模型。

小结

可视化建模和Rational Rose在软件开发过程的几个阶段很有用。在项目开始阶段，Rose可以产生使用案例模型。在细化阶段，Rose可以开发程序框图和合作框图，显示要开发的对象，及其相互间的交互。Rose开发的类框图显示对象间的相互关系。在构造初始阶段，用Rose生成组件框图，显示系统组件间的相关性，并可以产生系统的框架代码。在构造阶段，Rose可以将新开发代码逆向转出工程代码到模型中，将开发阶段出现的变化反映在模型中。构造之后，进入交接阶段，Rose主要用于在软件产品完成时更新模型。

下一章要简单介绍Rose工具的不同特性和功能，Rose中的菜单选项，并介绍如何生成和保存Rose模型。我们将介绍如何漫游Rose，Rose提供的四个模型视图和如何在Web上发表Rose模型。

第2章 Rose之游

- 安装Rose 98
- 安装Rose 98i
- Rose漫游
- Rose模型中的四个视图
- 使用Rose
- 设置全局选项

本章要介绍Rational Rose产品。首先介绍Rational Rose的内容和Rational Rose模型包括的内容, 然后介绍如何在计算机上安装Rose, 介绍可视漫游, 介绍屏幕的各个部分和如何在Rose中移动。最后介绍Rose模型中的四个视图及如何使用Rose设计系统。

何谓Rose

Rational Rose是分析和设计面向对象软件系统的强大工具, 可以帮助先建模系统再编写代码, 从而一开始就保证系统结构合理。利用模型可以更方便地捕获设计缺陷, 从而以较低的成本修正这些缺陷。

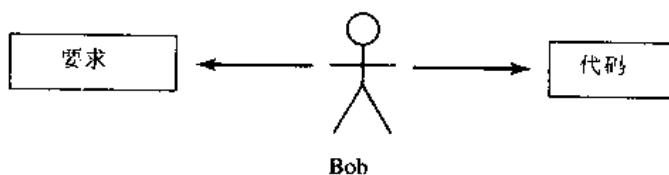
Rational Rose有助于系统分析, 可以先设计使用案例和Use Case框图, 显示系统的功能。可以用Interaction框图显示对象如何配合, 提供所需功能。类和Class框图可以显示系统中的对象及其相互关系。Component框图可以演示类如何映射到实现组件。最后, Deployment框图可以显示系统的网络设计。

Rose模型是系统的图形, 包括所有UML框图、角色、使用案例、对象、类、组件和部署节点。它详细描述系统的内容和工作方法, 开发人员可以用模型作为所建系统的蓝图。

这样就避免了一个老问题: 小组要与客户交流并记录其要求。现在开发人员可以准备编码了。一个开发人员(假设为Bob)取得一些要求, 作出一些设计决策, 编写一些代码。而Jane也取得一些要求, 作出完全不同的设计决策, 再编写一些代码。

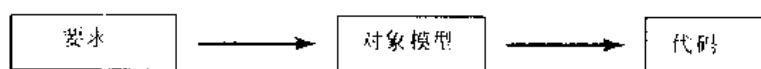
这种编程风格的差别非常自然, 20个开发人员对同样的要求可能开发出20种不同系统。这样, 有人要了解或维护系统时就会遇到问题。如果不详细与每个开发人员面谈, 就很难了解他们作出的开发决策, 系统各部分的作用和系统的总体结构。如果没有设计文档, 则很难保证你建的系统就是用户所要的系统。

传统上, 我们采用如下过程:



用户的要求被建成文档，但设计在Bob的脑子里，因此只有Bob知道系统的结构。如果Bob离开，则这个信息也随他一起离开。如果你代替Bob，则你会知道要了解一个文档不足的系统有多么费事。

Rose模型采用的过程如下：



设计被建成文档，开发人员就可以在编码之前在一起讨论设计决策了，不必担心系统设计中每个人选不同方向。

但这个模型不仅为开发人员使用：

- 客户和项目经理用Use Case框图取得系统的高级视图，确定项目范围。
- 项目经理用Use Case框图和文档将项目分解成可管理的小块。
- 分析人员和客户用使用案例文档了解系统提供的功能。
- 技术作者用使用案例文档开始编写用户手册和培训计划。
- 分析人员和开发人员用Sequence和Collaboration框图了解系统的逻辑流程、系统中的对象及对象间的消息。
- 质量保证人员用使用案例文档和Sequence、Collaboration框图取得测试脚本所需的信息。
- 开发人员用Class框图和State Transition框图取得系统各部分的细节及其相互关系的信息。
- 部署人员用Component和Deployment框图显示要生成的执行文件、DLL文件和其他组件以及这些组件在网络上的部署位置。
- 整个小组用模型来确保代码中遵循了要求，代码可以回溯到要求。

因此，Rose是整个项目组使用的工具，是每个小组成员可以收集所要信息的范围和设计信息的仓库。除此之外，Rational Rose还可以帮助开发人员产生框架代码，适用于市面上的多种语言，包括C++、Java、Visual Basic和PowerBuilder。此外，Rose可以逆向转出工程代码，根据现有系统产生模型。根据现有系统产生模型的好处很多。模型发生改变时，Rose可以修改代码，作出相应改变。代码发生改变时，Rose可以自动将这个改变加进模型中。这些特性保证模型与代码同步，避免遇到过时模型。

利用RoseScript可以扩展Rose，这是Rose随带的编程语言。利用RoseScript可以编写代码自动改变模型、生成报表、完成Rose模型的其他任务。

目前Rose有三种版本：

- Rose Modeler，可以对系统生成模型，但不支持代码产生和逆向转出工程代码。
- Rose Professional，可以用一种语言生成代码。
- Rose Enterprise，可以用C++、Java和Visual Basic以及Oracle8结构生成代码。

此外，Rose 98i是Rose的最新改进，能将Rose与其他工具集成，如Rational RequisitePro、TeamTest、Visual C++等等。Rose 98i还可以在Web上发表模型。Rose 98i与Rose 98相似，有Modeler、Professional和Enterprise版本。本书选配光盘中有本书所有练习的Rose 98与Rose 98i版本。

安装Rose 98

Rational Rose可以从光盘安装到Windows 95、NT或Windows 98上。安装开始时，将光盘插入计算机。安装程序自动启动，但如果你的计算机不支持光盘文件自动启动，则要从光盘根目录中运行setup.exe文件。安装过程开始后，出现图2.1所示屏幕。

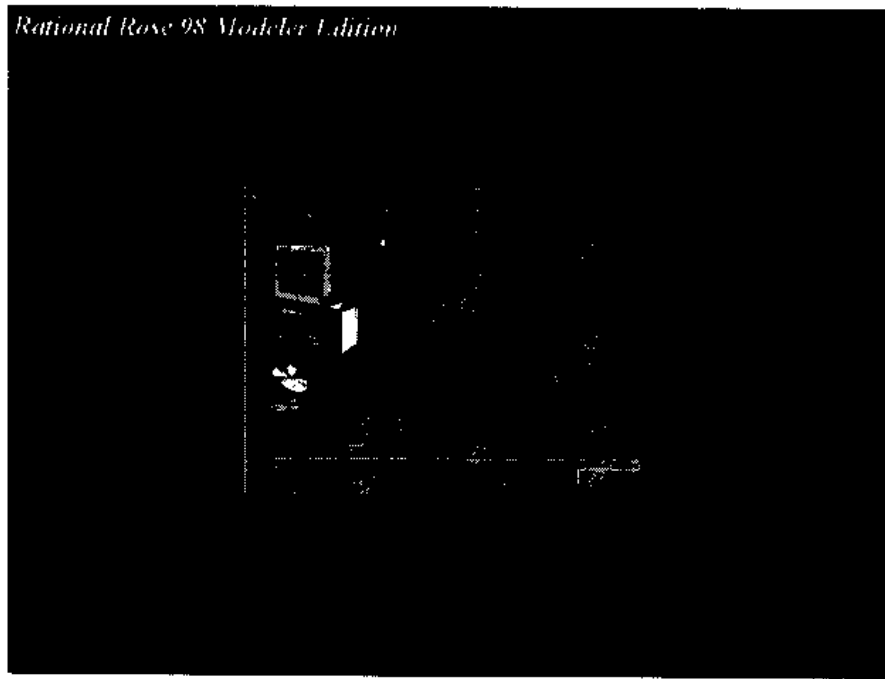


图2.1 欢迎安装Rose

安装时建议退出所有其他Windows应用程序。退出所有其他Windows应用程序后，单击Next继续。

然后，如图2.2所示，安装程序建议删除计算机上安装的旧版Rational Rose。用Windows控制面板中的添加/删除程序删除旧版本，然后单击Next继续。

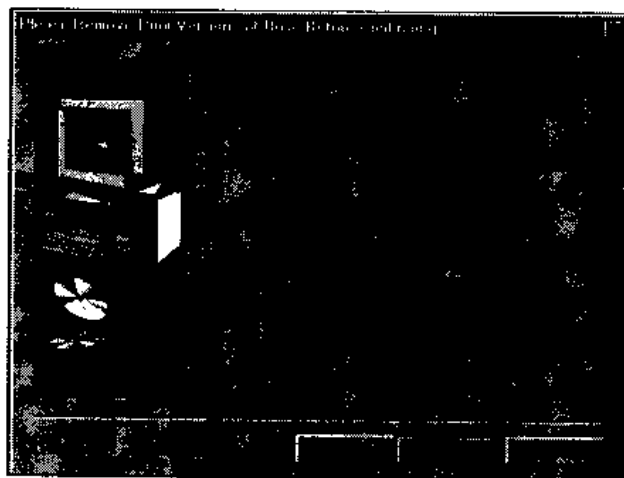


图2.2 删除计算机上安装的旧版Rational Rose

然后出现图2.3所示的许可证协议。阅读许可证协议后,按**Yes**接受条件,否则退回软件包。

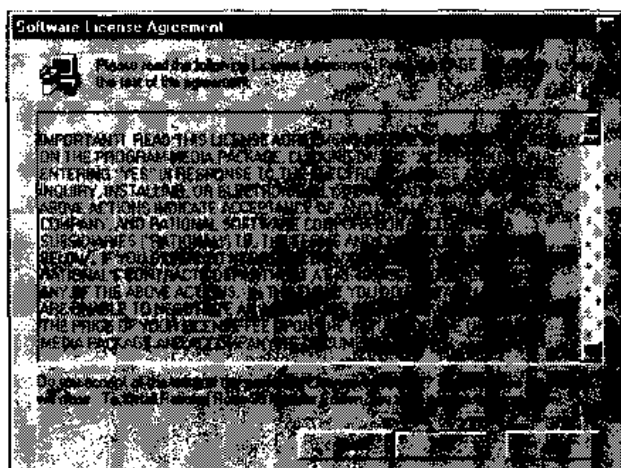


图2.3 许可证协议

下一个安装屏幕询问姓名、公司名和Rose序列号，如图2.4。必须要输入这三项之后才能继续，序列号见光盘盒背面。输入姓名、公司名和Rose序列号后，安装程序请求验证信息，如果正确则单击OK，如果序列号有效，则安装过程继续。

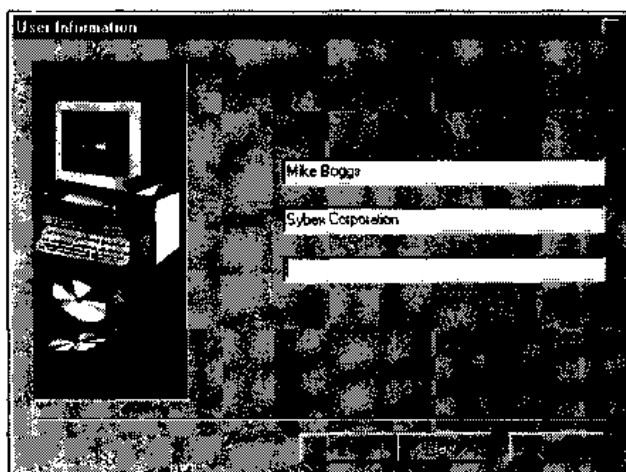


图2.4 用户信息

验证用户信息后，安装程序询问安装Rational Rose的地址，如图2.5。缺省地址为C:\Program Files\Rational\Rational Rose 98 Enterprise Edition。如果接受这个地址，单击Next继续。如果要改变地址，单击Browse按钮选择不同目录。

然后确定安装类型，选项见图2.6的Setup Type窗口。

选项包括Typical、Compact和Custom。Compact选项只安装Rose的Modeler组件和帮助文件，Typical选项安装Rose和常用插入件，包括：

- Microsoft Repository插入件
- SCC插入件

- Visual Basic插入件
- C++插入件
- Java插入件
- Oracle8插入件
- Framework插入件
- ERwin插入件
- TypeLibImporter插入件

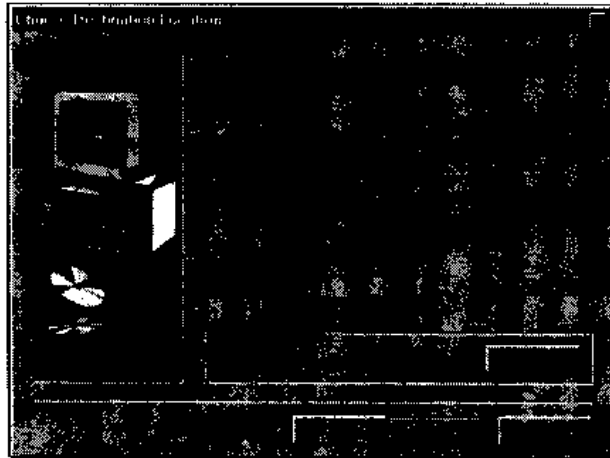


图2.5 选择目标地址

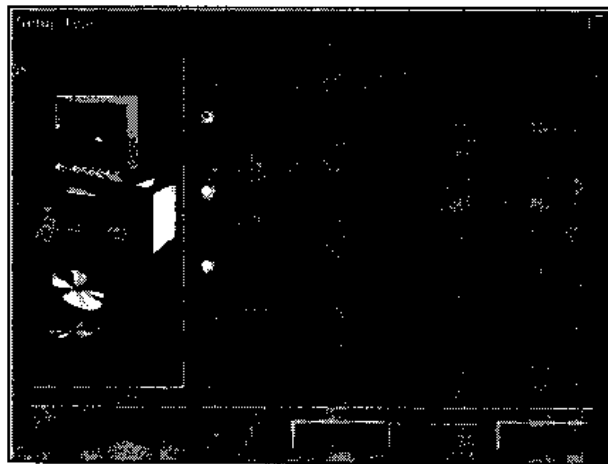


图2.6 确定安装类型

Custom选项可以选择要安装的插入件。建议选择**Typical**选项。本书的所有练习假设与**Typical**选项安装Rose。

指定选项后，单击**Next**继续。出现一个短消息，建议如果使用Microsoft Repository插入件，必须安装Visual Studio。

然后要选择图标程序文件夹，如图2.7缺省文件夹为Rational Rose 98 Enterprise Edition，也可以选择其他文件夹。选择之后，单击**Next**继续。

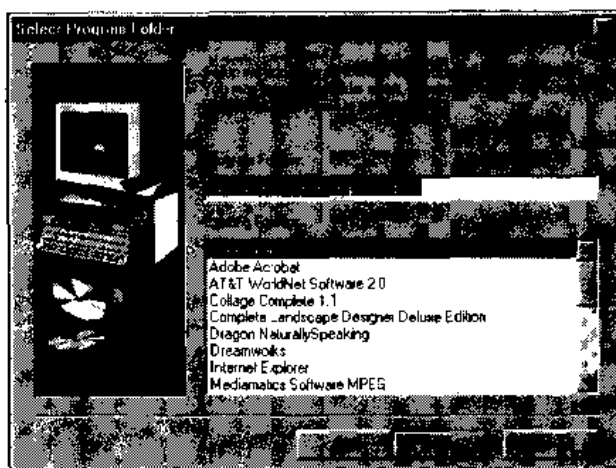


图2.7 选择图标程序文件夹

下一个要选的选项是缺省语言，如图2.8。

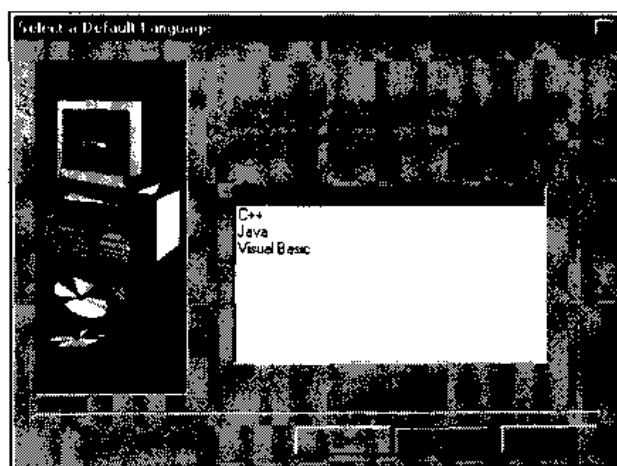


图2.8 选择缺省语言

启动Rose时，它配置所选语言的对象库。这些库可以用该语言产生代码或逆向转出工程代码。可以选择：

- Analysis
- C++
- Java
- Visual Basic

Analysis选项将Rose配置成没有缺省语言，其他选项将Rose配置成将该语言作为缺省语言，在模型中包括所需的对象库。例如，如果开发语言为Visual Basic，则选择缺省语言Visual Basic。开始新的Rose模型时，它自动包括Visual Basic对象、接口和组件。选择后，单击Next继续。

开始复制文件和配置Rose之前，还有一次机会检查所有设置，如图2.9。

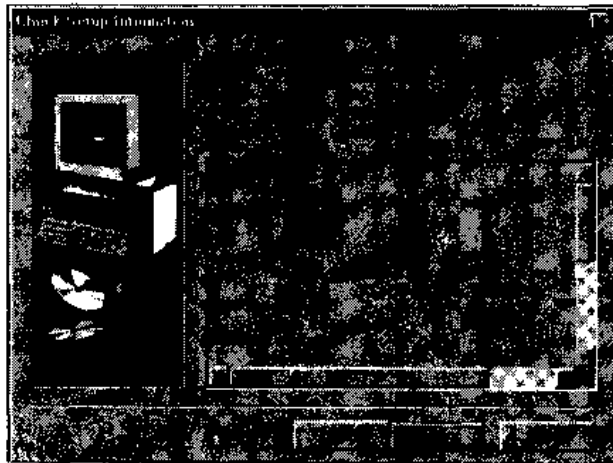


图2.9 检查设置信息

可以检查所需的磁盘空间，安装哪些组件，将文件安装在哪里，图标安装在哪个程序组中，选择哪个缺省语言。如果一切正确，单击Next继续，开始复制文件和配置Rose。

安装程序复制文件和配置Rose时，图2.10所示屏幕会显示安装进程。如果要取消安装，可以随时单击Cancel按钮。

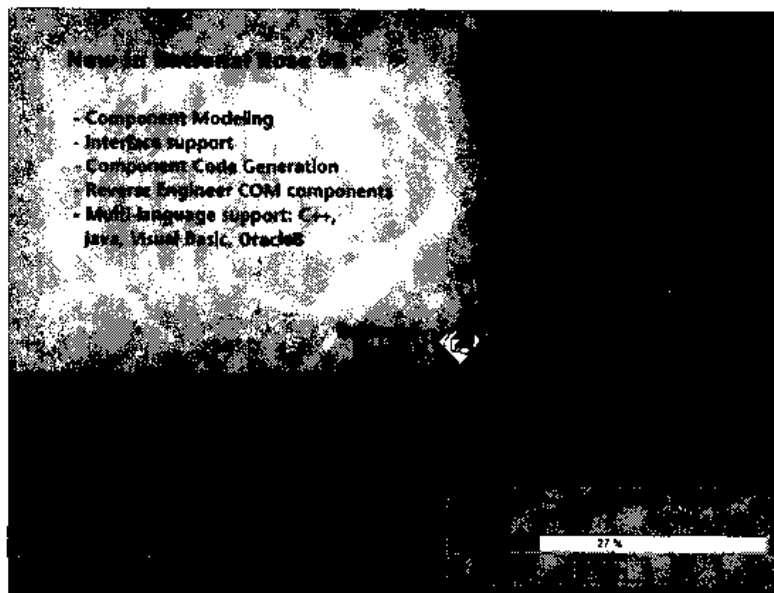


图2.10 复制文件

几分钟后，安装完毕。安装程序显示图2.11所示的屏幕，可以阅读Readme文件或启动Rose。如果要做其中一项工作，单击相应复选框，然后单击Finish按钮。

安装完毕，就可以用Rational Rose建立系统模型了。如果使用测试版，则有30天使用期，然后产品不再可用。注意，如果在计算机上重新安装或修改系统日期，这个产品将无法访问。要在30天测试期之后再使用Rational Rose，只能从授权经销商那里购买Rational Rose。



图2.11 阅读Readme文件或启动Rose

安装Rose 98i

Rational Rose 98i可以从Rational软件产品光盘安装到Windows 95、NT或Windows 98上。安装开始时，将光盘插入计算机。安装程序自动启动，但如果你的计算机不支持光盘文件自动启动，则要从光盘根目录中运行setup.exe文件。安装过程开始后，出现图2.12所示屏幕。

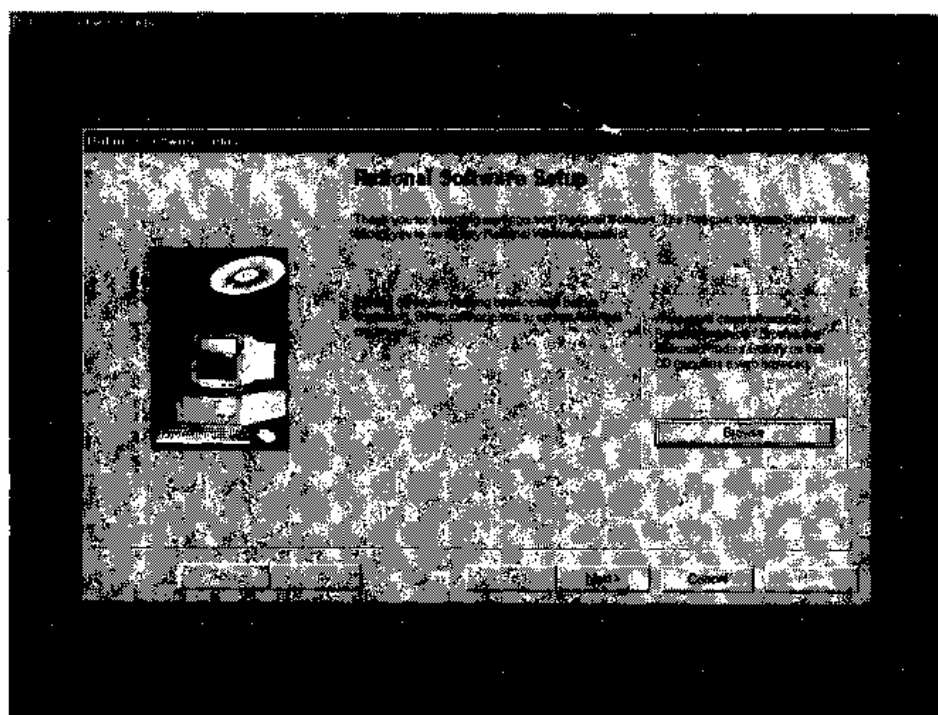


图2.12 Rational软件安装向导

安装时建议退出所有其他Windows应用程序。退出所有其他Windows应用程序后，单击Next继续。

如图2.13, 安装程序要求选择安装的产品。大部分Rational软件都来自光盘。选择Rational Rose 98i Enterprise、Professional或Modeler。

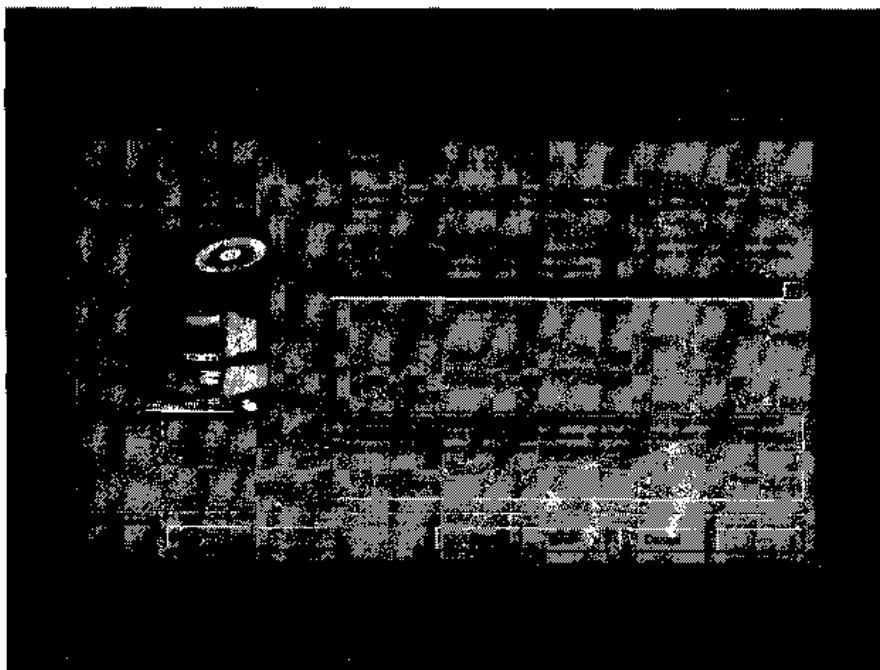


图2.13 选择安装的产品

还要选择安装Rose的位置, 缺省为C:\Program Files\Rational。如果接受这个地址, 单击Next继续。如果要改变地址, 单击Browse按钮选择不同目录。

然后出现图2.14所示的许可证协议。阅读许可证协议后, 按Yes接受条件, 否则退回软件包。

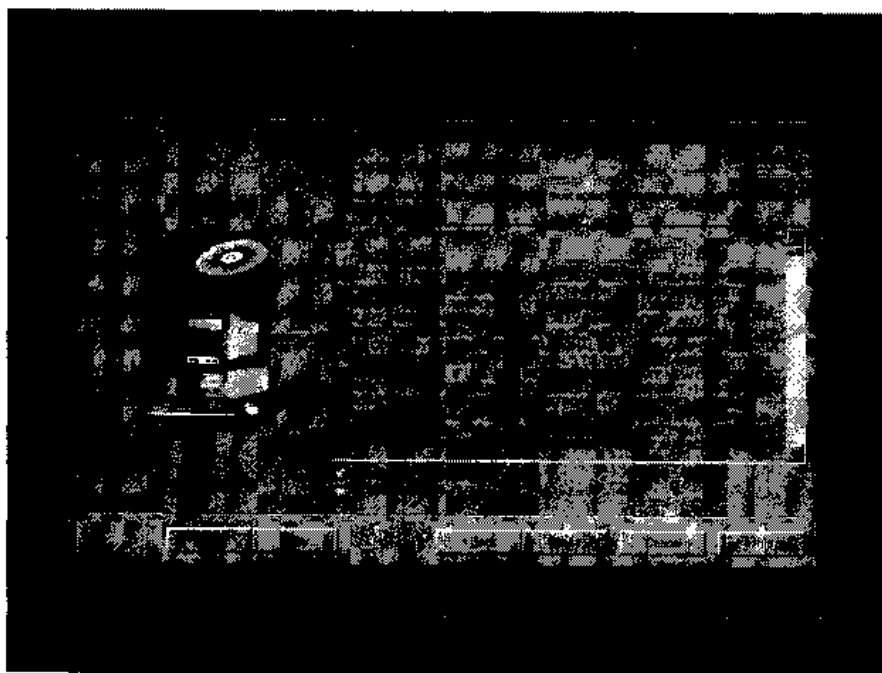


图2.14 许可证协议

然后确定安装类型，选项见图2.15的Setup窗口。

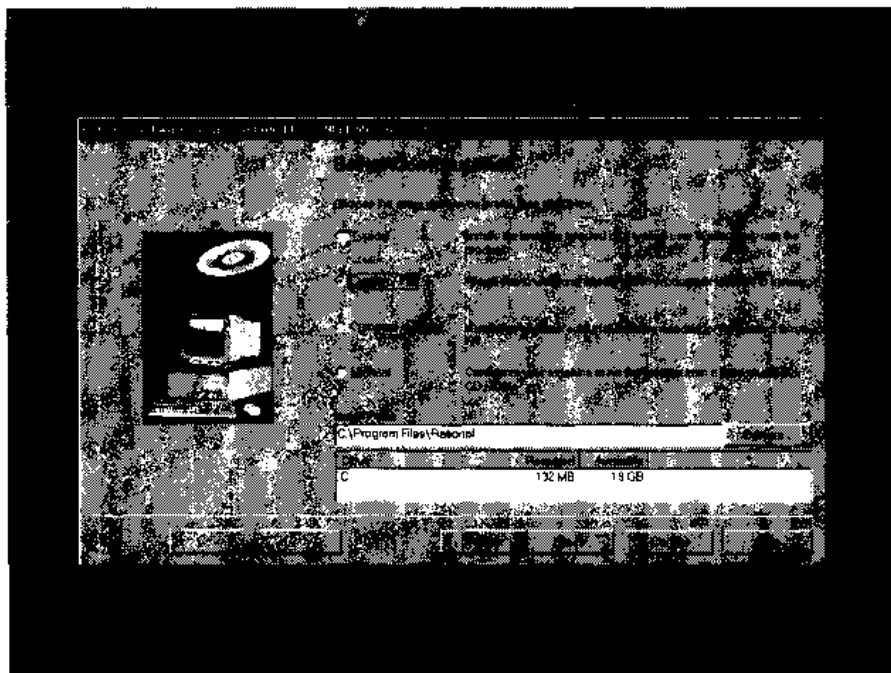


图2.15 确定安装类型

选项包括Typical、Custom/Full、Compact/Laptop和Minimal。Compact选项只安装Rose的Modeler组件和帮助文件，Typical选项安装Rose和常用插入件，包括：

- Rational Synchronizer
- Rose C++插入件
- Rose CORBA插入件
- Rose Java插入件
- Rose Oracle8插入件
- Rose Type Library Importer插入件
- Rose Version Control插入件
- Rose Visual Basic插入件
- Rose Visual C++插入件
- Rose Web Publisher插入件

Custom选项可以选择要安装的插入件。建议选择Typical选项。本书的所有练习均假设用Typical选项安装Rose。

然后要更新组件和文件。如果不更新组件，则安装不成功，因此出现图2.16时要单击Next继续。

开始复制文件和配置Rose之前，还有一次机会检查所有设置，如图2.17。可以检查所需的磁盘空间，安装哪些组件，将文件安装在哪里，图标安装在哪个程序组中，选择哪个缺省语言。如果一切正确，单击Next继续，开始复制文件和配置Rose。

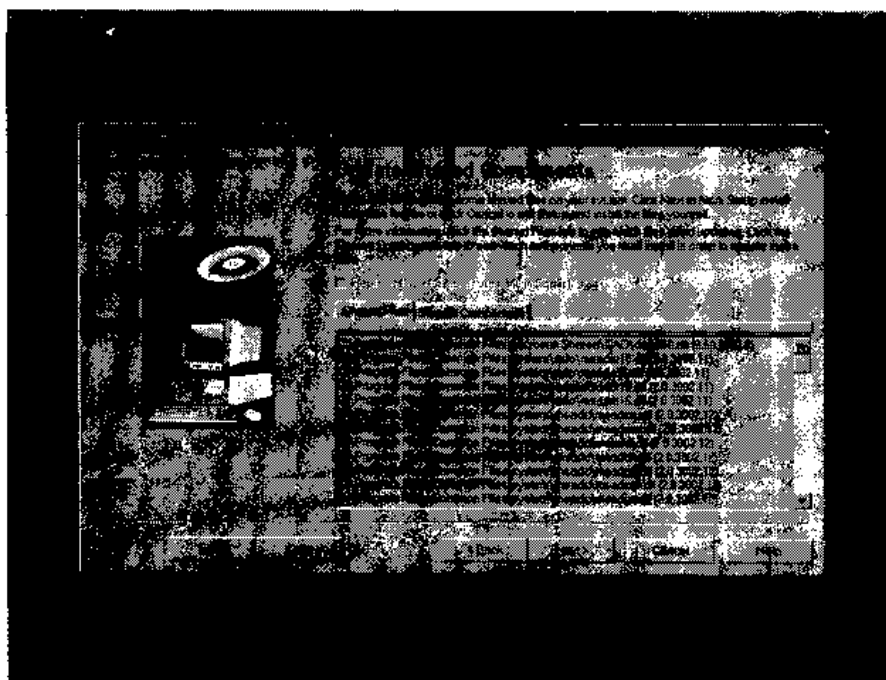


图2.16 更新共享组件

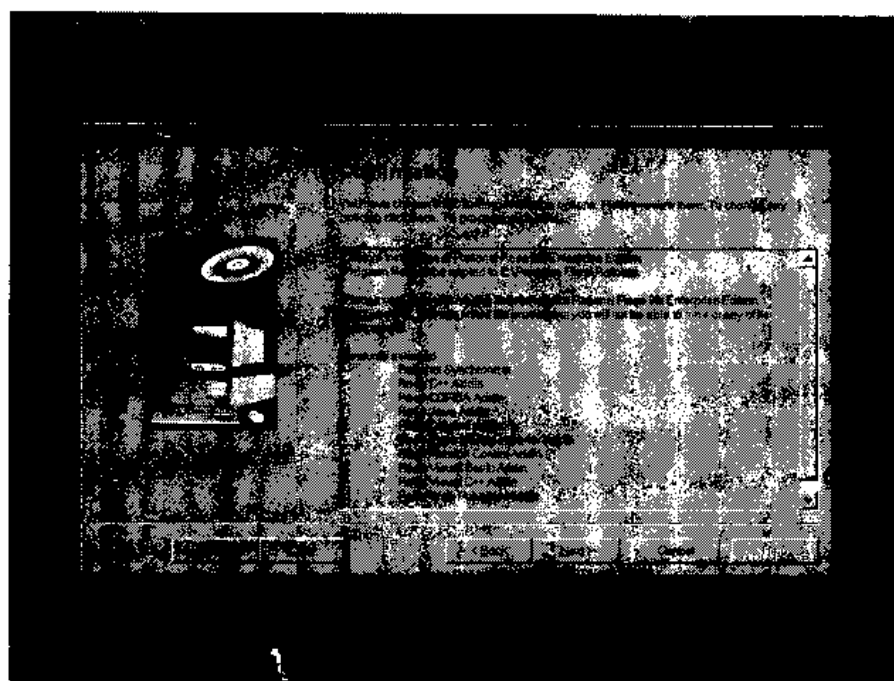


图2.17 检查设置信息

如果将Rational Rose 98升级到Rational Rose 98i, 则会出现图2.18所示屏幕, 显示安装错误汇总信息。如果安装程序要复制的文件已经存在, 则会出现错误提示。只要汇总信息中没有严重错误, 即可单击Next继续。

汇总信息复制完成后, 程序请求重新启动Windows, 如图2.19。要完成安装过程, 必须重新启动Windows。选择Restart并单击Finish重新启动Windows。

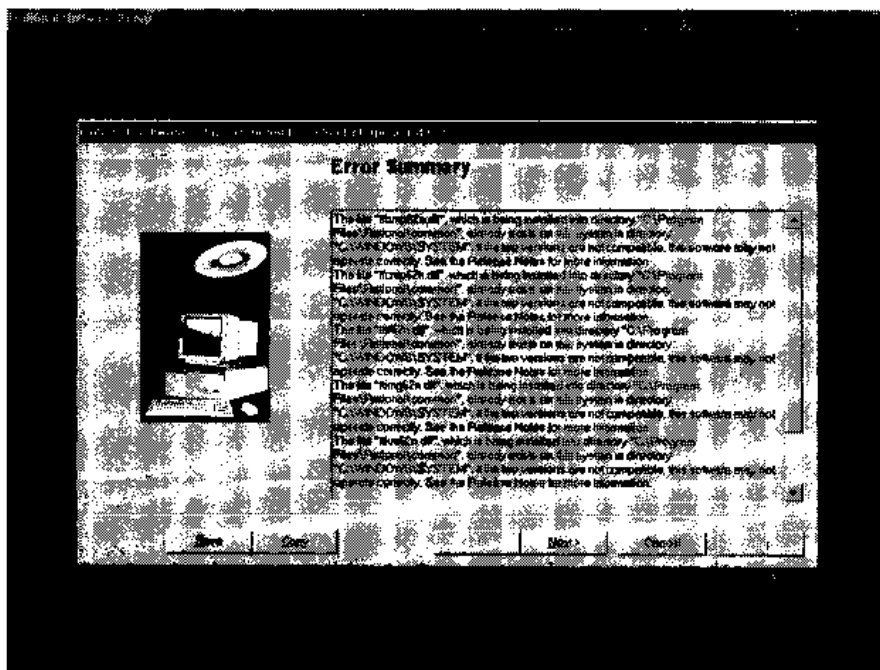


图2.18 如果将Rational Rose 98升级到Rational Rose 98i, 则会出现错误

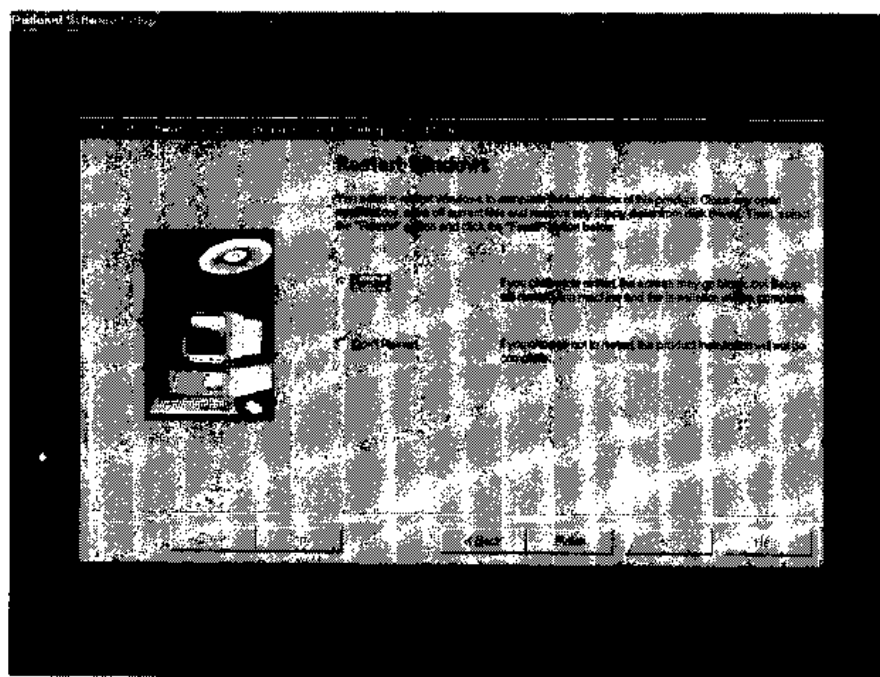


图2.19 重新启动Windows

重新启动Windows后, 出现图2.20。这时安装过程已经完成, 但还要输入有效许可证关键字之后才能运行Rational Rose。选择License Key Administrator的复选框, 然后单击Finish。

启动License Key Administrator后, 如图2.21, 单击Continue按钮开始授权过程

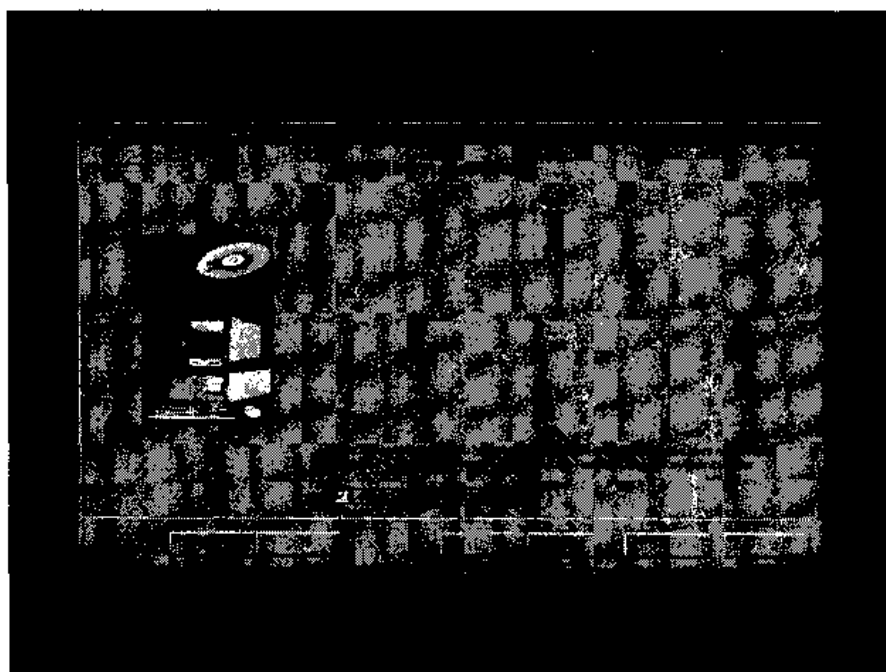


图2.20 启动License Key Administrator

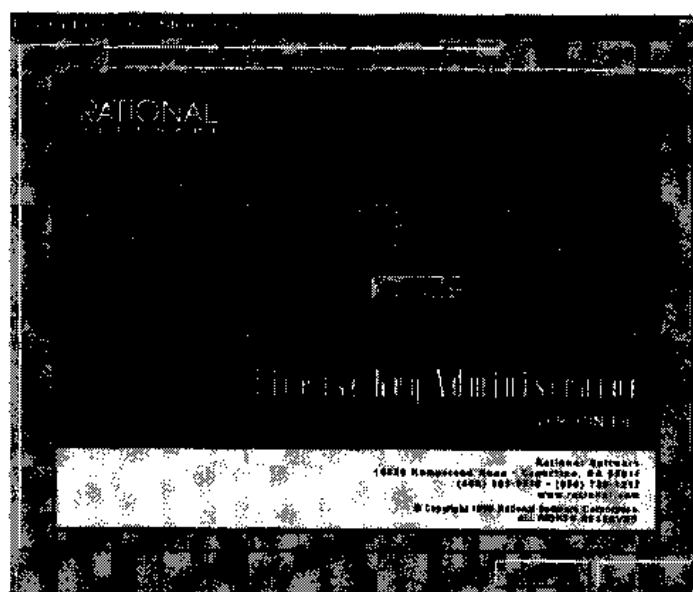


图2.21 License Key Administrator

License Key Administrator的General标签如图2.22所示。使用Rose之前，要先输入有效许可证关键字。输入公司名和Rational帐号，还可以输入用户名。然后选择License Contact标签。

要输入负责许可证的联系人姓名，这是在图2.23所示的License Contact标签输入的。红色字段是必需的，黑色字段是可选的。至少要输入下列信息：

- First name (名)
- Last name (姓)

- Country (国家)
- Phone number (电话号码)
- E-mail address or fax number (e-mail地址或传真号码)
- Complete postal address (完整邮政地址)

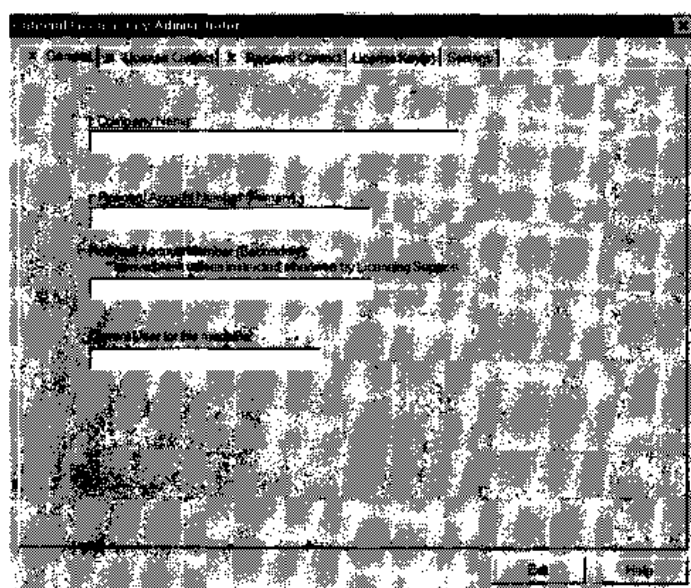


图2.22 License Key Administrator的General标签

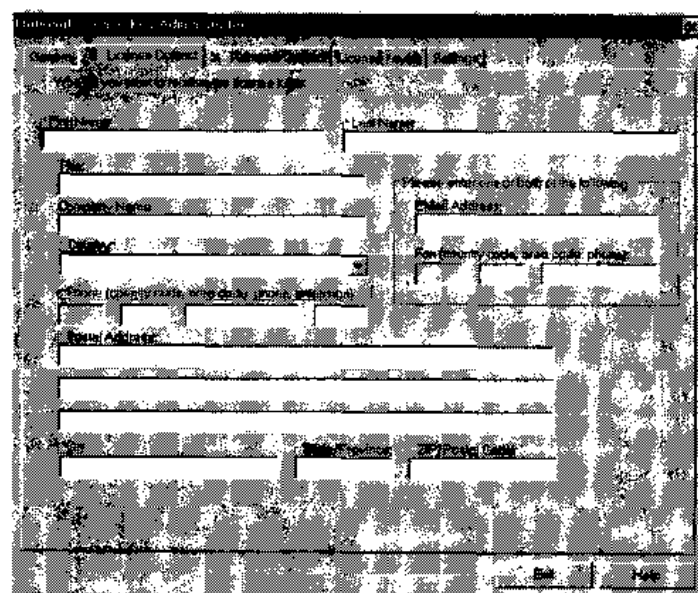


图2.23 License Key Administrator的License Contact标签

输入信息后，选择图2.24所示的Renewal Contact标签。

更新许可证时，Rational需要有个联系人。在这个标签中输入联系人信息，至少要输入下列信息：

- First name (名)
- Last name (姓)

- Country (国家)
- Phone number (电话号码)

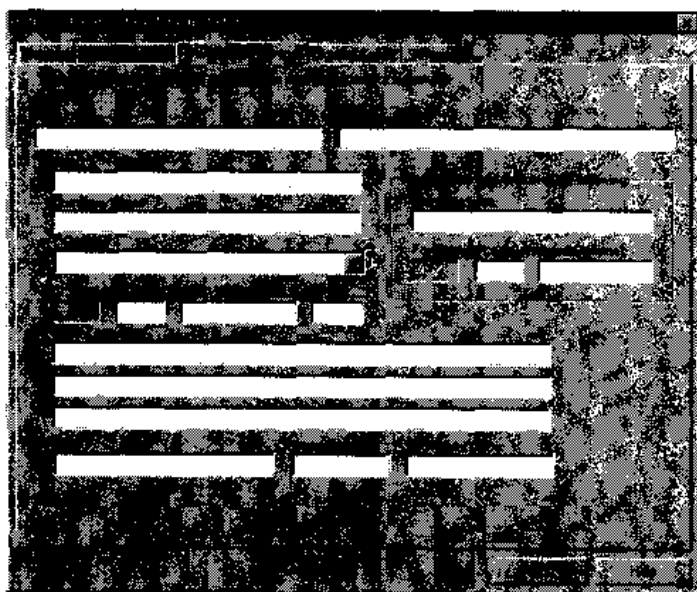


图2.24 License Key Administrator的Renewal Contact标签

然后选择图2.25所示的License Keys标签。

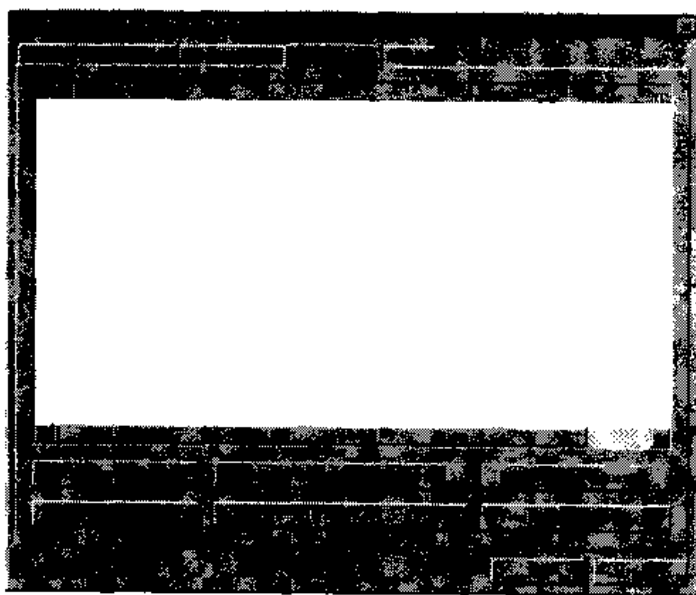


图2.25 License Key Administrator的License Keys标签

License Keys标签保存当前正在使用的许可证信息。即将到期的许可证用不同颜色加亮显示。如果首次在这台计算机上安装Rational产品，则没有正在使用的许可证。单击Enter a License Key按钮设置Rose的许可证。

这时启动图2.26所示向导，选择要输入的许可证类型。Rose产品所带的启动许可证可以使用Rose 30~60天。Term License Agreement可以在指定时间内使用这个产品。如果从Web

下载Rose或从光盘接收Rose, 则应包括启动许可证关键字。选择Startup License Key并单击Next。

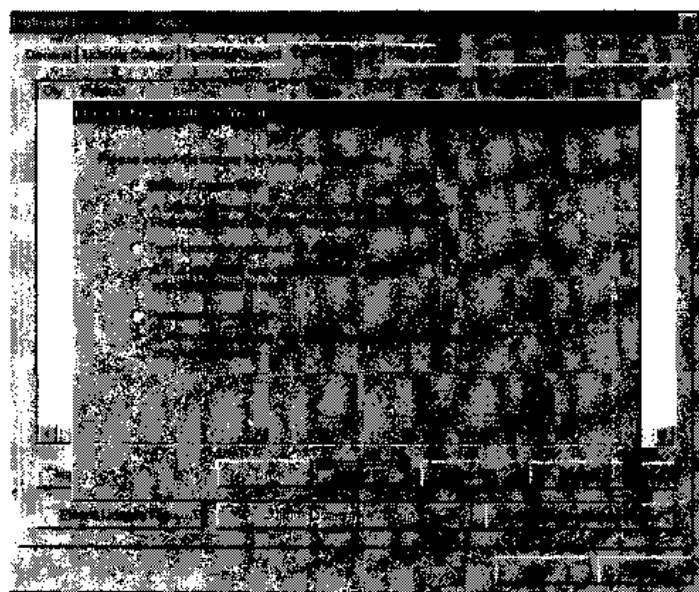


图2.26 许可证关键字向导第一个屏幕

向导询问要输入的许可证类型, 如图2.27。节点锁定许可证可以在一台计算机上使用。浮动许可证可以在网络上使用。除非用已经建立的许可证服务器安装浮点许可证Rose, 否则选择节点锁定。然后单击Next继续。

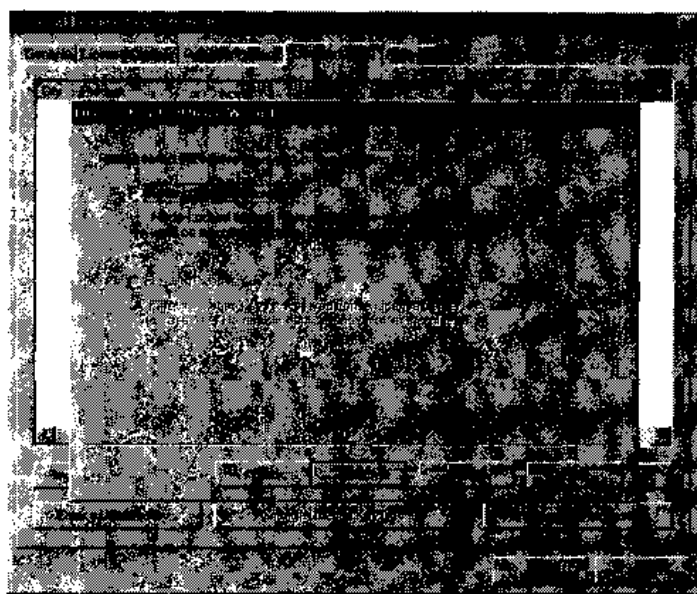


图2.27 许可证关键字向导第二个屏幕

最后出现图2.28所示屏幕。输入授权的产品名和版本号、到期日期和许可证证书中的许可证关键字。完成之后单击Finish。

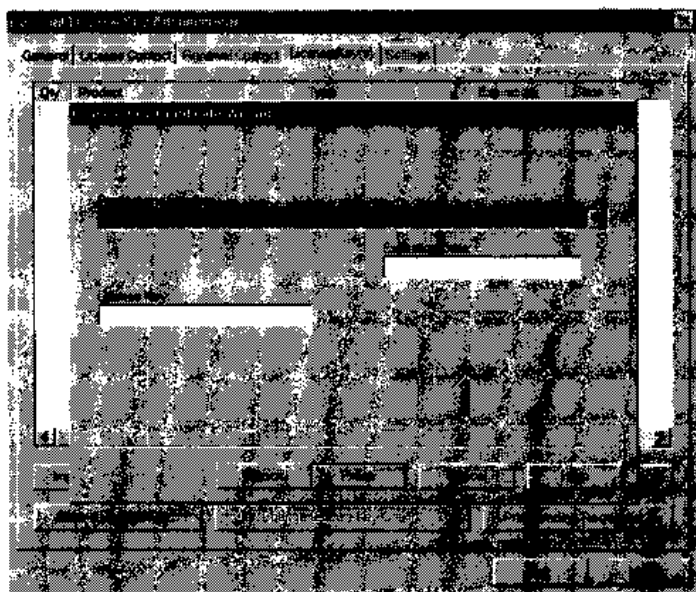


图2.28 许可证关键字向导第三个屏幕

安装完毕，可以开始用Rational Rose建立系统模型。如果使用启动许可证，则可以在指定时间内使用这个产品。经过这个时间后，产品不再可用。要继续使用这个产品，只能从Rational经销商那里购买Rose。

Rose漫游

本章下面几节要介绍Rose界面的各个部分。Rose是个菜单驱动应用程序，用工具栏帮助使用常用特性。Rose支持七种不同类型的UML框：Case框图、Sequence框图、Collaboration框图、Class框图、State Transition框图、Component框图和Deployment框图。Rose对不同框图显示不同的工具栏。本书第一部分余下章节介绍如何生成各种框图。

除了工具栏和菜单外，Rose还有相关的弹出菜单，可以右键单击项目访问。例如，右键单击Class框图中的类打开一个弹出菜单。选项有增加类的属性或操作、浏览和编辑类规范、产生类的代码和浏览产生的代码。

在Rose中漫游的最简单方法是使用浏览器。利用浏览器可以方便快捷地访问框图和模型中的其他元素。如果使用Rose遇到麻烦，可以随时按F1访问丰富的联机帮助文件。

屏幕组件

Rose界面的五大部分是浏览器、文档窗口、工具栏、框图窗口和日志。本节介绍这几个部分。简单地说，它们的作用如下：

浏览器：用于在模型中迅速漫游。

文档窗口：用于访问模型元素的文档。

工具栏：用于迅速访问常用命令。

框图窗口：用于显示和编辑一个或几个UML框图。

日志：用于浏览和报告各个命令的结果。

图2.29显示了Rose界面的各个部分。

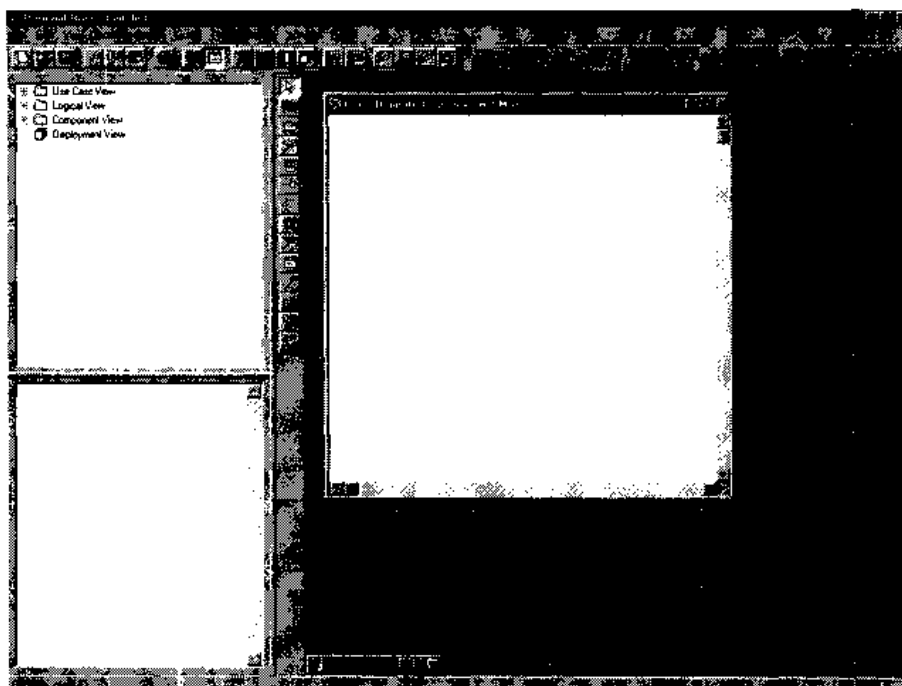


图2.29 Rose界面

浏览器

浏览器是层次结构，用于在模型中迅速漫游。浏览器显示模型中增加的一切：角色、使用案例、类、组件等等。图2.30显示了浏览器。

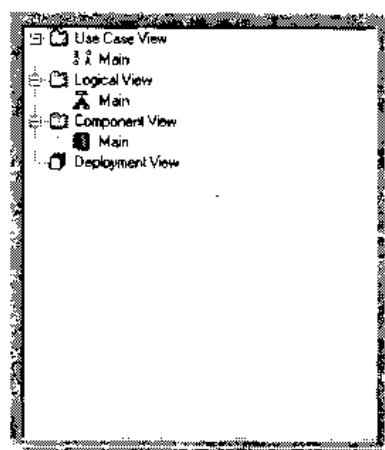


图2.30 Rose浏览器

利用浏览器，可以：

- 增加模型元素（角色、使用案例、类、组件、框图等）
- 浏览现有模型元素

- 浏览现有模型元素间的关系
- 移动模型元素
- 更名模型元素
- 将模型元素加进框图
- 将文件或URL链接到元素
- 将元素组成包
- 访问元素的详细规范
- 打开框图

浏览器中有四个视图：Use Case视图、Logical视图、Component视图和Deployment视图。表2.1列出了每个视图及其中包含的模型元素。

表2.1 Rose浏览器中的视图

视图	内容
Use Case视图	Actors (角色) Use cases (使用案例) Associations (98i) Use Case (文档) Use Case (框图) Sequence (框图) Colaboration (框图) Packages (包)
Logical视图	Classes (类) Class框图 Associations (98i) Interaction (框图) State Transition (框图) Packages (包)
Component视图	Components (组件) Component框图 Packages (包)
Deployment视图	Processes (进程) Processors (处理器) Devices (设备) Deployment框图

利用浏览器，可以浏览每个视图中的模型元素，移动和编程模型元素，增加新的元素。通过在浏览器中右单击元素，可以将文件或URL链接到元素、访问元素的详细规范、删除元素和更名元素。

浏览器组成树视图样式。每个模型元素可能又包含其他元素。模型元素旁边的负号表示该分支已经完全展开，而模型元素旁边的正号则表示该分支是收缩的。

缺省情况下，浏览器出现在屏幕左上角。可以将浏览器移到另一位置或让它作为浮动窗口，也可以隐藏浏览器。

要移动浏览器:

1. 单击选择浏览器窗口边框。
2. 将浏览器从当前位置拖动到屏幕另一区域。

要停靠浏览器:

1. 右单击浏览器窗口边框。
2. 从弹出菜单选择Allow Docking, Allow Docking选项旁边应有个复选标志。这时浏览器可以移动, 但依靠在Rose内。即浏览器窗口连接Rose的另一边框。

要让它作为浮动窗口:

1. 单击选择浏览器窗口边框。
2. 关掉Allow Docking选项。弹出菜单Allow Docking选项旁边的复选标志取消。这时浏览器窗口独立于Rose窗口。浏览器窗口可以在Rose窗口内外随意移动。

要显示或隐藏浏览器:

1. 右单击选择浏览器窗口边框。
2. 从弹出菜单选择Hide, 即可显示或隐藏浏览器。

或者选择View►Browser, Rose即显示或隐藏浏览器。

文档窗口

文档窗口用于建档Rose模型元素。例如, 可以对每个角色写一个简要定义, 可以在文档窗口中输入这个定义, 如图2.31。

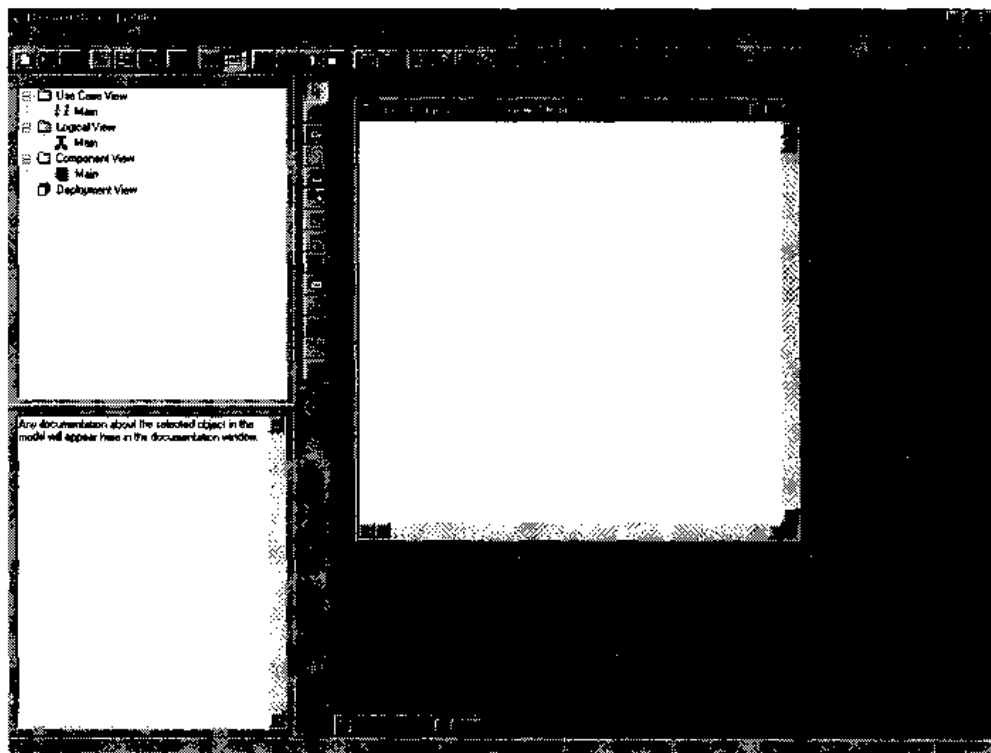


图2.31 文档窗口

将文档加进类中时, 文档窗口中输入的一切都出现为所产生代码的说明语句, 从而不必在今后输入系统代码的说明语句。文档还会在Rose产生的报表中出现。

从浏览器或框图中选择不同元素时, 文档窗口自动更新显示所选元素的文档。

和浏览器一样, 文档窗口也可以停靠或浮动。缺省情况下, 它出现在Rose窗口右下方, 但可以移动或隐藏。

要移动文档窗口:

1. 单击选择文档窗口边框。
2. 将文档窗口从当前位置拖动到屏幕另一区域。

要停靠文档窗口:

1. 右单击选择文档窗口边框。
2. 从弹出菜单选择Allow Docking, Allow Docking选项旁边应有个复选标志。这时文档窗口可以移动, 但依靠在Rose内。即文档窗口连接Rose的另一边框。

要让它作为浮动窗口:

1. 右键单击选择文档窗口边框。
2. 关掉Allow Docking选项。弹出菜单Allow Docking选项旁边的复选标志取消。这时文档窗口独立于Rose窗口, 可以在Rose窗口内外随意移动。

要显示或隐藏文档窗口:

1. 单击选择文档窗口边框。
2. 从弹出菜单选择Hide, 即可显示或隐藏文档窗口。

或者

选择View►Documentation, Rose即显示或隐藏文档窗口。

或者

选择View Documentation工具栏按钮, Rose即显示或隐藏文档窗口。

工具栏

Rose工具栏可以快速访问常用命令。Rose中有两个工具栏: 标准工具栏和框图工具栏。标准工具栏总是显示, 包含任何框图中都可以使用的选项。框图工具栏则随每种UML框图而改变。本书余下部分将详细介绍各种框图工具栏。

表2.2 标准工具栏图标

图标	按钮	用途
	Create New Model	生成新的Rose模型 (.MDL) 文件
	Open Existing Model	打开现有Rose模型 (.MDL) 文件
	Save Model or Log	保存Rose模型 (.MDL) 文件或当前模型日志
	Cut	移动文本到剪贴板
	Copy	复制文本到剪贴板
	Paste	粘贴剪贴板文本

(续表)

图标	按钮	用途
	Print Diagrams	从当前模型打印一个或几个框图
	Context Sensitive Help	访问帮助文件
	View Documentation	浏览文档窗口
	Browse Class Diagram	寻找和打开Class框图
	Browse Interaction Diagram	寻找和打开Sequence或Collaboration框图
	Browse Component Diagram	寻找和打开Component框图
	Browse Deployment Diagram	打开模型的Deployment框图
	Browse Parent	打开框图的父框图
	Browse Previous Diagram	打开最近浏览的框图
	Zoom In	放大比例
	Zoom Out	缩小比例
	Fit in Window	设置显示比例, 使整个框图放进窗口
	Undo Fit in Window	取消Fit in Window命令

所有工具栏都可以定制。要定制工具栏, 选择Tools►Options, 然后选择Toolbars标签。要显示或隐藏标准工具栏:

1. 选择Tools►Options。
2. 选择Toolbars标签。
3. 用Show Standard Toolbar复选框显示或隐藏标准工具栏。

要显示或隐藏框图工具栏:

1. 选择Tools►Options。
2. 选择Toolbars标签。
3. 用Show Diagram Toolbar复选项显示或隐藏框图工具栏。

要在工具栏中使用大按钮:

1. 右单击所选工具栏。
2. 选择Use Large Buttons选项。

要定制工具栏:

1. 右单击所选工具栏。
2. 选择Customize选项。
3. 增删工具栏上的按钮: 选择相应按钮并单击Add或Remove按钮, 如图2.32。

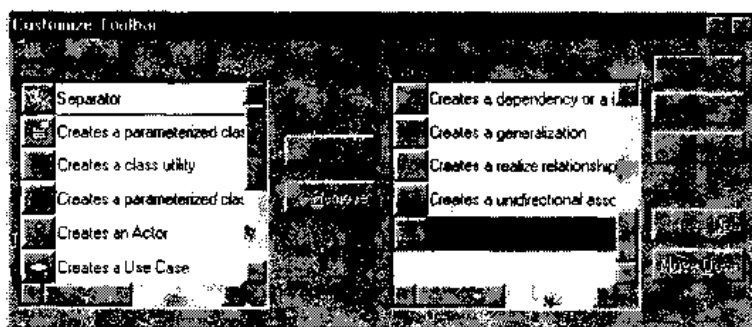


图2.32 定制标准工具箱

框图窗口

在图2.33所示的框图窗口中，可以浏览模型中的一个或几个UML框图。改变框图中的元素时，Rose自动更新浏览器。同样，用浏览器改变元素时，Rose自动更新相应框图。这样Rose就可以保证模型的一致性。

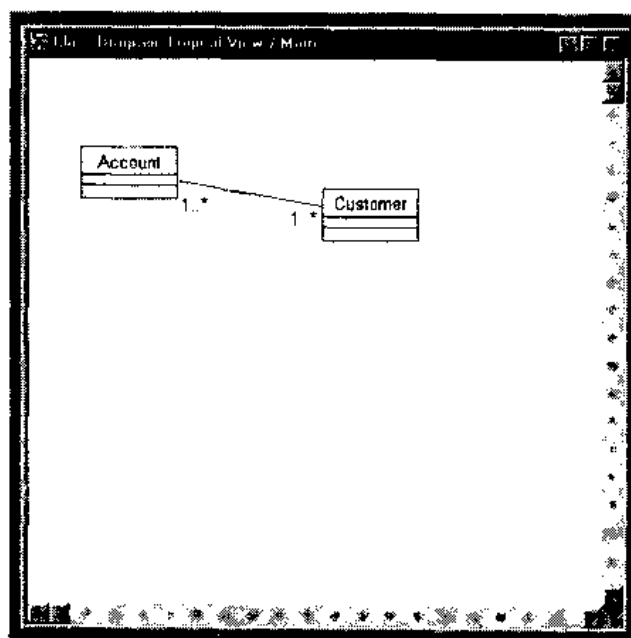


图2.33 框图窗口

日志

使用Rose模型时，有些信息会在日志窗口中发表。例如，生成代码时，生成的任何错误均会在日志窗口中发表，如图2.34。日志窗口无法关闭，但可以最小化。

Rose模型的四个视图

Rose模型的四个视图是Use Case视图、Logical视图、Component视图和Deployment视图。每个视图针对不同对象，具有不同用途。下面几节简要介绍Rose模型的四个视图。本书余下部分会详细介绍各视图中的每个模型元素。

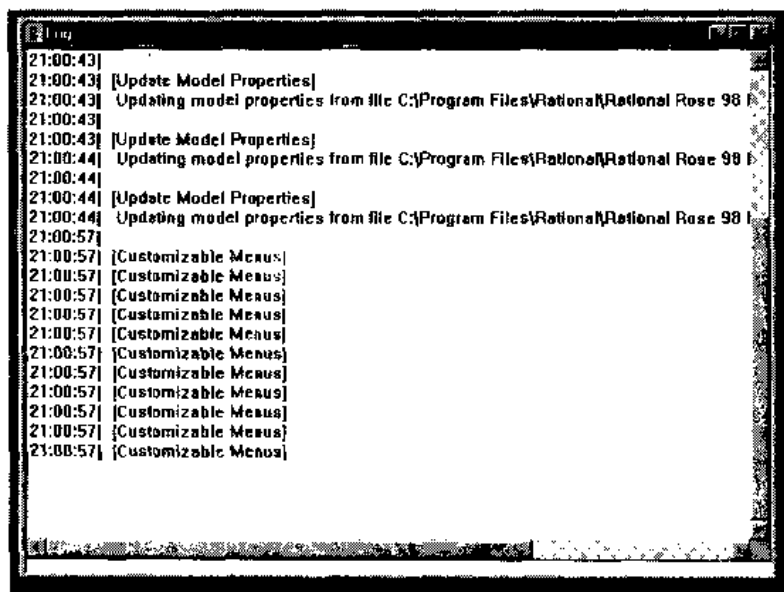


图2.34 日志窗口

Use Case视图

Use Case视图包括系统中的所有角色、使用案例和Use Case框图，还可能包括一些Sequence或Collaboration框图。Use Case视图是系统中与实现无关的视图，关注系统功能的高层形状，而不关注系统的具体实现方法。图2.35是Rose浏览器中的Use Case视图

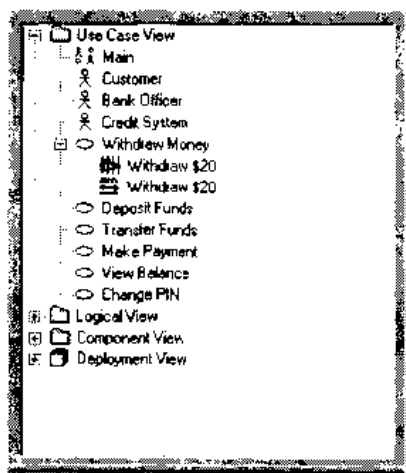


图2.35 Use Case视图

Use Case视图包括：

- ⊗ 角色，是与所建系统交互的外部实体。
- 使用案例，是系统的高层功能块。
- 📁 使用案例文档，详细介绍使用案例的流程，包括任何错误处理。这个图标表示连接Rose模型的外部文件，所用图标取决于建档事件流程所用的应用程序。这里用Microsoft Word。

- ✎ Use Case框图，显示角色、使用案例和它们之间的交互。每个系统通常有几个Use Case框图，分别显示角色和使用案例的子集。
- ✎ Interaction框图，显示一个使用案例流程涉及的对象或类。每个使用案例可能有许多Interaction框图。Interaction框图可以在Use Case视图或Logical视图中生成。独立于语言和实现方法的Interaction框图通常在Use Case视图中生成。这些框图通常显示对象而不是显示类。针对语言的Interaction框图在Logical视图中，这些框图通常显示类而不是显示对象。
- ☐ 包，是角色/使用案例组。包是UML机制，用于将类似项目组合在一起。大多数情况下，角色/使用案例很少，不需要包。但这个工具可以帮你组织Use Case视图。

项目首次开始时，Use Case视图的主要使用者是客户、分析人员和项目管理员。这些人员利用使用案例、Use Case框图和使用案例文档来确定系统的高层视图。这个系统只关注系统的作用，而不关注其实现细节。

随着项目的进行，小组的所有成员可以通过Use Case视图了解正在建立的系统使用案例文档，通过使用案例描述事件流程。利用这个信息，质量保证人员可以开始编写测试脚本。技术作者可以开始编写用户文档。分析人员和客户可以从中确认捕获了所有要求。开发人员可以看到系统生成哪些高级组件，系统逻辑如何。

一旦客户同意了角色/使用案例，就确定了系统范围。然后可以在Logical视图中继续开发，关注系统如何实现使用案例中提出的功能。

Logical视图

Logical视图如图2.36，关注系统如何实现使用案例中提出的功能。它提供系统的详细图形，描述组件间如何关联。除了其他内容外，Logical视图还包括需要的特定类、Class框图和State Transition框图。利用这些细节元素，开发人员可以构造系统的详细设计。

Logical视图包括：

- ☐ 类，是系统的建筑块。类是信息（属性）和功能（操作）的组合。例如，Employee类可能存放员工姓名、地址和社会保险号等信息，包括雇佣和解聘等行为。

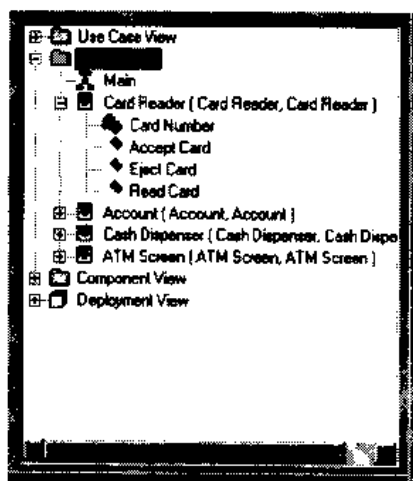
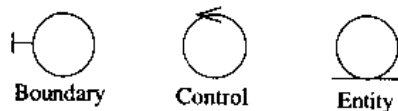


图2.36 Logical视图

-  **Class**框图，用于浏览系统中的类，类的属性与操作及其相互关系。通常，系统有几个Class框图，分别显示所有类的子集。
-  **Interaction**框图，用于显示参与使用案例事件流程的类。前面曾介绍过，Interaction框图可以在Use Case视图或Logical视图中生成。Use Case框图中的Interaction框图通常显示对象，而Logical视图中的Interaction框图通常显示类。
-  **State Transition**框图，显示对象的动态行为。State Transition框图包括对象存在的各种状态，并演示对象如何从一种状态过度到另一种状态，对象首次生成时的状态和对象删除前的状态。
-  **包**，是一组相关类。包装类不是必须的，但有助于组织。有些系统可能有几百个类，包装类可以减少模型的复杂性。要得到系统的一般图形，可以看看包。要看到更详细的视图，可以到包中浏览其中的类。

通常，Logical视图采用两步法。第一步，标识分析类。分析类是独立于语言的类。通过先关注分析类，小组可以不进入语言特定细节而了解系统结构。在UML中，分析类可以用下列图标表示。



Use Case视图的一些Interaction框图中也有分析类。一旦标识分析类，小组可以将每个分析类变为设计类。设计类就是具有语言特定细节的类。例如，我们可能有个负责与另一系统交流的分析类。我们不管这个类用什么语言编写，只关心其中的信息和功能。但是，将它变成设计类时，就要关注语言特定细节。我们可能决定用Java类，甚至确定用两个Java类来实现这个分析类，分析类和设计类不一定一一对应。设计类出现在Logical视图的Interaction框图中。

Logical视图关注的焦点是系统的逻辑结构。在这个视图中，要标识系统组件，检查系统的信息和功能，检查组件之间的关系。这里重复使用是一个主要目的。通过认真指定类的信息和行为，组合类，以及检查类和包之间的关系，就可以确定重复使用的类和包。完成多个项目后，你就可以将新类和包加进重复使用库中。今后的项目可以组装现有的类和包，而不必一切从头开始。

几乎小组中每个人都会用到Logical视图中的信息，但主要用户是开发人员和建筑师。分析人员利用类和Class框图信息确定代码会实现哪些业务要求。质量保证人员通过类、包和Class框图看看系统中的组块有哪些，哪些需要测试。他们还用State Transition框图显示特定类的功能。项目管理员通过类和框图确定系统构造是否合理，并估计系统的复杂程度。

但是，主要用户是开发人员和建筑师。开发人员关心生成什么类，每个类包含的信息和功能。建筑师更关心系统的总体结构。建筑师要负责保证系统结构稳定，考虑了重复使用，系统能灵活地适应需求变化。

一旦标识类并画出框图后，就可以转入Component视图，了解物理结构。

Component视图

Component视图包含模型代码库、执行文件、运行库和其他组件的信息。组件是代码的实际模块。

在Rose中，组件和Component框图在Component视图中显示，如图2.37。系统Component视图可以显示代码模块间的关系。

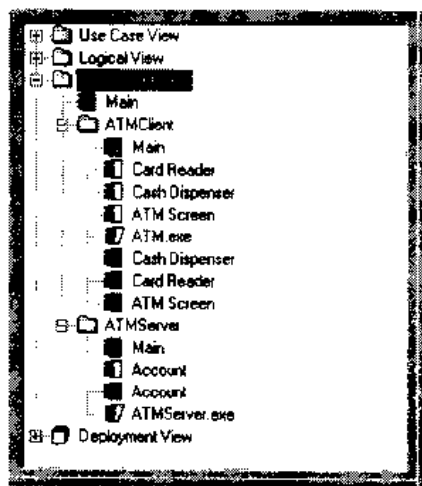


图2.37 Component视图

Component视图包括：

- 组件，代码的实际模块。
- Component框图，显示组件及其相互关系，组件间的关系可以帮你了解编译相关性。利用这个信息，就可以确定组件的编译顺序。
- 包，相关组件的组和包装类一样，包装组件时的目的之一是重复使用。相关组件可以更方便地选择并在其他应用程序中重复使用，只要认真考虑组与组之间的关系。稍后将详细介绍这些问题。

Component视图的主要用户是负责控制代码和编译部署应用程序的人。有些组件是代码库，有些是运行组件，如执行文件或动态链接库（DLL）文件。开发人员也用Component视图显示已经生成的代码库和每个代码库中包含的类。

Deployment视图

Rose中的最后一个视图是Deployment视图。Deployment视图关注系统的实际部署，可能与系统的逻辑结构有所不同。

例如，系统可能用逻辑三层结构。换句话说，界面与业务逻辑可能分开，业务逻辑又与数据库逻辑分开。但部署可能是两层的：界面放在一台机器上，而业务和数据库逻辑放在另一台机器上。

Deployment视图还处理其他问题，如容错、网络带宽、故障恢复和响应时间。图2.38显示了Deployment视图。

Deployment视图包括：

- 进程，是在自己的内存空间执行的线程。
- 处理器，任何有处理功能的机器。每个进程在一个或几个处理器中运行。
- 设备，包括任何没有处理功能的机器。例如打印机。

Deployment框图显示网络上的进程和设备及其相互间的实际连接。Deployment框图还显示进程，哪个进程在哪台机器上运行。

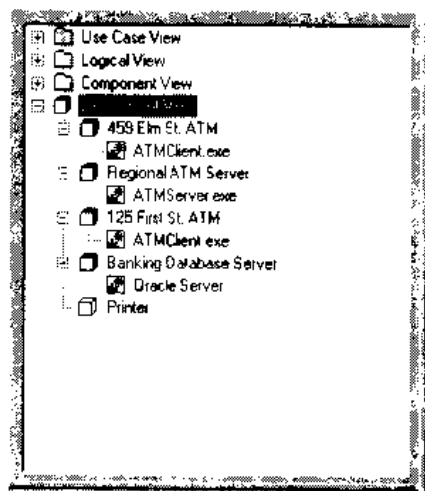


图2.38 Deployment视图

整个小组都用Deployment视图了解系统部署，但主要用户是发布应用程序的人员。

使用Rose

Rose中的一切均与模型有关。本节介绍如何使用模型。首先介绍如何生成和保存Rose模型，然后介绍小组设计（利用控制的单元），最后介绍Rose中的每个菜单项目。

生成模型

使用Rose的第一步是生成模型。模型可以从头生成，也可以利用现有框架模型。Rose模型（包括所有框图、对象和其他模型元素）都保存在一个扩展名为.MDL的文件中。

要生成模型：

1. 从菜单中选择File►New。
2. 如果安装了Framework Wizard，则会出现一些可用框架，如图2.39。选择要用的框架并单击OK，或单击Cancel不用框架。

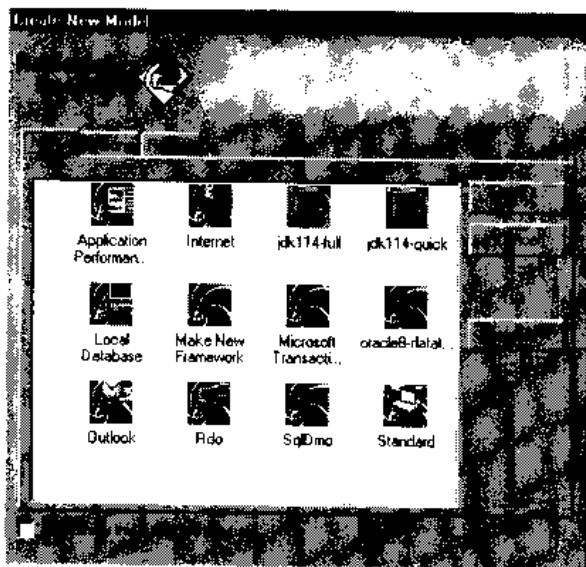


图2.39 Framework Wizard

保存模型

和任何其他应用程序一样，最好定期保存文件，**Rose**也不例外。前面曾介绍过，整个模型都保存在一个文件中。此外，可以将日志保存在另一个文件中。

要保存模型：

选择菜单中的**File>Save**。

或

单击标准工具条上的**Save**按钮。

要保存日志：

1. 选择日志窗口。
2. 选择菜单中的**File>Save Log As**。
3. 输入日志文件名。

或

1. 选择日志窗口。
2. 单击标准工具条上的**Save**按钮。
3. 输入日志文件名。

输出与输入模型

面向对象机制的一大好处是重复使用。重复使用不仅适用于代码，也适用于模型。要充分利用重复使用功能。**Rose**支持输出与输入模型和模型元素。可以输出模型或部分模型，将其输入另一模型。

要输出模型：

1. 选择菜单中的**File>Export Model**。
2. 输入输出文件名。

要输出类包：

1. 从**Class**框图中选择要输出的包。
2. 选择菜单中的**File>Export <Package>**。
3. 输入输出文件名。

要输出类：

1. 从**Class**框图中选择要输出的包。
2. 选择菜单中的**File>Export <Class>**。
3. 输入输出文件名。

要输入模型、包或类：

1. 选择菜单中的**File>Import Model**。
2. 选择要输入的文件名，可选模型（**.MDL**）和子系统（**.SUB**）。

向Web发表模型（98i）

利用**Rational Rose 98i**可以方便地将**Rose**模型发表到**Web**（**Intranet**、**Internet**或文件系统站点）。这样，许多需要浏览模型的人都可以浏览，不必是**Rose**用户，不必打印出一堆模型文档。图2.40显示了发表到**Web**上的模型。

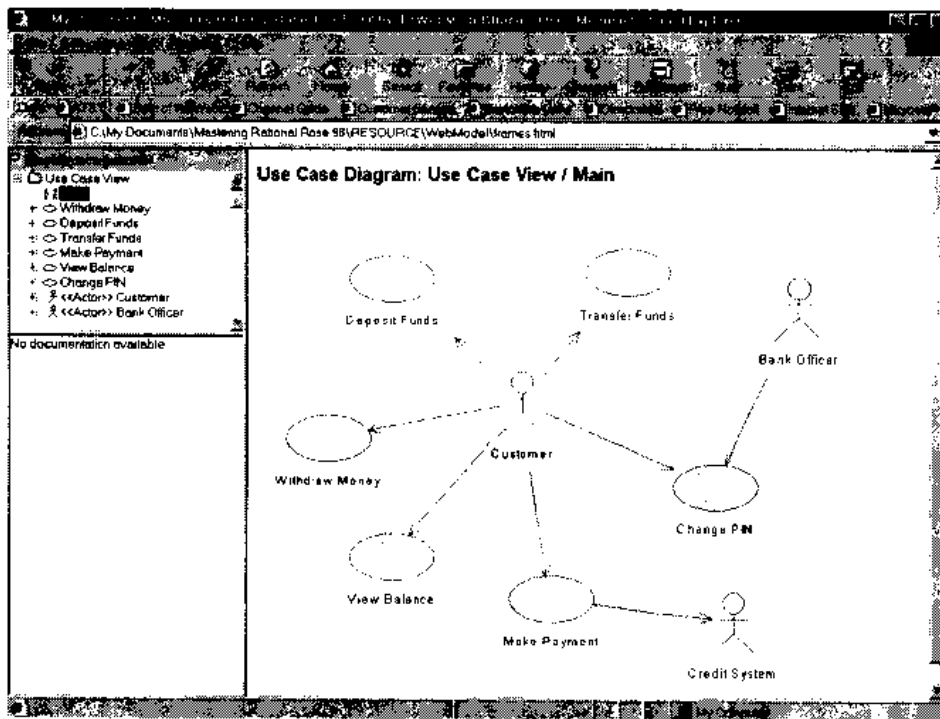


图2.40 发表到Web上的模型

要将模型发表到Web:

1. 选择菜单中的Tools►Web publisher。
2. 从图2.41所示的Web publisher窗口选择要发表的模型视图和包。

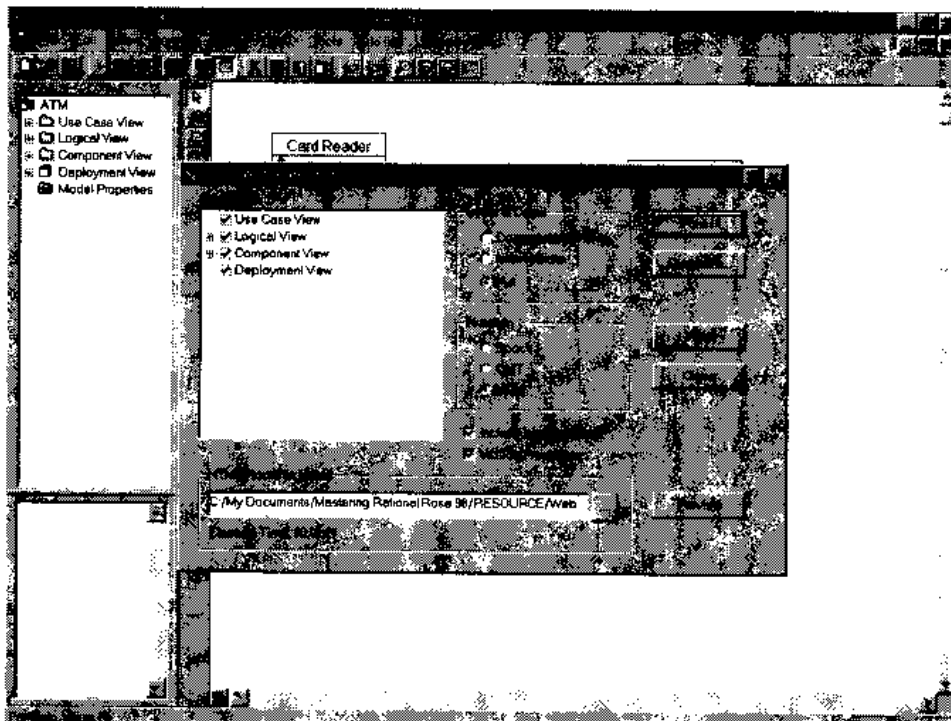


图2.41 Web publisher窗口

3. 选择所要的细节程度。**Documentation only**显示高级信息而不显示模型元素的属性。**Intermediate**显示模型元素规范General标签中的属性。**Full detail**发表所有属性，包括模型元素规范Detail标签中的属性。
4. 选择发表时使用的图注方法，**Notation**缺省为Rose中的缺省图注方法。
5. 选择是否发表继承项目。
6. 选择是否发表属性。
7. 输入发表模型的HTML根文件名。
8. 如果要选择框图的图形文件格式，选择**Diagrams**按钮，出现图2.42所示的**Diagram Options**窗口。

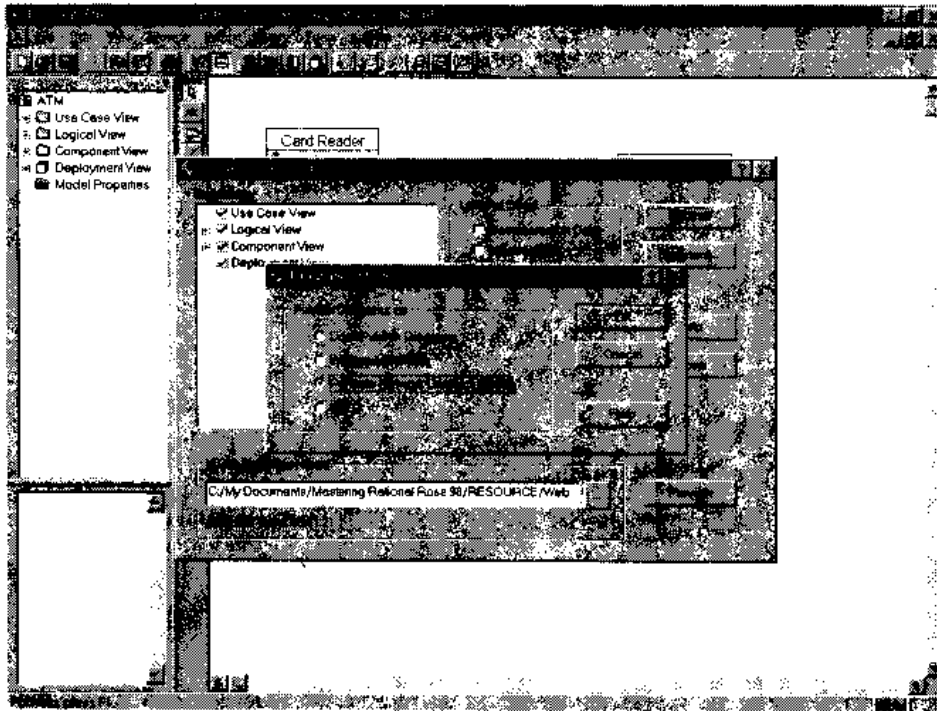


图2.42 Diagram Options窗口

9. 选择发表框图时使用的图形格式类型。选择**Windows Bitmap**、**Portable Network Graphic (PNG)**或**JPEG**，也可以选择不发表任何框图。
10. 准备之后，单击**Publish**。Rose生成发表模型的所有Web页面。
11. 如果需要，可以单击**Preview**浏览发表的模式。

使用控制单元

Rose通过控制单元支持多用户并行开发。Rose中的控制单元可以是Use Case视图、Logical视图或Component视图中的任何包。此外，Deployment视图和Model Properties单元也可以进行控制。控制一个单元时，它存放在独立于模型其他部分的文件中。这样，独立文件可以利用支持SCC的版本控制工具进行控制，如Rational ClearCase、Microsoft SourceSafe和Rose自带的基本工具。控制单元可以从浏览的模型中装入或卸载，使用控制工具还可以检

查进口和出口（Checked In和Out）。在Rose 98中，单元用图2.43所示的窗口管理。在Rose 98i中，单元通过右单击浏览器中的包时弹出的菜单管理。

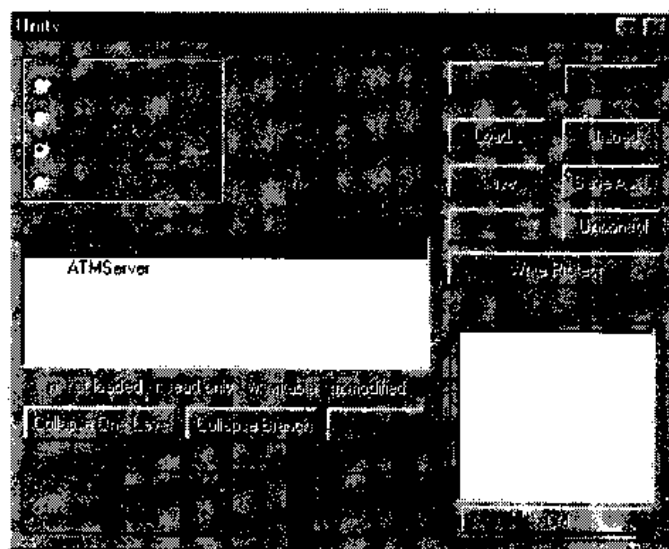


图2.43 管理单元

要在Rose 98中生成控制单元：

1. 选择菜单中的Browse>Units。
2. 出现Units窗口时，选择要控制的包。
3. 单击Control按钮。

要在Rose 98i中生成控制单元：

1. 右单击要控制的单元。
2. 选择菜单的Units>Control<package>。
3. 输入控制单元文件名。注意这时浏览器中的图标上用文件夹的页面符号表示控制该包。

在并行开发环境中，可能需要卸载包，使别人能使用这个包。要卸载控制单元：

1. 右单击要卸载的单元。
2. 选择菜单中的Units>Unload<package>。注意这时浏览器中的包项目删除，表示从模型中删除。

要卸载视图中的所有控制单元：

1. 右单击视图。
2. 选择菜单中的Units>Unload subunits of<View>。

你可能要定期重装另一开发组更新的包：

1. 右单击要重装的单元。
2. 选择菜单中的Units>Load<package>。
3. 单击Load按钮。
4. 选择控制单元。

要取消单元的控制:

1. 确保装入了该控制单元。
2. 右单击要取消控制的单元。
3. 选择菜单中的Units>Uncontrol<package>。

任何时候都可以浏览而不修改项目。要防止修改控制单元, 将单元标为写保护。

要写保护控制单元:

1. 右单击要写保护的单元。
2. 出现Units窗口时, 选择要写保护的包。这个包应当还没有写保护。
3. 单击Write Protect按钮。

要允许写入控制单元:

1. 选择菜单中的Browse>Units。
2. 出现Units窗口时, 选择要允许写入的包。这个包应为写保护。
3. 单击Write Enable按钮。

使用菜单

本节介绍Rose中的每个菜单项目及其简要说明。注意, 有些菜单项目只能在一定条件下访问。例如, Save Log菜单项目只能在激活日志时访问。有些Rose 98i中的菜单项目已经改变位置, 表2.3列出了Rose菜单项目及其简要说明。Rose 98i中的菜单项目标为(98i), Rose 98中的标为(98)。

表2.3 Rose菜单项目及其简要说明

菜单项目	说明
File>New	生成新Rose模型
File>Open	打开现有Rose模型
File>Save	保存当前Rose模型
File>Save As	用新文件名保存当前Rose模型
File>Units>Load	装入控制单元(包)
File>Units>Save	保存控制单元
File>Units>Save As	用新文件名保存控制单元
File>Units>Unload	卸载控制单元
File>Units>Control	控制包
File>Units>Uncontrol	停止控制包
File>Units>Write Protection	写保护控制单元
File>Units>CM	配置管理(只用于使用源配置工具时)
File>Import	输入模型、子系统、包或类
File>Export Model	输出模型、子系统、包或类
File>Update	从逆向转出工程代码模型更新模型
File>Print Diagrams	打印模型框图
File>Print Specifications	指定打印框图选项
File>Print Setup	设置打印机(Windows打印机选项)
File>Edit Path Map	编辑Rose使用的各种路径
File><Recent File>	打开最近的模型文件

(续表)

菜单项目	说明
File>Exit	退出Rose
Edit>Undo	取消上一操作
Edit>Redo	重做上一操作
Edit>Cut	从框图中剪切所选对象
Edit>Copy	复制框图中所选对象
Edit>Paste	粘贴前面剪切或复制的对象
Edit>Delete	删除框图中所选对象
Edit>Select All	选择当前框图中所有对象
Edit>Delete From Model	永久删除模型中的对象
Edit>Relocate	将一个包中的类变成当前包中的类
Edit>Diagram Object Properties>Font Size	改变所选对象的字体大小
Edit>Diagram Object Properties>Font	改变所选对象的字体
Edit>Diagram Object Properties>Line Color	改变所选对象的字体的线颜色
Edit>Diagram Object Properties>Fill Color	改变所选对象的字体填充颜色
Edit>Diagram Object Properties>Use Fill Color	触发使用所选对象的填充颜色
Edit>Diagram Object Properties>Automatic Resize	自动根据文本大小缩放所选对象图标
Edit>Diagram Object Properties>Stereotype Display	显示所选对象的版型
Edit>Diagram Object Properties>Stereotype Label	触发显示所选关系版型标签
Edit>Diagram Object Properties>Show Visibility	触发显示表示可见性的图标——public/protected/private/package (实现方法)
Edit>Diagram Object Properties>Show Compartment Stereotypes	触发显示当前单元中的版型
Edit>Diagram Object Properties>Show Operation Signature	触发显示操作签名
Edit>Diagram Object Properties>Show All Attributes	显示单元的所有属性
Edit>Diagram Object Properties>Show All Operations	显示单元的所有操作
Edit>Diagram Object Properties>Suppress Attributes	不显示属性单元
Edit>Diagram Object Properties>Suppress Operations	不显示操作单元
Edit>Find	寻找对象所在包
Edit>Reassign	将所选对象变为另一对象
Edit>Compartment	选择当前单元中显示的项目
Edit>Change Into>Class	将所选对象变为类
Edit>Change Into>Parameterized Class	将所选对象变为参数化类
Edit>Change Into>Instantiated Class	将所选对象变为实例化类
Edit>Change Into>Class Utility	将所选对象变为类实用程序
Edit>Change Into>Parameterized Class Utility	将所选对象变为参数化类实用程序
Edit>Change Into>Instantiated Class Utility	将所选对象变为实例化类实用程序
Edit>Change Into>Uses Dependency	将所选关系变为相关性
Edit>Change Into>Inherits	将所选关系变为一般化
Edit>Change Into>Instantiates	将所选关系变为实例化
Edit>Change Into>Association	将所选关系变为关联
Edit>Change Into>Realize	将所选关系变为实现

(续表)

菜单项目	说明
Edit>Change Line Style>Rectilinear	将所选关系的线型变为有向直线(带一定角度的直线)
Edit>Change Line Style>Oblique	将所选关系的线型变为斜线(任意做角的直线)
Edit>Change Line Style>Toggle	将所选关系的线型变为在直线与斜线间触发
View>Toolbars>Standard	触发显示标准工具栏
View>Toolbars>Toolbox	触发显示工具和工具栏
View>Toolbars>Configure	配置工具栏
View>Status Bar	触发显示状态栏
View>Documentation	触发显示文档窗口
View>Browser	触发显示浏览器
View>Zoom to Selection	缩放显示所选对象
View>Zoom In	放大当前框图
View>Zoom Out	缩小当前框图
View>Fit in Window	缩放当前框图以使在当前窗口中显示所有对象
View>undo Fit in Window	取消上一Fit in Window操作
View>Page Breaks	触发显示当前框图中的分页
View>Refresh	重画当前框图
View>As Booch	用Booch图注方法显示所有框图上的对象
View>As OMT	用OMT图注方法显示所有框图上的对象
View>As Unified	用UML图注方法显示所有框图上的对象
Browse>Class Diagram	选择所有包的Class框图
Browse>Use Case Diagram	选择所有包的Use Case框图
Browse>Interaction Diagram	选择所有包的Interaction框图
Browse>Component Diagram	选择所有包的Component框图
Browse>State Diagram	选择所有包的State框图
Browse>Deployment Diagram	选择所有包的Deployment框图
Browse>Expand	显示对所选包生成的第一个框图
Browse>Parent	显示所选框图的父框图
Browse>Specification	显示所选对象的指定窗口
Browse>Top Level	显示当前视图的顶层框图
Browse>Referenced Item	显示所选对象所在包的主框图
Browse>Previous Diagram	显示最近的框图
Browse>Create Message Trace Diagram	在Sequence框图与Collaboration框图之间变换
Browse>Units (98)	管理控制单元
Report>Show Usage	报告所选对象出现的每个Class框图
Report>Show Instances	报告所选对象出现的每个Interaction框图
Report>Show Access Violations	报告还没有建立关系的用法
Report>Show Participants in UC	报告所选使用案例所拥有框图的所有类和操作
Report>Documentation Report	产生Word格式的模型文档报表
Query>Add Classes	将模型中的类加进当前框图
Query>Add Use Cases	将模型中的使用案例加进当前框图

(续表)

菜单项目	说明
Query>Expand Selected Items	将与所选对象有关系的对象加进当前框图
Query>Hide Selected Items	删除当前框图中与所选对象有关系的对象
Query>Filter Relationships	只显示所有框图中所选关系类型
Tools>Layout Diagram	布置当前框图以减少关系线重叠
Tools>Autosize All	缩放当前框图中的所有对象
Tools>Create>Text	生成文本对象
Tools>Create>Note	生成图注对象
Tools>Create>Note Anchor	控制对象说明
Tools>Create>Class	生成新类
Tools>Create>Parameterized Class	生成新的参数化类
Tools>Create>Class Utility	生成新类实用程序
Tools>Create>Parameterized Class Utility	生成新的参数化类实用程序
Tools>Create>Association	生成新关联
Tools>Create>Aggregate Association	生成新汇总
Tools>Create>Unidirectional Association	生成新的单向关联
Tools>Create>Unidirectional Association(98i)	生成新的单向汇总
Tools>Create>Link Attribute	生成新的链路属性
Tools>Create>Generalization	生成新的-一般化
Tools>Create>Dependency	生成新的相关性
Tools>Create>Package	生成新包
Tools>Create>Instantiated Class	生成新的实例化类
Tools>Create>Instantiated Class Utility	生成新的实例化实用程序
Tools>Create>Instantiates	生成新的实例化关系
Tools>Create>Actor	生成新角色
Tools>Create>Use Case	生成新使用案例
Tools>Create>Interface	生成新接口
Tools>Create>Realize	生成新实现
Tools>Check Model	检查模型错误
Tools>Model Properties>Edit	编辑所选对象的规范
Tools>Model Properties>Replace	将模型属性的当前设置换成文件中存放的设置
Tools>Model Properties>Export	将模型属性的当前设置输出到文件中
Tools>Model Properties>Add	将文件中的模型属性加进模型属性的当前设置中
Tools>Model Properties>Update	用模型属性的当前设置更新模型属性文件
Tools>Options	显示选项窗口, 允许改变Rose中的所有选项
Tools>Open Script	打开RoseScript文件
Tools>New Script	生成新RoseScript文件
Tools>Synchronize (98i)	启动Rational Suite Synchronizer (如果安装了)
Tools>DDL>Generate Code	对所选对象生成Data Definition Language (DDL) 代码
Tools>DDL>Browse DDL	浏览生成的DDL文件
Tools>IDL>Generate IDL (98)	对所选对象生成IDL代码
Tools>IDL>Convert Rose 4.0 IDL to Rose 98 IDL(98)	将IDL代码从Rose 4.0变成Rose 98

(续表)

菜单项目	说明
Tools>CORBA>Project Specification (98i)	对CORBA生成设置项目规范, 如环境变量
Tools>CORBA>Syntax Check (98i)	检查模型的当前CORBA语法
Tools>CORBA>Browse CORBA Source (98i)	浏览生成的CORBA文件
Tools>CORBA>Reverse Engineer CORBA (98i)	将CORBA组件逆向转出工程代码为模型
Tools>CORBA>Generate Code (98i)	对所选对象生成CORBA代码
Tools>Java>Project Specification (98i)	对Java生成设置项目规范, 如环境变量
Tools>Java>Syntax Check	检查模型的当前Java语法
Tools>Java>Generate Java	对所选对象生成Java代码
Tools>Java>Browse Java Source	浏览生成的Java文件
Tools>Java>Reverse Engineer Java	将Java组件逆向转出工程代码为模型
Tools>Java>Import JDK 1.1.X (98)	将Java Developer's Kit (JDK) 对象输入当前模型
Tools>Oracle8>Data Type Creation Wizard	用向导界面生成新的Oracle8数据类型
Tools>Oracle8>Ordering Wizard	改变所选对象的属性顺序
Tools>Oracle8>Edit Foreign Keys	从关系表增加、编辑或删除外部关键字
Tools>Oracle8>Analyze Schema	分析Oracle8结构并将其输入当前模型
Tools>Oracle8>Schema Generation	从当前模型生成Oracle8结构
Tools>Oracle8>Syntax Checker	检查对象语法以便生成结构
Tools>Oracle8>Reports	生成Oracle 8文档报表
Tools>Oracle8>Import Oracle8 Data Types	将Oracle8数据类型输入当前模型
Tools>Publish to Repository (98)	将当前模型发表到Microsoft Repository
Tools>Import from Repository (98)	从Microsoft Repository输入模型
Tools>C++>Code Generation	对所选对象生成C++代码
Tools>C++>Reverse Engineering	将C++代码逆向转出工程代码为模型
Tools>C++>Browse Header	浏览生成的C++头文件
Tools>C++>Browse Body	浏览生成的C++体文件
Tools>Web Publisher (98)	将模型发表到Web上
Tools>Source Control>Add to Source Control(98)	将所选对象放在源控制内
Tools>Source Control>Remove From Source Control(98)	从源控制中删除所选对象
Tools>Source Control>Start Source Control Explorer(98)	检查当前放在源控制内的对象
Tools>Source Control>Check In (98)	将所选对象重新放在源控制内
Tools>Source Control>Check Out (98)	将所选对象从源控制内取出
Tools>Source Control>Undo Check Out (98)	取消上一取出操作
Tools>Source Control>Get Latest (98)	从源控制中取得所选对象的最新版本
Tools>Source Control>File Properties (98)	检查源控制文件的属性
Tools>Source Control>File History (98)	检查源控制文件的历史
Tools>Source Control>Source Control Options(98)	浏览源控制选项
Tools>Source Control>About SCC Provider	浏览源控制集成版本信息
Tools>Version Control>Add to Version Control(98)	将所选对象放在版本控制内
Tools>Version Control>Remove From Version Control(98)	从版本控制中删除所选对象

(续表)

菜单项目	说明
Tools>Version Control>Start Version Control Explorer(98)	检查当前放在版本控制内的对象
Tools>Version Control>Check In (98)	将所选对象重新放在版本控制内
Tools>Version Control>Check Out (98)	将所选对象从版本控制内取出
Tools>Version Control>Undo Check Out (98)	取消上一取出操作
Tools>Version Control>Get Latest (98)	从版本控制中取得所选对象的最新版本
Tools>Version Control>File Properties (98)	检查版本控制文件的属性
Tools>Version Control>File History (98)	检查版本控制文件的历史
Tools>Version Control>Version Control Options(98)	浏览版本控制选项
Tools>Version Control>About Rational Rose Version Control integration (98)	浏览版本控制集成版本信息
Tools>Visual Basic>Class Wizard	用向导生成新类
Tools>Visual Basic>Assign to New Component(98)	生成新的执行文件或DLL组件并对其指定所选对象
Tools>Visual Basic>Component Assignment Tool(98)	生成新的执行文件或DLL组件并对其指定所选对象
Tools>Visual Basic>Generate Code (98)	产生所选组件的Visual Basic代码
Tools>Visual Basic>Reverse Engineering Wizard(98)	将Visual Basic代码逆向转出工程代码为模型
Tools>Visual Basic>Update Code (98)	同步所选组件的Visual Basic代码
Tools>Visual Basic>Update Model from Code(98)	将Visual Basic代码逆向转出工程代码为模型
Tools>Visual Basic>Revert to Last Saved (98)	让模型退回最后成功生成代码时的状态
Tools>Visual Basic >Browse Source Code (98)	浏览生成的Visual Basic源代码
Tools>Visual C++>Model Assistant (98)	对所选对象设置模型属性
Tools>Visual C++>Component Assignment Tool(98)	生成新组件并对其赋予所选对象
Tools>Visual C++>Update Code (98)	对所选组件同步Visual C++代码
Tools>Visual C++>Update Model from Code (98)	将Visual C++代码逆向转出工程代码为模型
Tools>Visual C++>Class Wizard (98)	用类向导生成新类
Tools>Visual C++>Restore C++ Source Files (98)	对所选对象恢复上一源文件
Tools>Visual C++>Quick import MFC 6.0 (98)	将MFC 6.0类输入模型
Tools>Visual C++>Options (98)	设置Visual C++代码生成选项
Tools>Class Wizard	用类向导生成新类
Tools>Visual Differencing	检查当前模型与保存模型之间的差别
Tools>ERWin Translation Wizard>Translate Model(98)	用向导从Rose模型生成ERWin的数据模型或从ERWin生成Rose模型
Tools>ERWin Translation Wizard>Select Target Server(98)	对ERWin模型设置目标数据库管理系统
Tools>ERWin Translation Wizard>About ERWin Translation	显示ERWin翻译向导的版本信息
Add-Ins>Add-In Manager	对Rose启用或关闭插件件
Window>Cascade	在Rose中层叠所有打开窗口
Window>Tile	在Rose中平铺所有打开窗口
Window>Arrange Icons	在Rose中布置所有打开图标

(续表)

菜单项目	说明
Window><Window>	到所选窗口
Help>Rational Rose Help Topics	显示Rose帮助信息
Help>Search for Help On	搜索特定帮助主题
Help>Using Help	取得使用Windows Help的帮助
Help>Extended Help (98)	取得使用Rational Unified Process的帮助 (如有)
Help>About	显示Rose版本信息
Help>Rational on the Web>Online Support	到Internet上的Rational在线支持页面
Help>Rational on the Web>Rose Home Page	到Internet上的Rational Rose主页

设置全局选项

字体和颜色等选项用于所有模型对象——类、使用案例、接口、包等。本节介绍如何对模型对象改变字体和颜色。缺省字体和颜色可以用Tools>Options菜单项目设置。

使用字体

在Rose中，可以单独改变框图上对象的字体。这样可以提高模型的可读性。这里介绍如何设置框图上对象的字体或字号。字体和字号用图2.44的窗口设置。

设置对象字体：

1. 选择对象。
2. 选择菜单中的Edit>Diagram Object Properties>Font。
3. 选择字体、样式和字号。

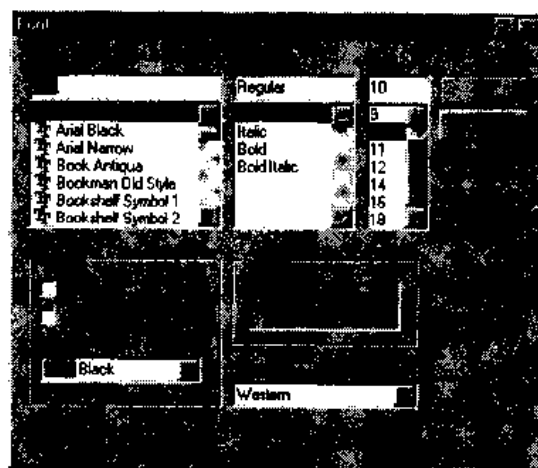


图2.44 字体选择窗口

设置对象字体大小：

1. 选择对象。
2. 选择菜单中的Edit>Diagram Object Properties>Font Size。
3. 选择字号。

使用颜色

除了改变字体外，也可以单独改变对象颜色。下面介绍如何改变对象的线颜色和填充颜色。这些选项用图2.45所示窗口改变。

要改变对象线颜色：

1. 选择对象。
2. 选择菜单中的Edit►Diagram Object Properties►Line Color。
3. 选择线颜色。

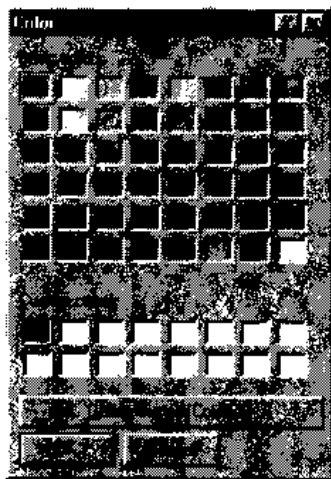


图2.45 颜色选择

要改变对象填充颜色：

1. 选择对象。
2. 选择菜单中的Edit►Diagram Object Properties►Fill Color。
3. 选择填充颜色。

小结

本章简单介绍了Rational Rose。首先介绍Rose及其作用，以及如何安装Rose。然后介绍Rose环境的各个部分，包括屏幕和菜单的不同部分。这样就结束了Rose与UML简介。下一部分要详细介绍如何生成和使用Rose中的不同UML元素与框图。

第3章 使用案例与角色

- 使用Use Case视图与Use Case框图
- 使用使用案例、角色与关系
- 使用图注
- 增加和删除Use Case包

本章介绍使用案例、角色和Use Case框图。使用案例和角色定义所建系统的范围。使用案例包括系统中的一切，角色包括系统外的一切。我们首先介绍如何生成Use Case框图。然后将使用案例加进框图中，并介绍增加的各种选项和细节。然后将角色加进Use Case框图中并介绍角色选项。最后要介绍使用案例与角色间的关系、角色之间的关系和使用案例之间的关系。

本章最后要介绍Rose中的第一个练习。在这些系列练习中，我们要建立订单处理系统模型，本章要引入问题并介绍建立使用案例模型的步骤。

Use Case视图

本章介绍用Rose的Use Case视图生成的一些项目。Use Case视图包含下列项目：

- 使用案例
- 角色
- 使用案例与角色间的通信关系
- 使用案例间使用和扩展关系
- 角色一般化关系
- Use Case框图
- Sequence和Collaboration框图

Sequence和Collaboration框图将在第4章介绍，其余均在本章介绍。

Use Case视图主要是独立实现的。使用案例与角色描述项目范围，而不关心所用编程语言等实现细节。

Use Case框图

Use Case框图显示系统中的使用案例与角色及其相互关系。使用案例是系统提供的高级功能块，角色是与所建系统交互的对象。本章稍后将详细介绍使用案例与角色。图3.1是个Use Case框图例子。

这个框图中有三个角色：客户、银行官员和信用系统。有六大功能：存钱、转帐、取钱、改变PIN、浏览结余和付款。

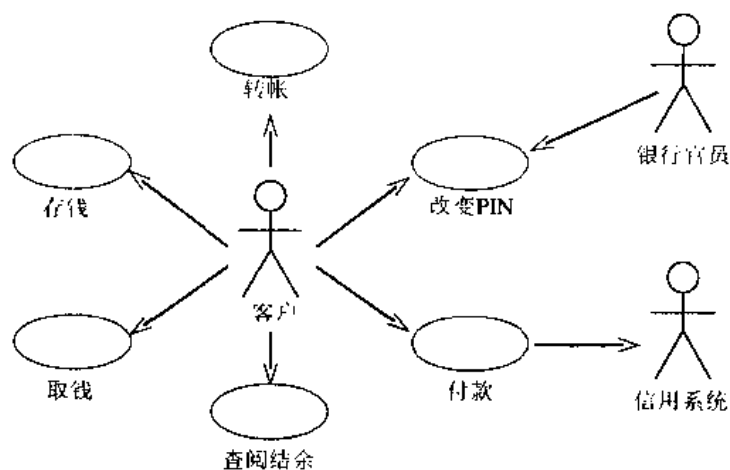


图3.1 Use Case框图样本

Use Case框图的一大优势是通信。客户可以从这个Use Case框图取得大量信息。通过这个使用案例，客户可以知道谁要反对系统。通过查阅使用案例与角色，他们知道项目的具体范围。这样就有助于寻找缺少的功能。例如，有人可能从框图中发现，还要增加浏览帐户中最后十个事务的功能。

通常，一个系统要生成几个Use Case框图，高层框图在Rational Rose中称为主框图，显示使用案例包（组）。其他框图显示一组使用案例与角色。也可能要生成一个所有使用案例与角色的组。生成多少Use Case框图和取什么名完全由你定。框图中应有足够的有用信息，又不至于太拥挤。

Use Case框图有个特定目的——建档角色（系统范围外的一切）、使用案例（系统范围内的一切）及其关系。生成Use Case框图要记住：

- 不要建模角色与角色的通信。按定义，角色在项目范围之外，因此，角色之间的通信不在所建范围之内。可以用 workflow 框图检查角色的通信。
- 不要在两个使用案例之间直接画箭头（除了后面介绍的使用与扩展关系）。框图显示可用的使用案例，但不显示使用案例执行的顺序，这可以用活动框图。
- 每个使用案例都应由角色启动。即应当有个从角色指向使用案例的箭头，除了后面介绍的使用与扩展关系。
- 可以把数据库看成整个Use Case框图下面的层。可以用一个使用案例在数据库中输入信息，然后在另一个使用案例中访问数据库中的信息。不必在使用案例之间用箭头显示信息流程。

生成Use Case框图

Rose在Use Case视图中生成Use Case框图。Rose有个精彩的主Use Case框图，可以生成多个框图以建模系统。

要访问主Use Case框图，做法如下：

1. 单击浏览器中Use Case视图旁边的“+”将其打开。

2. 出现主Use Case框图。注意Rose中的Use Case框图左边有下列图标：



3. 双击主框图将其打开。标题栏变成[Use Case Diagram: Use Case view/Main]。
要生成Use Case框图：

1. 右击浏览器中的Use Case视图。
2. 选择弹出菜单中的New>Use Case Diagram，如图3.2。

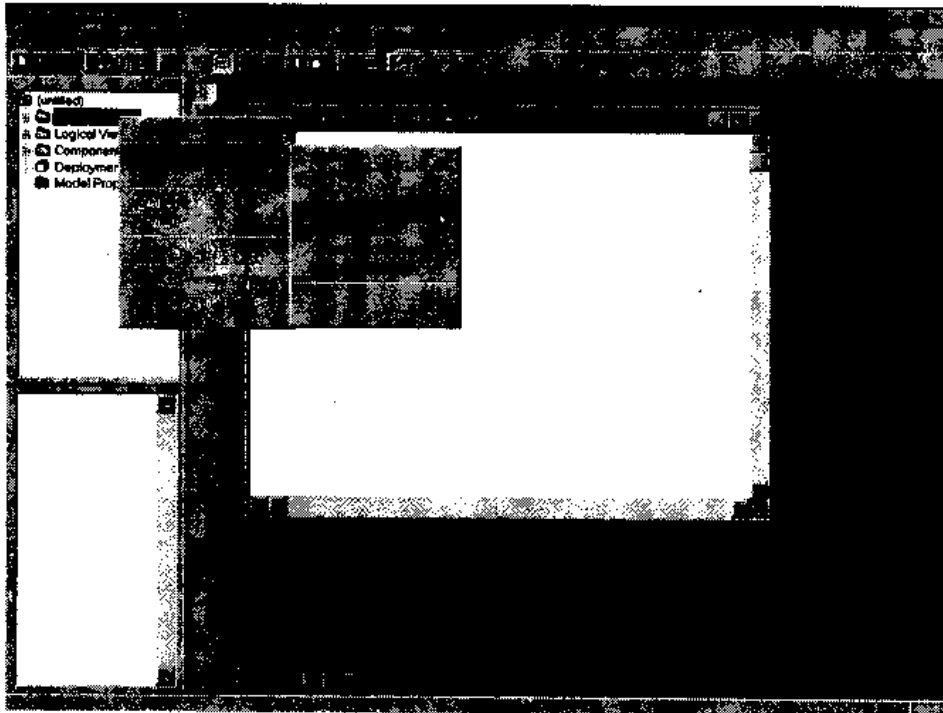


图3.2 增加新的Use Case框图

3. 选择新框图后，输入新框图名。

4. 双击浏览器中的新框图将其打开。

要打开现有Use Case框图：

1. 找到浏览器中Use Case视图中的Use Case框图。
2. 双击Use Case框图名将其打开。

或

1. 选择Browse>Use Case Diagram打开图3.3所示窗口。
2. 在Package列表框中，选择所要打开框图所在的包。
3. 在Use Case Diagrams列表框中选择所要打开的框图。
4. 按OK。

要在Use Case框图中增加项目，用下节介绍的工具栏按钮将使用案例、角色和关系加进框图中。

从Use Case框图中删除项目的方法有两种。第一种从打开框图中删除项目，但项目仍在浏览器中和其他框图中。第二种从整个模型（包括浏览器中和其他框图中）删除项目。

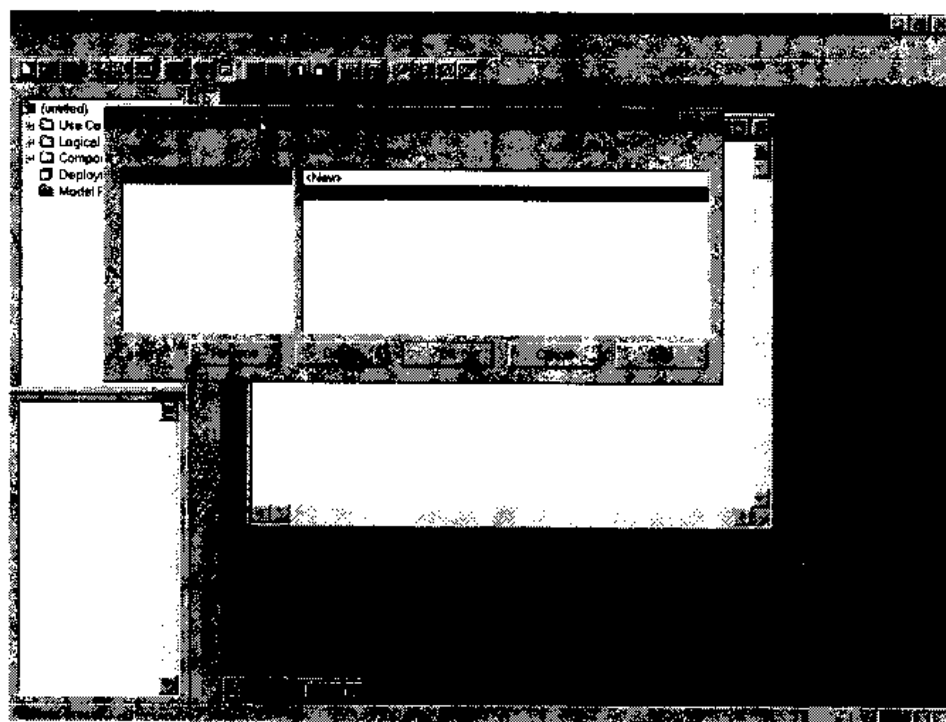


图3.3 打开现有Use Case框图

要从打开框图中删除项目，在框图中加亮项目并按Delete键。

要从整个模型中删除项目，在浏览器中加亮项目并右击，从弹出菜单选择Delete，或在框图中加亮项目后按Ctrl+D。

删除Use Case框图

生成的有些Use Case框图可能要删除。在项目开始时，可能要生成许多Use Case框图以确定范围。有些框图可能包含使用案例，有些显示角色，有些则显示使用案例和角色的子集。随着项目的进行，可能发现要清理一些旧框图。可以直接在浏览器中删除Use Case框图。但要小心，一旦删除之后，是没有办法恢复的。

要删除Use Case框图：

1. 在浏览器中右单击框图。
2. 从弹出菜单选择Delete。

说明：Rose不允许取消框图删除或删除主Use Case框图。

将文件与URL连接到Use Case框图

在Rose中，可以将文件与URL连接。任何补充文档都可以连接Use Case框图，如高级要求规范，项目版本文档或业务案例，以及项目计划。浏览器会在Use Case框图内列出连接的文件和URL。从浏览器中可以双击文件或URL，自动启动相应应用程序，装入文件或URL。

要将文件连接Use Case框图：

1. 在浏览器中右单击框图。

2. 选择New►File。
3. 用Open对话框寻找要连接的文件。
4. 选择Open将文件连接到Use Case框图。

要将URL连接到Use Case框图：

1. 在浏览器中右单击框图。

2. 选择New►URL。

3. 输入要连接的URL名。

要打开连接的文件：

1. 在浏览器中找到文件。

2. 双击文件名，Rose打开相应应用程序并装入文件。

或

1. 在浏览器中右单击文件名。

2. 从弹出菜单选择Open。Rose打开相应应用程序并装入文件。

要打开连接的URL：

1. 在浏览器中找到URL。

2. 双击URL名，Rose打开相应Web浏览器应用程序并装入URL。

或

1. 在浏览器中右单击URL名。

2. 从弹出菜单选择Open。Rose打开相应Web浏览器应用程序并装入URL。

要删除连接的文件或URL：

1. 在浏览器中右单击文件或URL。

2. 从弹出菜单选择Delete。

Use Case框图工具栏

打开Use Case框图时，框图工具栏显示Use Case框图工具栏图标。在工具栏中，Rose提供Use Case框图所有常用功能的快捷键。表3.1列出了其中一些按钮。本章余下部分将介绍如何用这些工具栏按钮在Use Case框图中增加使用案例、角色和其他细节。

表3.1 Use Case框图工具栏图标

图标	按钮	用途
	Selects or deselects an item	让光标返回箭头，以便选择项目
	Text Box	将文本框加进框图
	Note	将图注说明加进框图
	Anchor Note to Item	将图注连接到框图中的使用案例或角色
	Package	将新包加进框图
	Use Case	将新使用案例加进框图
	Actor	将新角色加进框图

(续表)

图标	按钮	用途
	Unidirectional Association	在角色与使用案例之间画出关系
	Dependency or Instantiates	在框图项目之间画出相关性
	Generalization	画出使用案例之间的使用或扩展关系 或角色间的继承关系

使用使用案例

使用案例是系统提供的功能块。换句话说，使用案例演示了人们如何使用系统。下面先举一个例子。ATM机向用户提供一些基本功能，使客户可以存钱、取钱、从一个帐号转帐到另一个帐号、浏览结余、改变PIN和进行信用卡付款。每个事务都是客户使用系统的方法，因此有一个独白的使用案例。在UML中，使用案例表示为如下符号：



通过使用案例观察系统，能够将系统实现与系统目标分开，有助于了解最重要的部分——满足用户要求和期望，而不会沉浸于实现细节。通过使用案例，客户可以看到系统提供的功能，先确定系统范围再深入开展项目工作。

使用案例采用与传统方法不同的方法。将项目分解成使用案例是个面向对象过程而不是面向实现的系统，因此不同于传统的功能分解法。尽管功能分解法关注如何分解成系统能处理的小块，但使用案例首先关注用户对系统的需求。

开始项目时，自然会问：如何寻找使用案例？一个好办法是检查客户提供的文档。通常，高级范围和观点文档有助于确定使用案例。还可以考虑项目的保管人，询问每个保管人要什么系统功能。对每个保管人，问下列问题：

- 这个保管人要用这个系统干什么？
- 保管人是否要维护任何信息（生成、读取、更新、删除）？
- 保管人是否要把外部事件告诉系统？
- 系统是否要把某些改变和事件告诉保管人？

前面曾介绍过，使用案例用独立于实现的用户需求高级视图。下面要单独检查这个定义的每一部分。

第一，使用案例独立于实现。定义使用案例时，假设正在建立用户手册系统。使用案例可能用Java、C++、Visual Basic建立。使用案例关注系统的作用而不是如何实现这个作用。稍后将介绍如何实现该过程。

第二，使用案例是系统的高级视图。如果系统有3000个使用案例，则失去了简单性的好处。使用案例的集合应让客户易于了解高层的整个系统，有这么多使用案例，客户就会在一页页文档间犹豫不决。一般用户的使用案例个数为20到50。稍后会介绍，可以用使用与扩展等关系将使用案例进行必要的分解。也可以将使用案例组合成组，以便于组织。稍后将对

此进行介绍。

最后，使用案例关注使用系统的用户。每个使用案例应表示用户与系统间的完整事务，为用户提供一定价值。使用案例应按业务术语命名，而不是按技术术语命名，应让客户一目了然。在ATM中，我们不能用Interface With The Banking System To Transfer Monies From A Checking To A Credit Card Account（从支票向信用卡帐号转帐的银行系统接口）使用案例，而要用更有意义的Make Credit Card Payment（信用卡付款）使用案例。使用案例通常用动词或短语命名，描述客户看到的最终结果。客户不关心你要面对多个其他系统、要完成哪些具体步骤或进行Visa付款需要几行代码，而只关心发生了付款。这里，你关注用户对系统的需求，而不是达到结果所需的步骤。

这样，有了使用案例的最后清单时，怎么知道是否完整呢？可以回答下列问题：

- 每个功能要求是否至少在一个使用案例中？如果要求不在使用案例中，则不会实现。
- 是否考虑了每个保管者如何使用系统？
- 每个保管人向系统提供什么信息？
- 每个保管人从系统接收什么信息？
- 是否考虑了维护问题？要有人启动和关闭系统。
- 是否标识了系统要交互的所有外部系统？
- 每个外部系统从系统接收什么信息、向系统发送什么信息？

建档事件流

使用案例开始描述系统的作用。要实际建立系统，则需要更具体的细节。这些细节写在事件流文档中。事件流的目的是建档使用案例中的逻辑流程。这个文档详细描述系统用户的工作和系统本身的工作。

尽管事件流程很详细，但仍然是独立于实现方法的。可以在编写事件流时假设有一个自动化系统，但还不应考虑系统到底用C++、PowerBuilder还是Java建立。这里的目的是描述系统干什么，而不是怎么干。事件流通常包括：

- 简要说明
- 前提条件
- 主事件流
- 其他事件流
- 事后条件

下面一项一项介绍这些项目。

简要说明

每个使用案例应有一个相关的说明，描述该使用案例的作用。ATM中的Transfer Funds（转帐）使用案例可以说明如下：

Transfer Funds使用案例让客户或银行职员将钱从一个支票或存款帐号转到另一个支票或存款帐号。

说明要简明扼要，但应包括执行使用案例的不同类型用户和用户通过这个使用案例要达到的最终结果。随着项目的进行，这些使用案例定义有助于整个小组记住项目中每个案例

的意义和作用，还有助于建立目的明确的使用案例，减少小组成员的混淆。

前提条件

使用案例的前提条件列出开始使用案例之前必须满足的条件。例如，前提条件可能是另一个使用案例已经执行或用户具有运行当前使用案例的访问权限。并不是所有使用案例都有前提条件。

前面曾介绍过，Use Case框图并不显示使用案例执行的顺序。但前提条件可以建档这方面的信息。例如，一个使用案例的前提条件可能是另一个使用案例已经执行。

主事件流和其他事件流

使用案例的具体细节在主事件流和其他事件流中描述。事件流描述执行使用案例功能的具体步骤。事件流关注系统干什么，而不是怎么干，是从用户角度写成的。主事件流和其他事件流包括：

- 使用案例如何开始
- 使用案例的各种路径
- 使用案例的正常（主）流程
- 使用案例主事件流的变形（其他事件流）
- 错误流
- 使用案例如何结束

例如，Withdraw Money（取钱）使用案例的事件流可能如下：

主事件流

1. 银行客户将卡插入ATM机，开始使用案例。
2. ATM出现一个欢迎消息并提示用户输入个人标识号（PIN）。
3. 银行客户输入PIN。
4. ATM确认这个PIN有效。如果PIN无效，则执行其他事件流A1。
5. ATM提供下列选项：
 - Deposit Funds
 - Withdraw Cash
 - Transfer Funds
6. 用户选择Withdraw Cash选项。
7. ATM提示输入所取金额。
8. 用户输入所取金额。
9. ATM确定该帐号是否有足够的金额。如果钱不足，则执行其他事件流A2，如果验钱时发生错误，则执行错误流E1。
10. ATM从客户帐号中减去所取金额。
11. ATM向客户提供要取的钱。
12. ATM向客户打印一张收据。

13. ATM退出客户的卡。

14. 使用案例结束

其他事件流A1：输入无效PIN

1. ATM告诉客户该PIN无效。

2. ATM退出客户的卡。

3. 使用案例结束。

其他事件流A2：资金不足

1. ATM告诉客户该帐号资金不足。

2. ATM退出客户的卡。

3. 使用案例结束

错误流E1：验钱时发生错误

1. ATM告诉客户验钱时发生错误，并向客户提供银行客户服务支持部门的电话号码。

2. ATM在错误日志中记下错误。日志项目包括错误日期和时间、客户名和帐号、错误代码。

3. ATM退出客户的卡。

4. 使用案例结束。

建档事件流时，可以像这里一样用编号清单、用段落文本、强调符清单或事件流程图。事件流应与收集的要求一致。确定文档流程时，记住文档的接收者。客户通过审查这个文档相信其准确反映客户的期望。分析人员通过文档保证与要求一致。项目管理员检查文档，更好地了解要建的项目，并可调整项目结算。

编写流程时，一定要避免详细讨论怎么干。就像写菜单，你要写“加两个蛋”，而不写“到冰箱的门上取两个蛋，取第一个蛋，将蛋用碗边敲破，…”。在事件流中，可以说“验证用户ID”，而不指定查找数据库中哪个表格。关注用户与系统之间交换的信息，而不关注系统实现的细节。

事后条件

事后条件是使用案例执行完毕之后必须为真的条件。例如，可以在使用案例完成之后设置一个标志。这种信息就是事后条件。和前提条件一样，事后条件可以增加使用案例顺序方面的信息。例如，如果一个使用案例运行之后必须运行另一个使用案例，则可以在事后条件中说明这一点。并不是每个使用案例都有事后条件。

增加使用案例

向模型中增加使用案例的方法有两种：将使用案例加进活动Use Case框图或直接将新使用案例加进浏览器中。从浏览器中可以再将使用案例加进Use Case框图。

要将新的使用案例加进Use Case框图：

1. 选择工具栏中的Use Case按钮。

2. 单击Use Case框图中任一位置，新使用案例名为NewUseCase。

3. 在新的使用案例仍然选择时，输入新使用案例名称。

4. 注意新使用案例自动加进浏览器的Use Case视图中。

或

1. 选择Tools►Create►Use Case，如图3.4。

2. 单击Use Case框图中任一位置，放上新的使用案例。新使用案例名为NewUseCase。

3. 在新的使用案例仍然选择时，输入新使用案例名称。

4. 注意新使用案例自动加进浏览器的Use Case视图中。

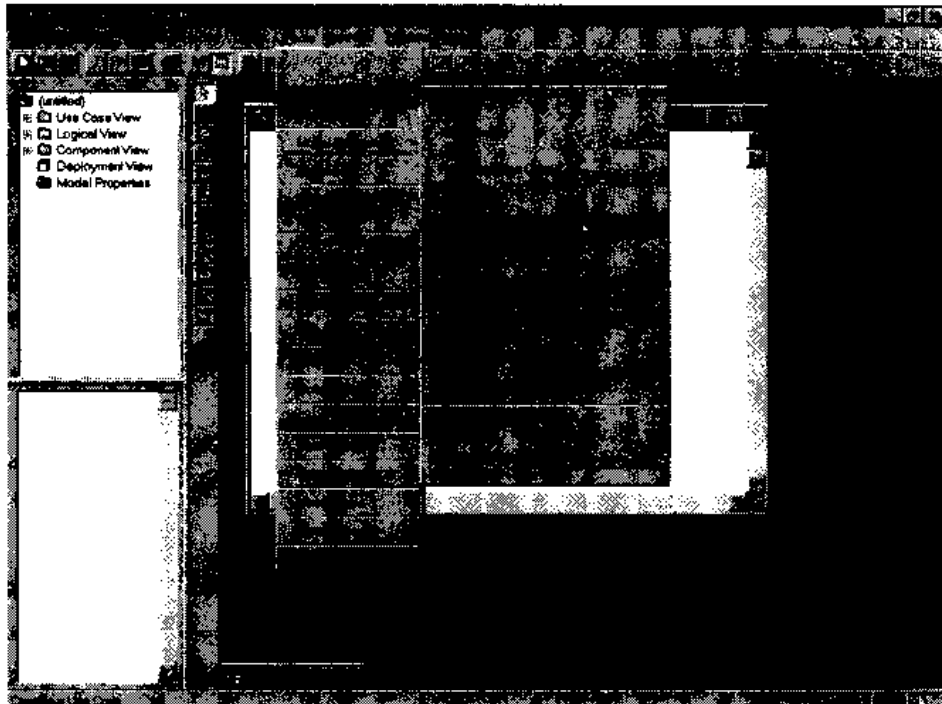


图3.4 将新的使用案例加进Use Case框图

要将现有使用案例加进Use Case框图：

1. 将使用案例从浏览器中拖动到打开的Use Case框图中。

或

1. 选择Query►Add Use Cases打开图3.5所示对话框，可以选择和增加现有使用案例。

2. 在Package下拉列表框图中，选择要加的使用案例所在包。

3. 将要加的使用案例从Use Cases列表框移到Selected Use Cases列表框。

4. 按OK将使用案例加进Use Case框图。

要将使用案例加进浏览器中：

1. 右单击浏览器中的Use Case视图包。

2. 从弹出菜单中选择New►Use Case。

3. 浏览器中出现新的使用案例NewUseCase。新使用案例左边是UML使用案例图标。

4. 选择使用案例后，输入新使用案例名。

5. 要把使用案例加进框图，将新使用案例从浏览器中拖动到框图中。

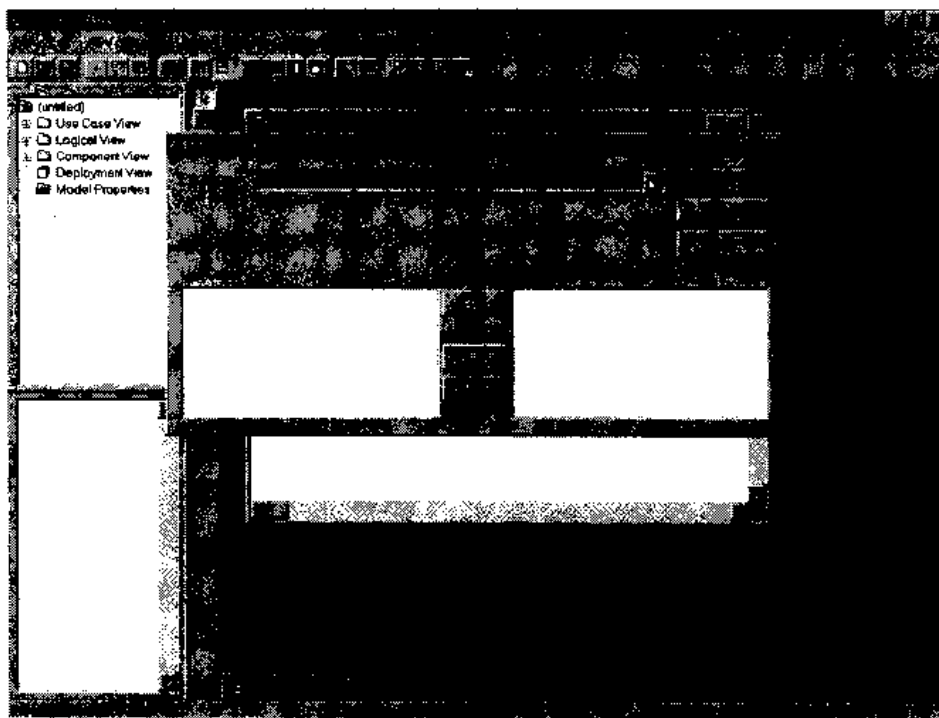


图3.5 将现有使用案例加进Use Case框图

删除使用案例

删除使用案例的方法有两种，可以从一个框图中删除，也可以从整个模型和所有框图中删除。和Use Case框图一样，项目开始时可能有许多多余的使用案例，它们对确定项目范围非常有用。一旦确定最后使用案例组，就要删除多余的使用案例。

从Use Case框图中删除使用案例：

1. 选择框图中的使用案例。
2. 按Delete。
3. 注意使用案例从Use Case框图中删除，但在浏览器和其他Use Case框图中仍然存在。

要从模型中删除使用案例：

1. 选择框图中的使用案例。
2. 选择Edit>Delete from Model或按Ctrl+D键。
3. Rose从整个模型和所有框图中删除这个使用案例。

或

1. 右击浏览器中的使用案例。
2. 从弹出菜单中选择Delete。
3. Rose从整个模型和所有框图中删除这个使用案例。

使用案例规范

Rose提供每个使用案例的详细规范。这些规范可以帮助建档使用案例的特定属性，如使用案例名称、优先级和版型。图3.6显示了使用案例规范窗口，用于设置使用案例规范。下面几节介绍这个窗口中的每个规范。

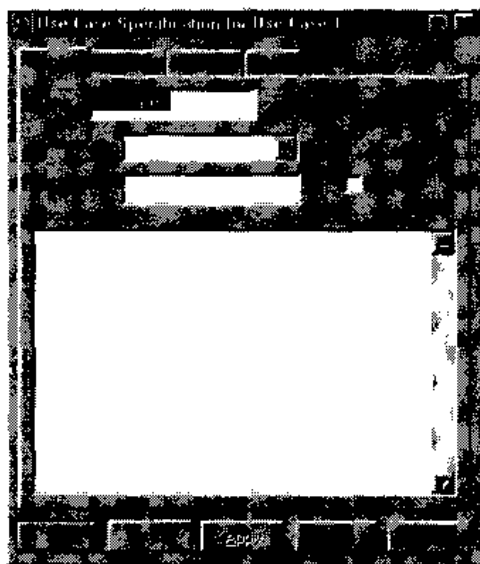


图3.6 使用案例规范窗口

打开使用案例规范：

1. 右击Use Case框图中的使用案例。
2. 从弹出菜单中选择Open Specification。

或

1. 右击浏览器中的使用案例。
2. 从弹出菜单中选择Open Specification。

或

1. 选择框图中的使用案例。
2. 选择Browse►Specification。

或

1. 选择框图中的使用案例。
2. 按Ctrl+B。

命名使用案例

模型中的每个使用案例应有唯一名称。使用案例应从客户角度命名，以帮助确定项目范围。使用案例名还应独立于实现方法。要避免使用使用案例与特定实现方法相联系的短语，如Internet。使用案例通常用动词和短语命名。

使用案例命名的方法有两种，可以用使用案例规范窗口，也可以在框图中直接命名。

命名使用案例：

1. 选择浏览器或Use Case框图中的使用案例。
2. 输入使用案例名。

或

1. 右击浏览器或Use Case框图中的使用案例。
2. 从弹出菜单选择Open Specification。

3. 在Name字段中输入使用案例名。

将文档加进使用案例:

1. 选择浏览器中的使用案例。
2. 在文档窗口中输入使用案例说明。

或

1. 右单击浏览器或Use Case框图中的使用案例。
2. 从弹出菜单选择Open Specification。
3. 在规范窗口的Documentation区输入使用案例说明。

浏览使用案例参与者

你可能想列出参与某个使用案例的所有类和操作。随着项目的进行, 要求可能增加或改变, 知道这个改变影响哪些类很有必要。在ATM例子中, 如果改变取钱的要求, 则要知道哪些类参与Withdraw Money使用案例。

即使系统完成后, 也要记录每个使用案例包括哪些类。系统进入维护阶段时, 需要控制升级和改变的范围。在Rose中, 可以用Report菜单浏览使用案例参与者。

要浏览参与使用案例的类和操作:

1. 选择Use Case框图中的使用案例。
2. 选择Report►Show Participants in UC。
3. 出现Participants窗口, 如图3.7。

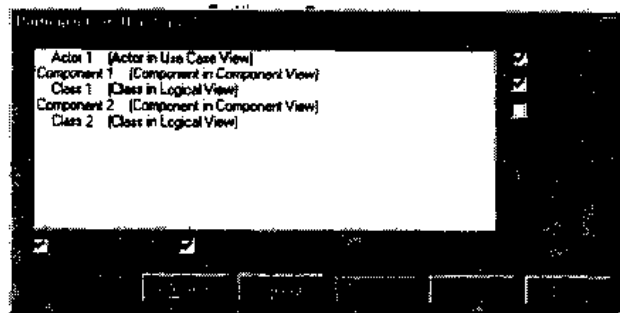


图3.7 使用案例Participants窗口

Display Parent复选框显示参与使用案例的每个类所在的包。包分别在类和操作名后面的括号中。

Display Type复选框可以在列表框图中每个项目旁边放上图注说明, 让你知道项目是类还是操作。类型在类和操作名后面的括号中。

Components、Classes或Operations复选框控制列表框中列出组件、类、操作或三者都列出。Open It按钮浏览清单中一个项目的规范, Goto It按钮选择浏览器中的项目。

指定使用案例版型

在UML中, 版型(stereotypes)用于分类模型元素。例如, 如果有两种使用案例, A类和B类, 可以生成两个新使用案例版型A和B。版型不常用于使用案例, 更常用于其他模型元素, 如类和关系。但如果喜欢, 也可以增加使用案例版型。

要指定使用案例版型:

1. 右单击浏览器或Use Case框图中的使用案例。
2. 从弹出菜单中选择Open Specification。
3. 在Stereotype字段中输入版型。

指定使用案例优先级

定义使用案例时,可能要指定优先级。利用优先级,随着项目的进展,你可以知道处理使用案例的顺序。在Rose的使用案例规范中,可以用Rard字段输入使用案例优先级说明。

要指定使用案例优先级:

1. 右单击浏览器或Use Case框图中的使用案例。
2. 从弹出菜单中选择Open Specification。
3. 在General标签的Rard字段输入使用案例优先级。

生成抽象使用案例

抽象使用案例不是由角色直接启动,而是提供一些其他功能,让其他使用案例使用。抽象使用案例是参与使用或扩展关系(见本章稍后“浏览使用案例关系”一节)的使用案例。图3.8是个抽象使用案例的例子。

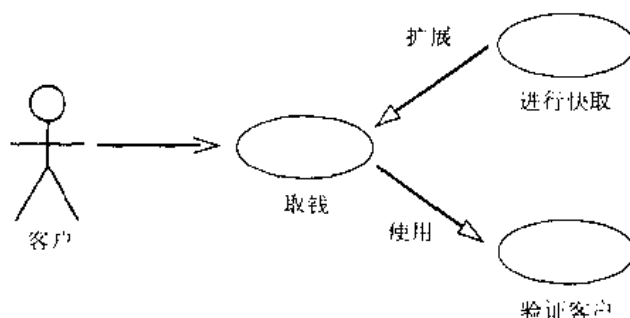


图3.8 抽象使用案例

本章稍后一节将介绍如何在用例之间画箭头。

要生成抽象使用案例:

1. 在浏览器或Use Case框图中生成使用案例。
2. 右单击浏览器或Use Case框图中的使用案例。
3. 从弹出菜单中选择Open Specification。
4. 复选Abstract框。

浏览使用案例的框图

在使用案例规范中可以看到浏览器或Use Case框图中定义的Sequence框图、Collaboration框图、Class框图、Use Case框图和State Transition框图。图3.9显示了使用案例规范窗口的框图标签。在这个标签中,可以看到表示框图类型的Rose图标和框图名。双击框图即可在框图窗口中打开这个框图。

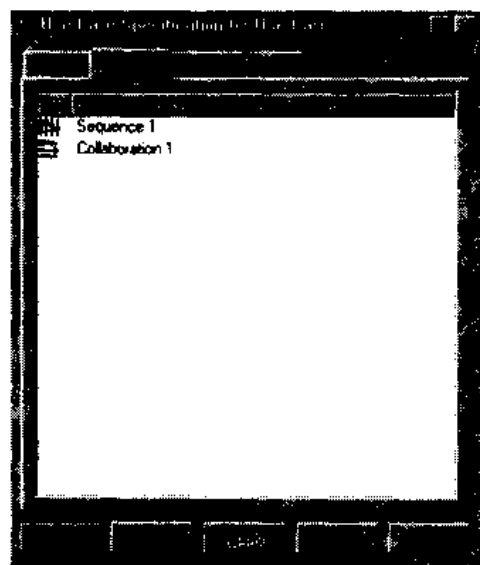


图3.9 使用案例规范窗口的框图标签

要浏览使用案例的框图：

1. 右单击浏览器或Use Case框图中的使用案例。
2. 从弹出菜单中选择Open Specification。
3. 框图即会在使用案例规范窗口的框图标签中列出。

或

浏览浏览器，使用案例框图会在浏览器的使用案例下面列出。

要打开使用案例框图：

双击使用案例规范窗口的框图标签中的框图名。

或

1. 右单击使用案例规范窗口的框图标签中的框图名。
2. 从弹出菜单选择Open Diagram。

或

双击浏览器中的框图。

要将框图加进使用案例：

1. 右单击使用案例规范窗口Diagrams标签中任一位置。
2. 从弹出菜单中选择要加的框图类型（Use Case、Sequence、 Collaboration、 State或 Class）。
3. 输入新框图名。

或

1. 右单击浏览器中的使用案例。
2. 从弹出菜单中选择New►（Collaboration Diagram, Sequence Diagram, Class Diagram, Use Case Diagram）。
3. 输入新框图名。

要删除使用案例的框图:

1. 右单击使用案例规范窗口Diagrams标签中的框图名。
2. 从弹出菜单中选择Delete。

或

1. 右单击浏览器中的框图名。
2. 从弹出菜单中选择Delete。

浏览使用案例的关系

使用案例规范窗口的Relations标签列出了使用案例参与其他使用案例或角色的所有关系,如图3.10。清单列出关系名和关系连接的项目名。关系名包括加进关系中的作用名或关系名。

要浏览使用案例的关系:

1. 右单击浏览器中的使用案例。
2. 从弹出菜单中选择Open Specification。
3. Relations标签列出关系。

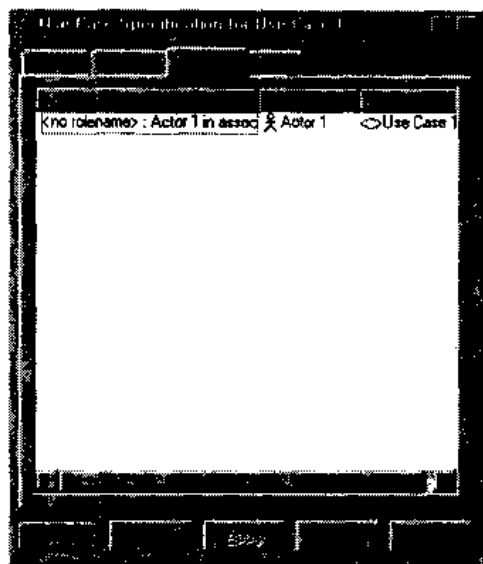


图3.10 使用案例规范窗口Relations标签

或

1. 选择Use Case框图上的使用案例。
2. 选择Report►Show Usage。

要浏览关系规范:

1. 双击清单中的关系。
2. 出现关系规范窗口。下面详细介绍关系规范。

或

1. 右单击清单中的关系。
2. 选择弹出菜单中的Specification。
3. 出现关系规范窗口。下面详细介绍关系规范。

要删除关系：

1. 右单击清单中的关系。
2. 选择弹出菜单中的Delete。

将文件与URL连接使用案例

Files标签如图3.11，可以将文件连接使用案例。例如，文档中可能包含使用案例的事件流。可能还有其他文档，介绍使用案例情形、使用案例中要求的规范和其他描述使用案例的文件。所有这些文件都可以连接Files标签中的使用案例。如果愿意，也可以将URL连接使用案例。

要将文件连接使用案例：

1. 右单击Files标签中任一空白处。
2. 选择弹出菜单中的Insert File插入文件。

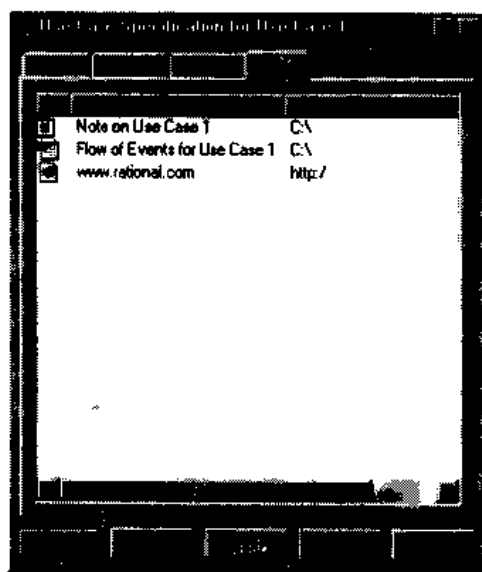


图3.11 使用案例规范窗口Files标签。

3. 用Open对话框寻找要连接的文件。
 4. 选择Open将文件连接使用案例。
- 或
1. 右单击浏览器中的使用案例。
 2. 选择New►File对话框寻找要连接的文件。
 3. 选择Open将文件连接使用案例。

要将URL连接使用案例：

1. 右单击Files标签中任一空白处。
 2. 选择弹出菜单中的Insert URL插入URL。
 3. 输入要连接的URL名。
- 或
1. 右单击浏览器中的使用案例。

2. 选择**New►URL**。

3. 输入要连接的URL名。

要打开连接的文件：

1. 打开使用案例规范。

2. 双击**Files**标签中的文件名。Rose自动启动相应应用程序并装入文件。

或

1. 在浏览器的相应使用案例下面找到文件。

2. 双击**Files**标签中的文件名。Rose自动启动相应应用程序并装入文件。

或

1. 打开使用案例规范。

2. 右单击**Files**标签中的文件。

3. 从弹出菜单选择**Open File/URL**。Rose自动启动相应应用程序并装入文件。

或

1. 在浏览器的相应使用案例下面找到文件。

2. 右单击浏览器中的文件。

3. 从弹出菜单选择**Open**。Rose自动启动相应应用程序并装入文件。

要打开连接的URL：

1. 打开使用案例规范。

2. 双击**Files**标签中的URL名。Rose自动启动相应Web浏览器应用程序并装入URL。

或

1. 在浏览器的相应使用案例下面找到URL。

2. 双击**Files**标签中的URL名。Rose自动启动相应Web浏览器应用程序并装入URL。

或

1. 打开使用案例规范。

2. 右单击**Files**标签中的URL。

3. 从弹出菜单选择**Open File/URL**。Rose自动启动相应Web浏览器应用程序并装入URL。

或

1. 在浏览器的相应使用案例下面找到URL。

2. 右单击浏览器中的URL。

3. 从弹出菜单选择**Open**。Rose自动启动相应Web浏览器应用程序并装入URL。

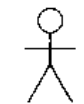
要删除连接的文件或URL：

1. 右单击浏览器中的文件或URL。

2. 从弹出菜单选择**Delete**。

使用角色

角色是与所建系统交互的人或物。使用案例描述系统范围内的一切，而角色描述系统范围外的一切。在UML中，角色表示为如下图形：



Actor 1

角色有三大类：系统用户、与所建系统交互的其他系统和时间。

第一种角色是实际的人，即用户，是最常用的角色，几乎每个系统都有。在ATM例子中，有些用户是ATM客户，有些是ATM维护人员。命名这些角色时，按作用命名而不是按位置命名。一个人可能有许多作用。例如，John Doe负责维护ATM，起ATM维护作用。后来他要取钱吃午饭，起ATM客户作用。利用作用名而不是按位置名就可以得到更稳定的角色图形。位置名随时改变，而作用和责任从一个位置移到另一位置。利用作用命名，就不必在每次增加新位置或改变位置时更新模型。

第二种角色是另一个系统。例如，假设银行有个信息系统，用于维护每个客户的信用帐号信息。ATM系统应与这个信用系统交互。这时，信用系统就成为角色。

注意，通过让这个系统成为角色，我们假设信用系统一成不变。记住，角色在所建范围之外，因此不在我们的控制之内。如果要对信用系统作出重大修改，则应当把它放进项目范围内，而不是作为角色。

第三种常用角色是时间。时间作为角色时，经过一定时间触发系统中的某个事件。例如，ATM系统可能每天午夜运行一些协调处理。由于时间不在我们的控制之内，因此是个角色。

增加角色

和使用案例一样，角色有两种增加方法，打开Use Case框图或直接加进浏览器中。浏览器中的角色可以再加进一个或几个Use Case框图中。

要将角色加进Use Case框图中：

1. 选择工具栏中的Actor按钮。
2. 单击Use Case框图内任一位置。新角色取名NewClass。
3. 选中新角色后，输入新角色名。注意新角色自动加进浏览器中的Use Case视图内。

或

1. 选择Tools►Create►Actor，如图3.12。
2. 单击Use Case框图内任一位置放上新角色。新角色取名NewClass。
3. 选中新角色后，输入新角色名。注意新角色自动加进浏览器中的Use Case视图内。

要将角色加进浏览器中：

1. 右单击浏览器中的Use Case视图包。
2. 选择New►Actor。
3. 新角色NewClass出现在浏览器中，角色名左边是UML角色图标。
4. 选中新角色后，输入新角色名。
5. 要将角色加进框图，从浏览器中将角色拖动到框图中。

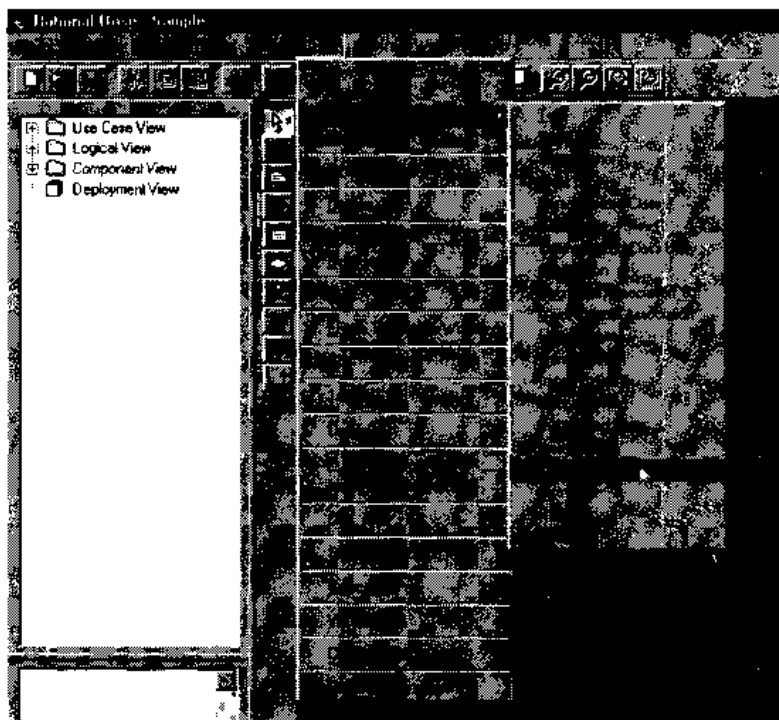


图3.12 将角色加进Use Case框图

删除角色

和使用案例一样，角色有两种删除方法：从单个框图或从整个模型中删除。如果从整个模型中删除角色，则它从浏览器和所有Use Case框图中删除。如果从单个框图中删除角色，则它还留在浏览器和其他Use Case框图中。

要从Use Case框图中删除角色：

1. 选择框图上的角色。
2. 按Delete。

要从模型中删除角色：

1. 选择框图上的角色。
2. 选择Edit>Delete from Model或按Ctrl+D。

或

1. 右单击浏览器中的角色。
2. 从弹出菜单中选择Delete。

Rose从浏览器和所有Use Case框图中删除角色。

角色规范

和使用案例一样，每个角色在Rose中有一些详细规范在图3.13所示的角色规范窗口中，可以指定角色名、版型、基数和其他细节。下面几节要介绍角色可以设置的每个细节。

本书稍后使用类时，可能会发现角色规范窗口与类规范窗口相似。这是因为Rose把角色看成特殊形式的类。角色规范窗口包含与类规范窗口相同的字段，但有些字段对角色关闭

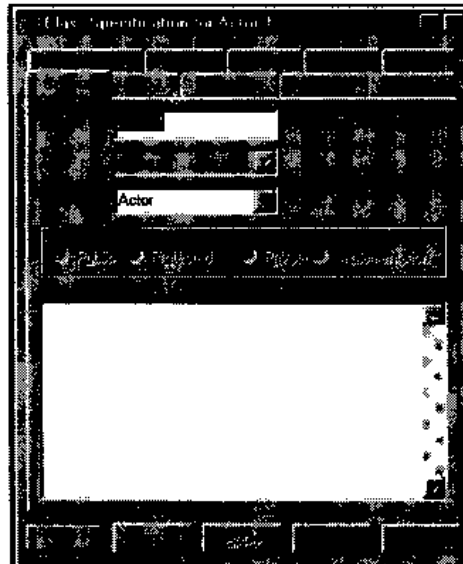


图3.13 角色规范窗口

要打开角色规范:

1. 右单击Use Case框图中的角色。
 2. 从弹出菜单选择Open Specification, 出现角色规范窗口。
- 或
1. 右单击浏览器中的角色。
 2. 从弹出菜单选择Open Specification, 出现角色规范窗口。
- 或
1. 选择Use Case框图中的角色。
 2. 选择Browse Specification, 出现角色规范窗口。
- 或
1. 选择Use Case框图中的角色。
 2. 按Ctrl+B, 出现角色规范窗口。

角色规范窗口中的大多数标签页面适用于类, 但不适用于角色。包括角色信息的标签页面有General标签、Detail标签、Relations标签和Files标签。这些标签中的有些选项只适用于类。下面介绍适用于角色的选项。

命名角色

每个角色应有唯一名称。可以用角色规范窗口命名角色, 也可以在Use Case框图或浏览器中直接输入名称。

要命名角色:

1. 右单击Use Case框图或浏览器中的角色。
2. 从弹出菜单选择Open Specification。
3. 在Name字段中输入角色名。

或

1. 在Use Case框图或浏览器中选择角色。
 2. 输入角色名。
- 要将文档加入角色：
1. 选择浏览器中的角色。
 2. 在文档窗口中输入角色说明。
- 或
1. 右单击Use Case框图或浏览器中的角色。
 2. 从弹出菜单选择Open Specification。
 3. 在规范窗口的Documentation区输入角色说明。

指定角色版型

和使用案例一样，可以在规范窗口指定角色的版型。但如果改变角色版型，则Rose改变Use Case框图中表示角色的图标。Rose不是用角色符号，而是用表示类的标准矩形。

除了Actor外，角色没有其他可用的版型。但可以定义自己的角色版型并在自己的Rose模型中使用。

要指定角色版型：

1. 右单击Use Case框图或浏览器中的角色。
2. 从弹出菜单选择Open Specification。
3. 在Stereotype字段中指定角色版型。

警告：如果改变角色版型，则Rose显示角色时不再用UML角色符号，而是用表示类的标准矩形。

设置角色基数

可以在Rose中指定某个角色要有多少实例。例如，可能要知道许多人起客户角色的作用，而只有一人起管理者角色的作用。这可以用基数字段表示。

Rose提供几个基数选项：

Cardinality (基数)	含义
n (缺省)	多
0..0	0
0..1	0或1
0..n	0或多
1..n	1
1..n	1或多

也可以用下列格式输入自己的基数：

格式	含义
<number>	<number>
<number 1>..<number 2>	<number 1>到<number 2>
<number>..n	<number>或多
<number 1>,<number 2>	<number 1>或<number 2>
<number 1>,<number 2>..<number 3>	<number 1>或<number 2>到<number 3>
<number 1>..<number 2>,<number 3>..<number 4>	<number 1>到<number 2>或<number 3>到<number 4>

要设置角色基数：

1. 右单击Use Case框图或浏览器中的角色。
2. 从弹出菜单选择Open Specification。
3. 选择Detail标签。
4. 从Cardinality下拉清单框选择或按上表所列格式输入角色基数。

生成抽象角色

抽象角色是没有实例的角色。换句话说，角色基数等于0。例如，你可能有几种角色：计时工、合同工和临时工。所有这些都属于第四种角色——员工。但公司中没有一个人是单纯的员工，每个人都是计时工、合同工和临时工中的一种。员工角色只是显示计时工、合同工和临时工的一些共性。员工角色没有实例，是个抽象角色。图3.14显示了抽象角色的例子。

关于角色间箭头的画法，见本间稍后关系一节。

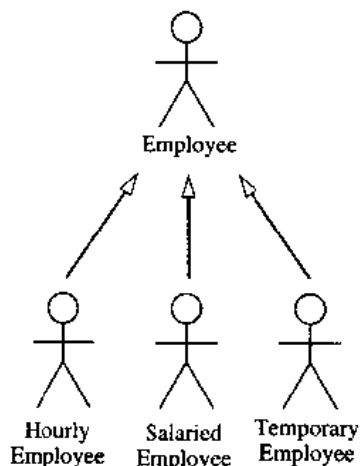


图3.14 抽象角色

要生成抽象角色：

1. 在浏览器或Use Case框图中生成角色。
2. 右单击Use Case框图或浏览器中的角色。
3. 从弹出菜单选择Open Specification。
4. 选择Detail标签。
5. 选择Abstract框。

浏览角色关系

角色规范窗口的Relations标签列出角色参与的所有关系。图3.15是角色规范窗口的Relations标签。这个标签包括角色与使用案例和其他角色的所有关系。清单包括关系名和参与关系的角色与使用案例。

从这个标签可以浏览、增加或删除关系。下面介绍浏览、增加或删除关系的过程。

要浏览角色关系：

1. 右单击Use Case框图或浏览器中的角色。
2. 从弹出菜单选择Open Specification。Relations标签列出角色参与的关系。



图3.15 角色规范窗口的Relations标签

要浏览器关系规范:

1. 双击清单中的关系。
2. 出现关系规范窗口。下面将详细介绍关系规范。

或

1. 右单击清单中的关系。
2. 选择弹出菜单中的Specification。
3. 出现关系规范窗口。下面将详细介绍关系规范。

要删除关系:

1. 右单击清单中的关系。
2. 选择弹出菜单中的Delete。

将文件和URL连接角色

和使用案例一样, 可以将文件和URL连接角色。这是用角色规范窗口的Files标签进行的, 如图3.16。连接角色的文件或URL应为只适用于该角色的文档。例如, 可能有个工作流程图, 演示角色的责任和工作流程。如果角色是另一系统, 则可能要连接一些与这个外部系统相关的文档或规范。

连接角色的文件或URL出现在浏览器中该角色之下。一旦加入之后, 可以在浏览器中双击连接角色的文件或URL, 启动相应应用程序, 装入文件或URL。

要将文件连接角色:

1. 右单击Files标签中任一空白处。
2. 选择Insert Files插入文件。
3. 用对话框找到要连接的文件。
4. 选择Open将文件连接角色。所连接文件的名称和路径出现在Files标签中, 并出现在浏览器的角色下面。

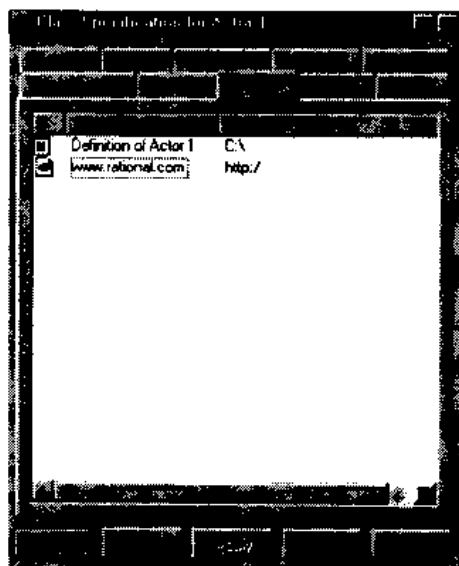


图3.16 角色规范窗口Files标签

要连接URL与角色:

1. 右单击Files标签中任一空白处。
2. 选择Insert URL插入URL。
3. 输入要连接的URL。URL名出现在浏览器的角色。

要打开连接的文件:

1. 打开角色规范。
 2. 双击Files标签中的文件名。Rose自动启动相应应用程序并装入文件。
- 或

1. 在浏览器的相应角色下面找到文件。
2. 双击Files标签中的文件名。Rose自动启动相应应用程序并装入文件。

或

1. 打开角色规范。
2. 右单击Files标签中的文件。
3. 从弹出菜单选择Open File/URL。Rose自动启动相应应用程序并装入文件。

或

1. 在浏览器的相应角色下面找到文件。
2. 右单击浏览器中的文件。
3. 从弹出菜单选择Open。Rose自动启动相应应用程序并装入文件。

要打开连接的URL:

1. 打开角色规范。
2. 双击Files标签中的URL名。Rose自动启动相应Web浏览器应用程序并装入URL。

或

1. 在浏览器的相应角色下面找到URL。
2. 双击Files标签中的URL名。Rose自动启动相应Web浏览器应用程序并装入URL。

或

1. 打开角色规范。
2. 右单击Files标签中的URL。
3. 从弹出菜单选择Open File/URL。Rose自动启动相应Web浏览器应用程序并装入URL。

或

1. 在浏览器的相应使用案例下面找到URL。
2. 右单击浏览器中的URL。
3. 从弹出菜单选择Open。Rose自动启动相应Web浏览器应用程序并装入URL。

要删除连接的文件或URL：

1. 右单击浏览器中的文件或URL。
2. 从弹出菜单选择Delete。

浏览角色实例

造型一个系统时，可能要知道特定角色所在的Sequence和Collaboration框图。Rose通过Report菜单提供这个功能。要浏览角色所在的Sequence和Collaboration框图：

1. 选择Use Case框图中的角色。
2. 选择Report►Show Instances。
3. Rose列出角色所在的Sequence和Collaboration框图，如图3.17。要打开框图，在列表框图中双击或按Browse按钮。

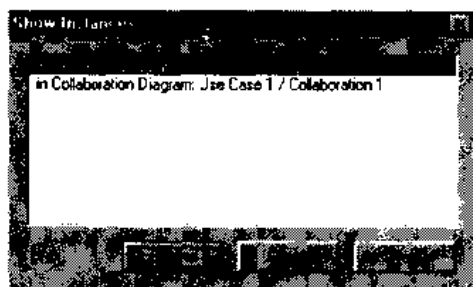


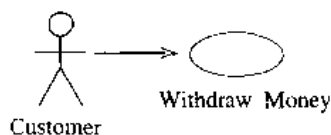
图3.17 浏览角色实例

使用关系

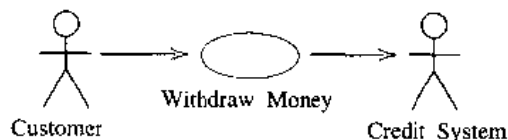
UML对使用案例和角色支持几种类型的关系，包括：通信关系、使用关系、扩展关系和角色一般化关系。通信关系描述角色与使用案例之间的关系。使用关系和扩展关系描述使用案例之间的关系。角色一般化关系描述角色之间的关系。

通信关系

通信关系（communicates relationship）描述角色与使用案例之间的关系。在UML中，通信关系是使用箭头表示的：



关系中的箭头表示启动通信者。上例中客户启动与系统的通信，运行取钱（Withdraw Money）功能。使用案例也可启动与角色的通信：



本例中，使用案例启动与Credit System角色的关系。运行付款（Make Payment）使用案例时，ATM系统启动与Credit System的通信，完成这个事务。信息流从ATM系统到信用系统，又从信用系统流回ATM系统，箭头只表示系统通信。

每个使用案例都应由角色启动，除下面介绍的使用关系和扩展关系中的使用案例。

要增加通信关系：

1. 选择单向关联（Unidirectional Association）工具栏按钮。
2. 将鼠标从角色拖动到使用案例。
3. Rose画出使用案例与角色之间的关系。

要删除通信关系：

1. 选择Use Case框图中的关系。
2. 选择Edit►Delete from Model或按Ctrl+D。

使用关系

使用关系（uses relationship）使一个使用案例可以利用另一使用案例提供的功能。使用关系通常用于造型一些两个或多个使用案例共同的可复用功能。在ATM例子中，Withdraw Money和Deposit Funds使用案例都需要验证客户及其PIN之后才能进行事务。与其在这两个使用案例中详细描述验证过程，不如将这个功能移到另一使用案例中，称为Authenticate Customer。另一使用案例需要验证客户时，可以利用Authenticate Customer中的功能。

UML中将使用关系显示为箭头和<<uses>>字样，如图3.18。

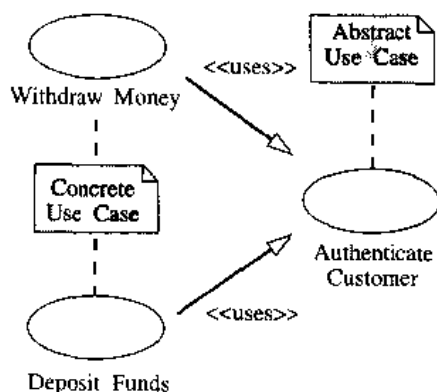


图3.18 使用关系

本例中，Withdraw Money和Deposit Funds使用案例都使用Authenticate Customer中的功能。Authenticate Customer使用案例是个抽象使用案例。抽象使用案例通过使用或扩展关系向其他使用案例提供功能。Withdraw Money和Deposit Funds使用案例是具体使用案例。

使用关系中一个使用案例总是利用另一使用案例的功能。不管Withdraw Money如何进行，Authenticate Customer使用案例总是执行。相反，扩展关系则允许一个使用案例（可选）扩展另一使用案例提供的功能。

要增加使用关系：

1. 选择一般化（Generalization）工具栏按钮。
2. 将一个使用案例拖动到被利用的使用案例（从具体使用案例到抽象使用案例）。
3. Rose在两个使用案例画一个一般化关系。
4. 右单击关系线并选择Open Specification，如图3.19。

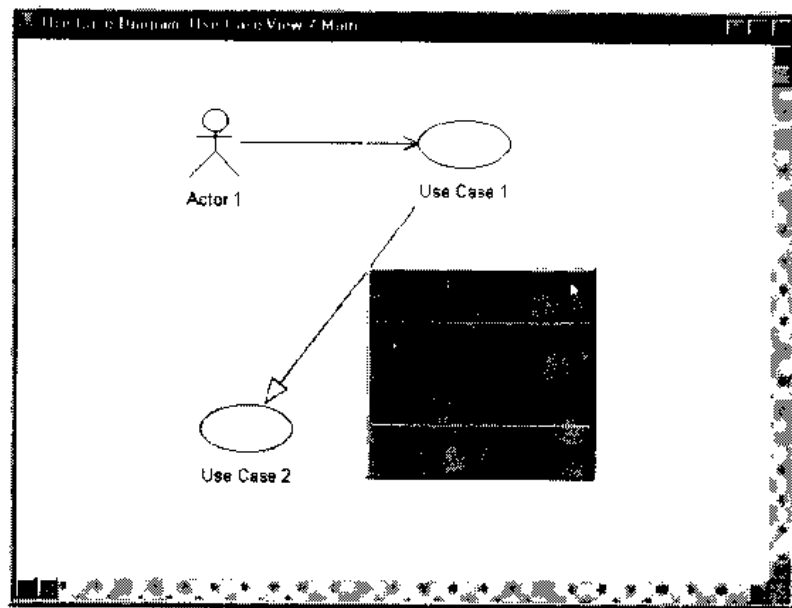


图3.19 打开一般化规范

5. Rose打开一般化规范，如图3.20。

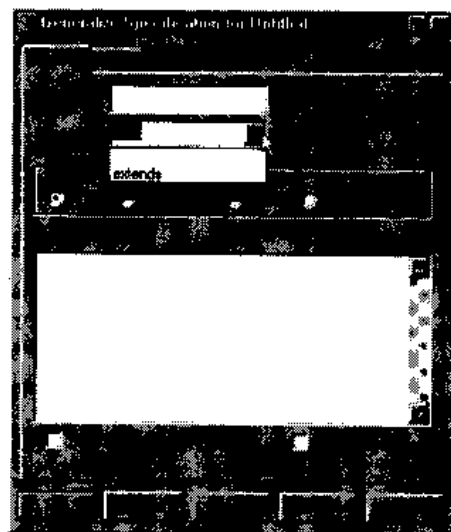


图3.20 一般化规范

6. 在Stereotype下拉列表框中, 选择uses或输入uses (如果没有)。输入之后, 它即放在下拉清单中, 可供今后选择。

说明: 规范窗口中还有几个其他选项, 它们不适用于使用关系, 见第7章。

7. 单击OK关闭规范窗口。
8. 一般化箭头上出现<<uses>>字样。如果没有, 右单击关系线并确保复选Stereotype Label, 如图3.21。
9. 打开抽象使用案例的使用案例规范窗口。
10. 复选Abstract框。

要删除使用关系:

1. 选择Use Case框图中的关系。
2. 选择Edit►Delete from Model或按Ctrl+D。

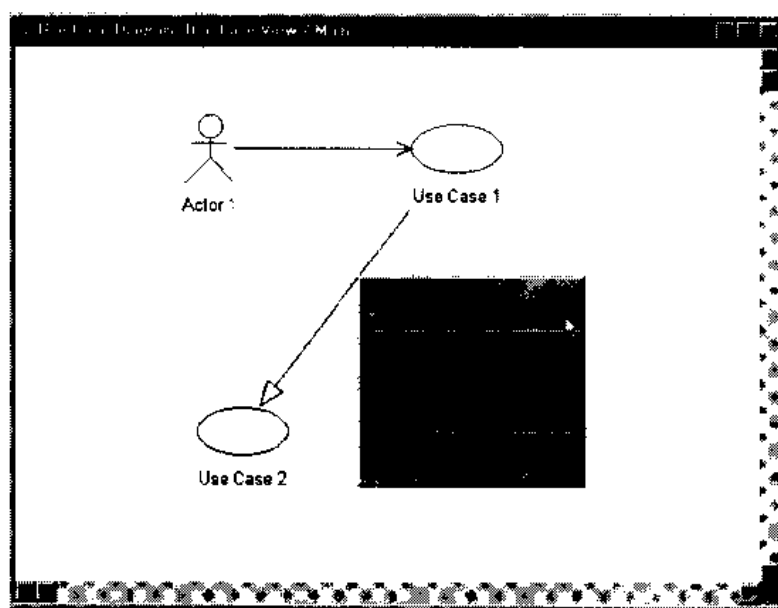


图3.21 确保复选Stereotype Label

扩展关系

扩展关系 (extends relationship) 允许一个使用案例 (可选) 扩展另一使用案例提供的功能。它与使用关系相似, 这两个关系都是把共同功能分离到另一个使用案例中。

在UML中, 扩展关系表示为箭头加<<extends>>字样, 如图3.22。

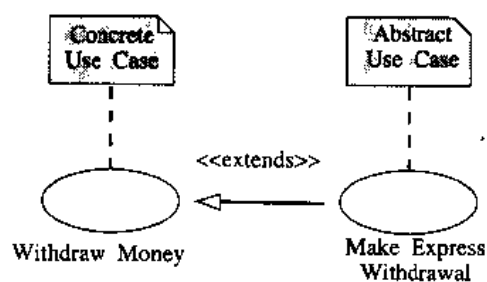


图3.22 扩展关系

本例中，Withdraw Money使用案例有时使用Make Express Withdrawal使用案例中的功能。如果用户在Withdraw Money使用案例中选择40美元快取选项，则运行Make Express Withdrawal使用案例。

Make Express Withdrawal使用案例提供扩展功能，是个抽象使用案例，而Withdraw Money使用案例是个具体使用案例。

要增加扩展关系：

1. 选择一般化（Generalization）工具栏按钮。
2. 将一个使用案例拖动到被扩展的使用案例（从具体使用案例到抽象使用案例）。
3. Rose在两个使用案例画一个一般化关系。
4. 右单击关系线并选择Open Specification，如图3.19。
5. Rose打开一般化规范，如图3.23。

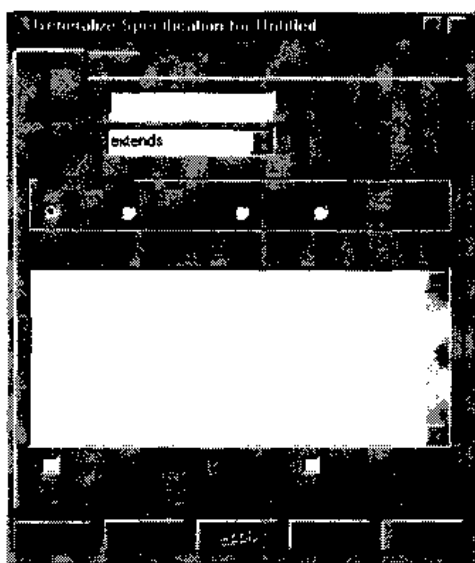


图3.23 一般化规范

6. 在Stereotype下拉列表框中，选择extends或输入extends（如果没有）。输入之后，它即放在下拉清单中，可供今后选择。

说明：规范窗口中还有几个其他选项，它们不适用于扩展关系，见第7章。

7. 单击OK关闭规范窗口。
8. 一般化箭头上出现<<extends>>字样。如果没有，右单击关系线并确保复选Stereotype Label，如图3.21。
9. 打开抽象使用案例的使用案例规范窗口。
10. 复选Abstract框。

要删除扩展关系：

1. 选择Use Case框图中的关系。
2. 选择Edit►Delete from Model或按Ctrl+D。

角色一般化关系

角色一般化关系 (actor generalization) 表示几个角色有一些共性。例如, 可能有两个客户: 公司客户和个人客户。可以用图3.24所示的图注表示这种关系。

这个框图表示有两个客户: 公司客户和个人客户。个人和公司角色是具体角色, 可以直接实例化。客户角色是抽象角色, 无法直接实例化, 只表示有两种客户。

如果需要, 还可以进一步分解。假设有两种公司客户——政府机构和私人公司, 则可以将框图修改成图3.25。

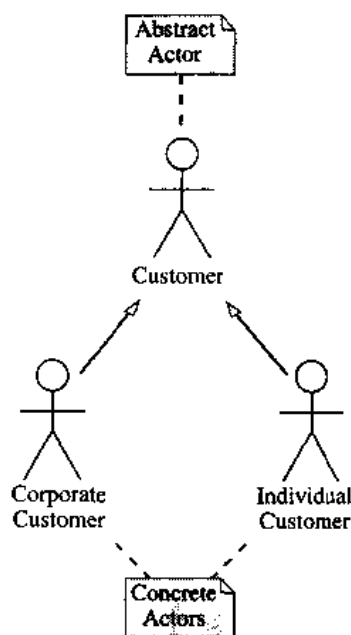


图3.24 角色一般化关系

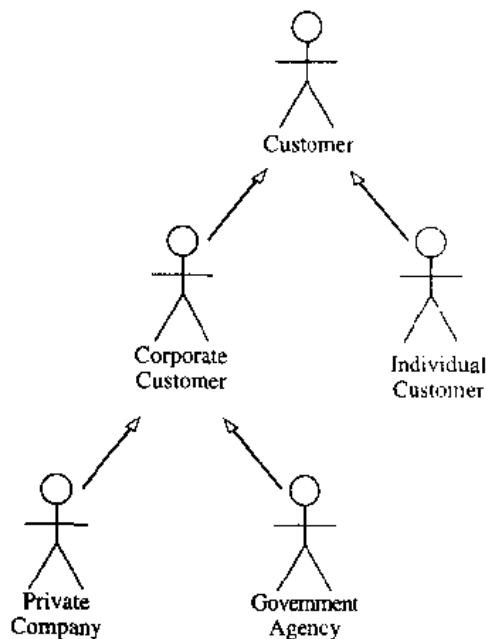


图3.25 修改后的角色一般化关系

这种关系并非总是必需的。一般来说, 只有在系统认为一种角色与另一种角色的表现不同时才需要。如果公司客户启动一些个人客户不启动的使用案例, 则最好进行角色一般化。如果两种客户使用相同的使用案例, 则没必要进行角色一般化。如果两种客户使用相同的使用案例只是稍有不同, 则仍然没必要进行角色一般化。细微差别可以在使用案例的事件流文档中表示。

这些框图的作用仍然是通信。如果角色一般化能使小组得到有用的信息, 则进行角色一般化, 否则就没必要进行角色一般化。

要增加角色一般化:

1. 将角色加进Use Case框图。
2. 选择Generalization工具栏按钮。
3. 从具体角色拖动到抽象角色。
4. 对抽象角色打开角色规范窗口。
5. 选择Detail标签。
6. 复选Abstract框。

要删除角色一般化关系：

1. 选择Use Case框图中的关系。
2. 选择Edit►Delete from Model或按Ctrl+D。

使用图注说明

生成框图时，可以给使用案例和角色加上图注说明。例如，可能要标明哪个角色与哪个使用案例交互，为什么一个使用案例参与使用或扩展关系，为什么一个角色继承另一角色。Rose可以通过Note工具栏按钮把这种图注说明加进框图中。

增加图注

可以加进的图注有两种：说明和文本框。一般来说，说明适用于框图上的一个项目，而文本框在框图中放上框图名或其他一般信息。

要在框图中增加图注：

1. 选择Note工具栏按钮。
2. 单击框图中任一位置放上图注。
3. 在新图注仍然选择时，输入图注文本。

要将图注与项目相连接：

1. 选择Anchor Note to Item工具栏按钮。
2. 将图注拖动到相关联的使用案例或角色。Rose图注与相关联的使用案例或角色之间画一条虚线，如图3.26。

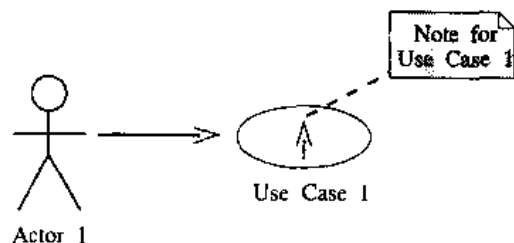


图3.26 将图注与相关联的使用案例或角色相连接

除了说明外，还可以增加文本框。例如可以给Use Case框图加一个标题。

要给框图增加文本框：

1. 选择Text Box工具栏按钮。
2. 单击框图中任一位置放上文本框。
3. 在文本框仍然选择时，输入图注文本。

删除图注

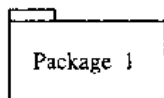
改变或更新框图时，可能要删除一些说明和文本框。

要删除框图的图注说明和文本框：

1. 选择框图的图注说明和文本框。
2. 按Delete键。

使用包

在UML中，角色、使用案例、类和组件等项目可以组成包以便组织。在Use Case视图中，可以将角色与使用案例组成包。在UML中，包用下列符号表示。



增加包

在Rose中，可以增加多个包，以便在模型中组织项目。如果愿意，也可以在一个包内生成另一个包，通过嵌套进一步管理模型。生成新包时，Rose自动生成Package Overview框图和关联表（Associations list）。

要增加包：

1. 右单击浏览器中的Use Case视图。要在现有包内生成另一个包，右单击浏览器中的现有包。
2. 选择New►Package，如图3.27。
3. 输入新包名。

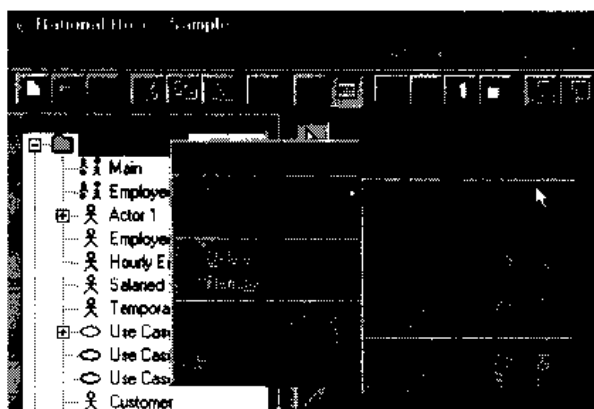


图3.27 增加使用案例包

要将项目移进包中：

在浏览器窗口中，将项目从现有位置拖动到新包中。

删除包

在Rose中，可以从一个Use Case框图或从整个模型删除包。如果从整个模型删除包，则包及其其中的一切均被删除。

要从Use Case框图删除包：

1. 选择Use Case框图中的包。
2. 按Delete键。
3. 注意这个包从Use Case框图删除，但在浏览器和其他Use Case框图中仍然存在。

要从整个模型删除包:

1. 右单击浏览器中的包。
2. 从弹出菜单选择Delete

或

1. 选择Use Case框图中的包。
2. 选择Edit►Delete from Model或按Ctrl+D。

警告: 从整个模型删除包时, 包中的所有角色、使用案例和其他项目均从模型中删除。

练习

这些练习是本部分每一章的最后一节。本章练习介绍生成订单处理系统的Use Case框图。

问题

“我明白了,” Bob放下电话说。

Mary从计算机上移开目光问, “什么事?”

“这是客户第四次抱怨没收到订单, 再这么下去, 我们要关门了”。

“冷静些”, Mary答道, “我们的业务增长太快了。处理书面订单只能适应5个人的小公司, 根本适应不了目前的情况。还是与Susan谈谈, 看看能否建立一个系统, 帮助我们跟踪订单信息”。

Robertson's Cabinets, Inc是个假想的小公司, 专门制造标准和定制厨房用具。三年前刚开业时, 业务量不太, 用书面订单跟踪就够了。但随着名气越来越大, 订单也越来越多, 他们开始增加员工, 如今已经是个有50多名员工的制造厂。

尽管公司仍然不算太大, 但手工处理订单已经很吃力了。Bob和Mary Robertson是公司老板, 决定与Susan商量办法。Susan是几个计算机专家之一, 在公司信息技术组。

Bob对Susan说:

“我们显然需要某种系统来跟踪订单, 否则有失去顾客的危险”。

“没错”。

“能否用一个Java程序来跟踪订单?”

“先别考虑怎么实现, 还是考虑系统的要求吧”。

“用来跟踪订单”。

“能否更具体些? 先看看目前的过程吧”。

“好, 收到电话时, 我们填一份订单, 把订单交给仓库的Clint, 他会填好单据, 安排向客户发货。然后我们把另一份订单交给会计部的Don, 他把其输入会计系统, 产生发票”。

“你是否要让新系统支持这整个过程。”

“是的”。

通过这个谈话, Susan可以确定系统要支持增加新订单、修改现有订单、填单、检查当前库存和补货。增加新订单时, 系统要通知会计部门, 以便生成发票。如果项目缺货, 则系统要等待该项目的订单。Susan由此建立了造型这个系统的Use Case框图。

生成Use Case框图

生成订单处理系统的Use Case框图。生成框图的步骤如下，最后的Use Case框图如图3.28所示。

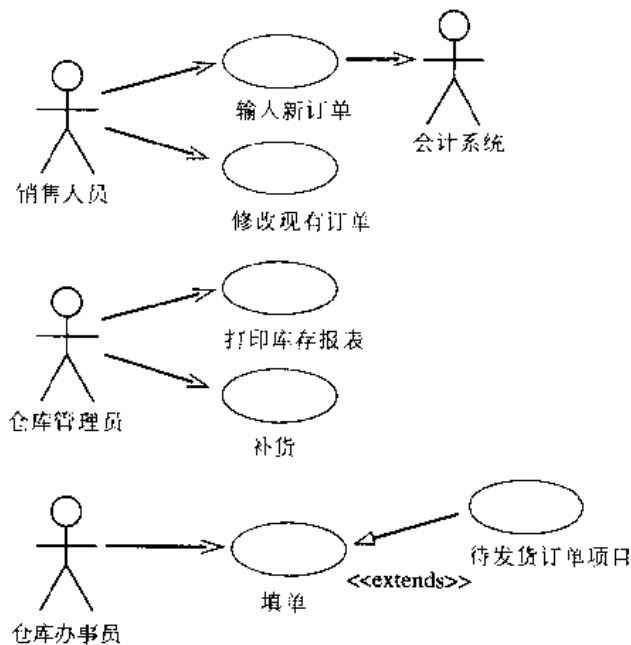


图3.28 订单处理系统的Use Case框图

练习步骤

增加Use Case框图、使用案例和角色

1. 双击浏览器中的主Use Case框图打开框图。
2. 用Use Case工具栏按钮将新的使用案例加进框图。
3. 将新的使用案例取名为Enter New Order。
4. 重复第2步和第3步，将其余使用案例加进框图：

Modify Existing Order
Print Inventory Report
Restock Inventory
Fill Order
Backorder Item

5. 用角色工具栏按钮将新的角色加进框图。
6. 将新角色取名为Salesperson。
7. 重复第5步和第6步，将其余角色加进框图：

Warehouse Manager
Warehouse Clerk
Accounting System

标出抽象使用案例

1. 右单击框图中的Backorder item使用案例
2. 从弹出菜单选择Open Specification选项。
3. 复选Abstract框，将这个使用案例标为抽象使用案例。

增加关联

1. 用单向关联（Unidirectional Association）工具栏按钮画出Salesperson角色与Enter New Order使用案例之间的关联。
2. 重复第1步，将其余关联加进框图中。

增加扩展关系

1. 用Generalization工具栏按钮画出Backorder item使用案例与Fill Order使用案例之间的扩展关系，从Backorder item画到Fill Order。扩展关系表示Backorder item可以扩展Fill Order提供的功能。
2. 右单击Backorder item与Fill Order之间的新关系。
3. 从弹出菜单中选择Open Specification。
4. 在stereotype下拉列框中，输入extends并单击OK。
5. 关系线上出现<<extends>>字样。

增加使用案例说明

1. 在浏览器中选择Enter New Order使用案例。
2. 用文档窗口给Enter New Order使用案例加上如下说明：This use case will allow a user to add a new order into the system.
3. 用文档窗口给其余使用案例加上说明。

增加角色说明

1. 选择浏览器中的Salesperson角色。
2. 用文档窗口给Salesperson角色加上下列说明：A salesperson is an employee who markets and intends to sell products.
3. 用文档窗口给其余角色加上说明。

将文件连接使用案例

1. 本书选配光盘上有文件OrderFlow.doc，包含Enter New Order使用案例的主事件流。利用这个文件，或者输入图3.29所示主事件流到文件OrderFlow.doc中
2. 右单击Enter New Order使用案例。
3. 从弹出菜单选择Open Specification。
4. 选择Files标签。
5. 右单击空白区并从弹出菜单选择Insert File。
6. 找到文件OrderFlow.doc并单击Open按钮将其连接使用案例。

Primary Flow of Events for 'Enter New Order' Use Case

1. The salesperson selects the 'Create New Order' option from the options menu.
2. System displays the Order Detail form.
3. Salesperson enters the order number, customer, and items ordered
4. Salesperson saves the order.
5. System creates a new order and saves it to the database.

图3.29 Enter New Order使用案例的主事件流

小结

本章介绍了如何使用使用案例、角色和Use Case框图。所建系统的要求组成所有使用案例和角色设置。首先生成主Use Case框图，显示系统的总体视图。然后可以生成其他框图，演示使用案例与角色间的交互。使用案例可以使用和扩展其他使用案例。此外使用案例之间不能直接通信。一个使用案例总是需要另一使用案例的功能时，使用使用案例。一个使用案例可选利用另一使用案例的功能时，扩展使用案例。如果一个使用案例使用或扩展另使用案例，则被利用者是抽象使用案例。直接由角色参与的使用案例是具体使用案例。

角色可以和使用案例通信，演示哪个角色参与哪个使用案例。角色还可以相互继承。例如，学生可能是系统中的角色，我们要进一步将学生分为全日制和半日制学生。这时全日制和半日制学生都继承学生角色。

使用案例和Use Case框图可以很好地描述系统功能。下一章介绍Sequence和Collaboration框图的用法，显示对象与角色间的交互。

第4章 对象交互

- Sequence和Collaboration框图
- 向Sequence和Collaboration框图中增加对象
- 使用案例消息和Sequence、Collaboration框图
- Sequence和Collaboration框图间切换
- 用两步法生成Interaction框图

本章介绍如何建模系统对象之间的交互。Interaction框图分两种：Sequence和Collaboration框图。两者都显示参与使用案例流程的对象和对象之间发送的消息。Sequence框图按时间排序，Collaboration框图按对象本身组织。

本章末尾的练习要建立一个Sequence框图，支持订单处理系统Enter New Order使用案例的流程。

Interaction框图

Interaction框图一步一步显示使用案例的流程。在ATM例子中，Withdraw Money使用案例有几个替换流程。因此，这个使用案例要有几个Interaction框图。我们显示流畅的流程，即一切顺利时的Interaction框图。而其他流程如显示输入错误PIN时、帐号中没有足够的钱可取时等情况下的流程则不显示。

我们介绍两种Interaction框图：Sequence和Collaboration。Sequence框图按时间排序，图4.1是一个Sequence框图。

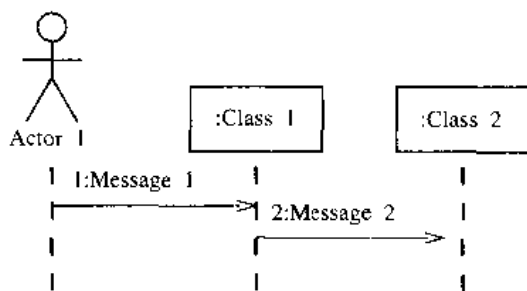


图4.1 Sequence框图

Collaboration框图显示同一信息，但组织方式不同。图4.2是一个Collaboration框图。

Sequence和Collaboration框图显示同一信息，但组织方式不同。Sequence框图可以显示控制焦点，Collaboration框图可以显示数据流。下面介绍消息的时候会介绍这些差别。

Interaction框图包含事件流中的许多相同细节，但这里的信息表示成对开发人员更好用的方式。这些框图关注实现使用案例功能时要生成的对象。Sequence和Collaboration框图可以显示对象、类或两者都显示。

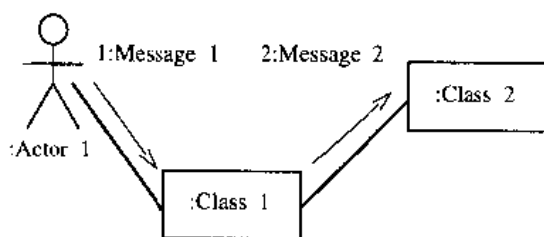


图4.2 Collaboration框图

何谓对象

我们周围到处都是对象。我们坐的椅子、我们读的书、书桌上的白炽灯都是对象。软件中的对象也差不多。

对象包装信息和功能，表示具体、实际的事项。例如：

- Joe的帐目
- Main大街7638号的房子
- 窗前的黄花

在ATM例子中，ATM屏幕、读卡机、Joe的ATM卡和Joe取钱后打出的收据都是对象。

每个对象都包括信息和功能。例如，Joe帐目有一些信息：帐号为123456789，户主为Joe Smith，目前结余为350美元。Joe的帐号还有一些功能，它知道如何在Joe存钱时加钱，在Joe取钱时减钱，如何判断是否透支。

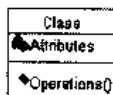
对象保存的信息称为属性。尽管属性值随着时间改变（Joe的帐号下周可能有400美元），但属性本身不变。Joe帐目总是有帐号名、持有者和结余。

对象的功能称为操作。本例中Joe帐号的操作包括调整存款或取款结余和检查帐号是否透支。

在Rose中，对象加进Interaction框图。拖动角色（类版型）或其他类到Interaction框图时，自动生成这个类的对象实例。Rose中删除框图内的对象并不从模型中删除这个类。

何谓类

类是提供对象蓝图的项目。换句话说，类定义对象持有的信息类型和对象的操作。例如，Joe支票帐号的类、Main大街7638号房子的类和窗前黄花的类分别为：帐号、房子和花。房子类指定房子有高、宽、房间数和面积。Main大街7638号房子对象的高为20英尺、宽为60英尺、10间房、面积为2000平方英尺。类是提供对象模板的一般术语。



可以把类看成房子的蓝图，对象是从蓝图建成的房子。下一章将详细介绍类。

寻找对象

研究事件流中的名词是寻找对象的好办法，另一个办法是查阅情境文档。情境是事件流的特定实例。我们的Withdraw Money使用案例的事件流介绍一个人从ATM取钱。一种情

境是Joe取20美元；另一种是Jane想取20美元，但PIN有错；另一种是Bob想取20美元，但帐上没那么多钱。事件流通常有许多种情境。Sequence或Collaboration框图显示其中一种情境。

研究情境时，有些名词是角色，有些是对象，有些是对象属性。建立Interaction框图时，名词介绍对象是什么。如果不知道一个名词是对象还是对象属性，可以看看它是否有任何行为。如果是单纯的信息，则可能是属性，如果还有一些行为，则可能是对象。

并非所有对象都在事件流中。例如，表单可能不出现在事件流中，但必须放在框图中，角色才能输入或浏览信息。其他不在事件流中的对象是控制对象。这些对象控制使用案例中流程的顺序。下一章将详细介绍类和控制对象。

使用Interaction框图

通过框图，设计人员和开发人员可以确定需要开发的类，类之间的关系和每个类的操作或责任。Interaction框图是设计工作的奠基石。

Sequence框图按时间排序，用于通过情境检查逻辑流程。尽管Collaboration框图包括顺序信息，但从Sequence框图更容易看出。

Collaboration框图用于了解改变的影响，从Collaboration框图很容易看出哪个对象与哪个对象通信。如果要改变对象，就可以方便地看到受影响的其他对象。

Interaction框图包含：

对象：Interaction框图可以使用对象名、类名或两者。

消息：通过消息，一个对象或类可以请求另一对象或类完成特定功能。例如，窗体可能要求报表对象打印。

生成Interaction框图时要记注，你是在向对象指定责任。将消息加进Interaction框图时，是在向接收消息的对象指定责任。一定要向适当对象指定适当责任。在大多数应用程序中，屏幕和窗体不应进行业务处理。它们只能让用户输入和浏览信息。通过分开前端与业务逻辑，生成的结构可以减少改变对效果的影响。如果业务逻辑需要改变，不应影响界面。如果改变一个或两个屏幕的格式，也不应影响业务逻辑。其他对象也要指定相应责任。如果要打印所有帐号的所有结余报表，则不能让Joe的帐号负责。Joe帐号只负责Joe自己的钱。另一对象可以负责查找所有帐号，生成报表。

Sequence框图

下面首先介绍Sequence框图。Sequence框图是按时间排序的Interaction框图，从上往下阅读每个使用案例。前面曾介绍过，每个使用案例有几个流程。每个Sequence框图表示使用案例的一个流程。例如，图4.3是银行客户Joe从ATM取20美元时的Sequence框图。

首先看看这个框图中的对象和消息。参与流程的对象在框图顶部的矩形中显示。本例中有五个对象：Joe、读卡机、ATM屏幕、Joe的帐目和取款机。角色对象Joe启动这个使用案例，如框图左上角所示。

过程开始时，Joe向读卡机插卡。读卡机读取Joe的卡号。然后让ATM屏幕初始化。ATM提示Joe输入PIN。Joe输入PIN（1234），ATM打开他的帐目。Joe的PIN有效，因此ATM提示选择事务。Joe选择取钱。ATM提示Joe输入金额，Joe输入20美元。ATM验证Joe帐上有足够的钱，就从帐上减去20美元。ATM取出20美元，并退出Joe的卡。

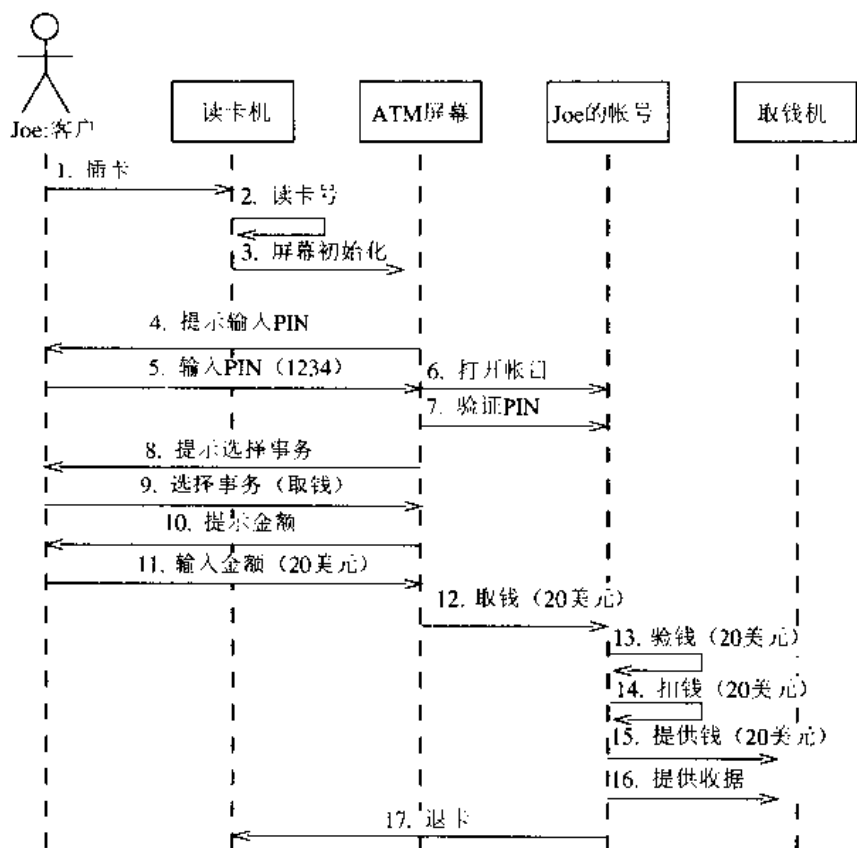


图4.3 Joe从ATM取20美元时的Sequence框图

每个对象有个生命线，画成对象下面的竖虚线。两个对象的生命线之间画一个消息，显示对象通信。每个消息表示一个对象向另一对象进行函数调用。后面定义类操作的过程中，每个消息会成为操作。消息也可以反过来表示对象调用自己的操作。本例中，消息2表示读卡机让自己读卡号。

生成Sequence框图

Sequence框图可以在浏览器的Use Case或Logical视图中生成。Sequence框图应在使用案例之内，如图4.4，或直接放在包中。

要生成Use Case或Logical视图：

1. 右单击浏览器中的使用案例或包。
2. 从弹出菜单中选择New►Sequence diagram。
3. 将新Use Case或Logical视图命名。
4. 双击打开浏览器中的新Sequence框图。

要打开现有Sequence框图：

1. 在浏览器的Use Case视图找到Use Case或Logical视图。
2. 双击打开Sequence框图。

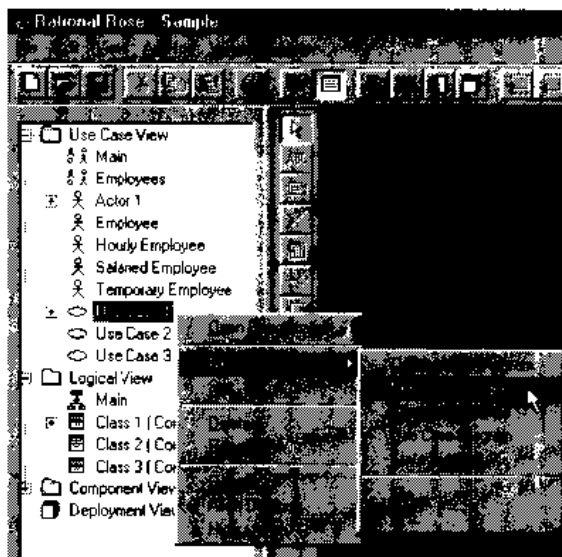


图4.4 生成Use Case或Logical视图

或

1. 选择**Browse►Interaction diagram**，出现图4.5所示窗口。
2. 在Package列表框中，选择要打开的框图所在包。
3. 在Interaction diagrams列表框中，选择要打开的框图。
4. 按OK。

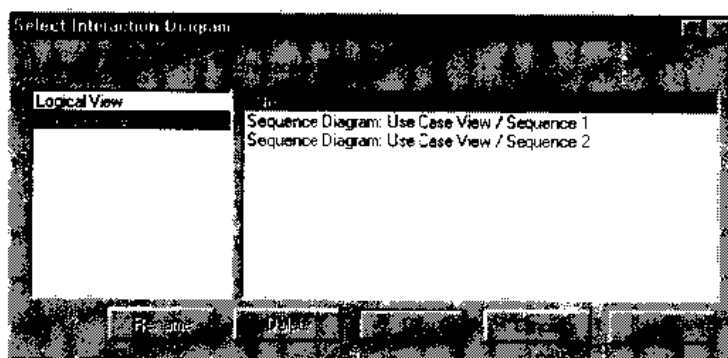


图4.5 打开现有Sequence框图

要将项目加进Sequence框图中：

用下节介绍的工具栏按钮将对象和消息加进框图中。

或

将对象或类从浏览器中拖动到Sequence框图中。

要删除Sequence框图中的项目：

1. 选择Sequence框图中的项目。
2. 选择**Edit►Delete from Model**或按Ctrl+D。

说明：从框图中删除对象并不从模型中删除相应的类。

删除Sequence框图

继续进行项目时，你可能发现有些Sequence框图是过时的或多余的。要让模型保持一致，最后删除不再使用或不再反映系统设计的Sequence框图。利用Rose中的浏览器可以从模型中删除Sequence框图。

要删除Sequence框图：

1. 右单击浏览器中的Sequence框图。
2. 从弹出菜单中选择Delete。

将文件和URL连接Sequence框图

在Rose中，可以将文件和URL连接。例如文档描述Interaction框图模型的情形。可以连接一个文件，该文件包含实现框图逻辑的一些代码。或者连接要求文档，其中包含针对该框图的一些要求。

无论连接哪个文档，一定要保证文件或URL中的信息只与该Interaction框图有关，而不是与整个使用案例有关。如果与整个使用案例有关，则应连接使用案例本身。

要将文件连接Sequence框图：

1. 右单击浏览器中的Sequence框图。
2. 从弹出菜单中选择New►File。
3. 用Open对话框选择要连接的文件。
4. 选择Open连接文件。

要将URL连接Sequence框图：

1. 右单击浏览器中的Sequence框图。
2. 从弹出菜单中选择New►URL。
3. 输入要连接的URL名。

要打开连接的文件：

双击浏览器中的文件名。Rose打开相应应用程序并装入文件。

或

1. 右单击浏览器中的文件。
2. 从弹出菜单选择Open。Rose打开相应应用程序并装入文件。

要打开连接的URL名：

双击浏览器中的URL名。Rose打开相应应用程序并装入URL。

或

1. 右单击浏览器中的URL。
2. 从弹出菜单选择Open。Rose打开相应应用程序并装入URL。

要删除连接的文件或URL：

1. 右单击浏览器中的文件或URL。
2. 从弹出菜单中选择Delete。

Sequence框图工具栏

打开Sequence框图时,框图工具栏变成可以向框图中增加对象、消息和其他项目。工具栏中的项目如下。下面几节介绍如何增加这些项目。表4.3列出了Sequence框图工具栏中的按钮并一一介绍其作用。

表4.1 Sequence框图工具栏图标

图标	按钮	用途
	Selects or deselects an item	将光标返回箭头以选择项目
	Text Box	将文本框加进框图
	Note	将图注加进框图
	Anchor Note to Item	将图注连接框图中的项目
	Object	将新对象加进框图
	Object Message	在两个对象之间绘制消息
	Message to Self	画出反身消息

Collaboration框图

和Sequence框图一样, Collaboration框图也显示使用案例中特定情形的流程。Sequence框图按时间排序, 而Collaboration框图则着重于对象之间的关系。图4.6是Joe取20美元的Collaboration框图。

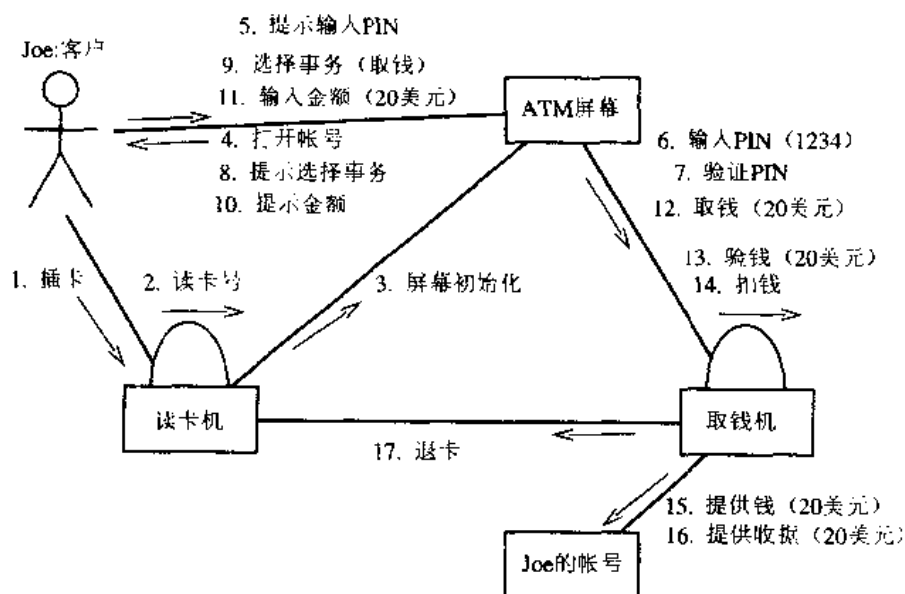


图4.6 Joe取20美元的Collaboration框图

可以看出, Collaboration框图与Sequence框图中的信息相同, 但Collaboration框图显示了不同的流程视图。在这个框图中, 更容易看出对象之间的关系, 但顺序信息则不够明显。

为此，可以对一个情景同时生成Sequence框图和Collaboration框图。尽管它们作用相同，包含相同信息，但视图有所不同。在Rose中，可以按F5或选择Browse►Create (Sequence/Collaboration) Diagram在Collaboration框图与Sequence框图之间进行转换。

生成Collaboration框图

和Sequence框图一样，Collaboration框图通常在浏览器中生成，直接在使用案例或包中生成。图4.7演示了如何对模型生成新Collaboration框图。前面曾介绍过，另一种生成Collaboration框图的方法是生成Sequence框图，然后按F5，Rose自动从Sequence框图生成Collaboration框图。

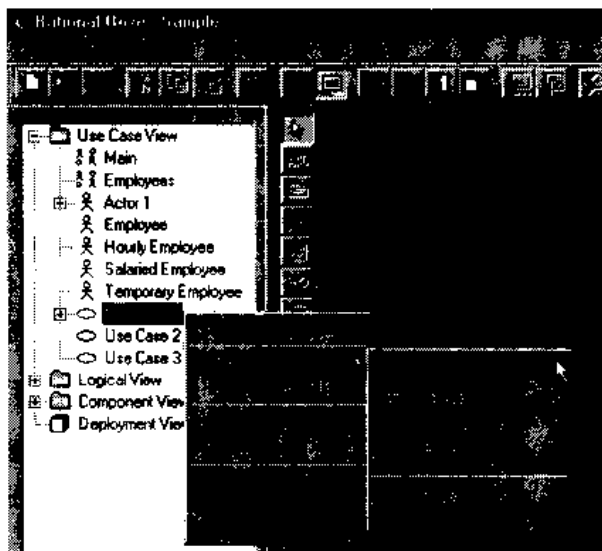


图4.7 生成新Collaboration框图

要生成Collaboration框图：

1. 右单击浏览器中的相应使用案例。
2. 从弹出菜单选择New►Collaboration diagram。
3. 命名新的Collaboration框图。
4. 双击打开浏览器中的Collaboration框图。

删除Collaboration框图

建立模型时你可能发现有些Collaboration框图不再准确或不再使用。要清理模型，就要在Rose中用浏览器删除多余的Collaboration框图。

要删除Collaboration框图：

1. 右单击浏览器中的Collaboration框图。
2. 从弹出菜单选择Delete。

将文件和URL连接Collaboration框图

和Sequence框图一样，可以将文件和URL连接Collaboration框图。连接的文件或URL可能包含框图演示的事件流信息，框图实现的特定要求或其他与框图有关的文档。如果连接的

文件或URL与整个使用案例有关，则应连接使用案例本身。

要将文件连接Collaboration框图：

1. 右单击浏览器中的Collaboration框图。
2. 从弹出菜单中选择New►File。
3. 用Open对话框选择要连接的文件。
4. 选择Open连接文件。

要将URL连接Collaboration框图：

1. 右单击浏览器中的Collaboration框图。
2. 从弹出菜单中选择New►URL。
3. 输入要连接的URL名。

要打开连接的文件：

双击浏览器中的文件名。Rose打开相应应用程序并装入文件。

或

1. 右单击浏览器中的文件。
2. 从弹出菜单选择Open。Rose打开相应应用程序并装入文件。

要打开连接的URL名：

双击浏览器中的URL名。Rose打开相应应用程序并装入URL。

或

1. 右单击浏览器中的URL。
2. 从弹出菜单选择Open。Rose打开相应应用程序并装入URL。

要删除连接的文件或URL：

1. 右单击浏览器中的文件或URL。
2. 从弹出菜单中选择Delete。

Collaboration框图工具栏

Collaboration框图工具栏与Sequence框图工具栏非常相似，但有几个Sequence框图没有的选项，如对象链接和数据流。下面几节介绍如何用这些工具栏按钮将项目加进框图中。表4.2列出了Collaboration框图工具栏中的按钮。

表4.2 Collaboration框图工具栏中的图标

图标	按钮	用途
	Selects or deselects an item	将光标返回箭头以选择项目
	Text Box	将文本框加进框图
	Note	将图注加进框图
	Anchor Note to Item	将图注连接框图中的项目
	Object	将新对象加进框图
	Class Instance	向框图中增加新的类实例

(续表)

图标	按钮	用途
	Object Link	生成对象之间的通信路径
	Link to Self	显示对象可以调用自己的属性
	Link Message	在两个对象间或一个对象本身增加消息
	Reverse Link Message	在两个对象间或一个对象本身从相反方向增加消息
	Data Flow	显示两个对象之间的信息流
	Reverse Data Flow	在相反方向显示两个对象之间的信息流

使用Interaction框图中的角色

每个Sequence和Collaboration框图都应有角色对象。角色对象是外部刺激，让系统运行某个功能。Interaction框图中的角色对象包括与Use Case框图上的使用案例交互的角色。

要生成Interaction框图中的角色对象：

1. 打开Interaction框图。
2. 在浏览器中选择角色。
3. 将角色从浏览器中拖动到打开框图中。

要删除Interaction框图中的角色对象：

1. 选择Interaction框图中的角色。
2. 选择Edit►Delete from Model或按Ctrl+D。

说明：从框图中删除对象，但不从模型中删除相应类。

使用对象

Sequence和Collaboration框图显示参与特定使用案例的某个流程的对象。一旦角色对象加进框图后，下一步要加进其他对象。前面曾介绍过，可以通过检查事件流和情境文档中的名词而找到参与特定Sequence和Collaboration框图的对象。完成这个步骤后，就要在对象之间增加消息。

将对象加进Interaction框图

生成Sequence和Collaboration框图的前面几步之一就是增加对象。可以通过检查事件流和情境文档中的名词而找到参与的对象。

要将对象加进Interaction框图：

1. 选择Object工具栏按钮。
2. 单击框图中要放对象的位置。在Sequence框图中，对象在顶上排成行。
3. 输入新对象名。
4. 加进对象后，可以拖放对象进行重排，可以在两个对象之间插入对象，只要第二步单击两个对象之间即可。

要将对象加进Collaboration框图：

1. 选择Object工具栏按钮。
2. 单击框图中要放对象的位置。在Collaboration框图中，对象位置随意。
3. 输入新对象名。

从Interaction框图删除对象

建立Interaction框图时，可能要删除一些对象。从框图中删除对象时，Rose自动删除以该对象开头或结尾的消息。Rose自动重新编号所有其余消息。

从Sequence框图中删除对象时，Rose自动删除对应Collaboration框图中的对象，但不从模型中删除对应的类。同样，从Collaboration框图中删除对象时，Rose自动删除对应Sequence框图中的对象。如果改变主意，可以用Edit菜单的Undo选项。要从Collaboration框图或Sequence框图删除对象：

1. 选择Collaboration或Sequence框图中的对象。
2. 选择Edit►Delete from Model或按Ctrl+D。

说明：从框图中删除对象并不从模型中删除相应类。

对象规范

Rose提供了多个不同字段，可以给框图中的对象增加细节。例如，可以设置对象名、它的类、持续性、对象是否有多个实例。还可以在对象规范窗口中增加对象文档，如图4.8。下面几节介绍对象规范窗口中的每个选项。

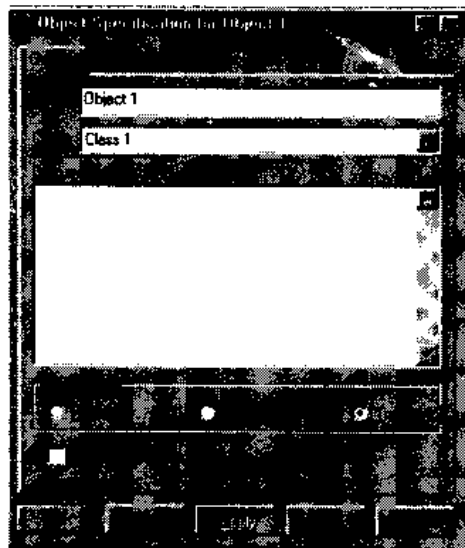


图4.8 对象规范窗口

要打开对象规范窗口：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。

或

1. 选择Collaboration或Sequence框图中的对象。
2. 选择Browse►Specification或按Ctrl+B。

命名对象

每个Collaboration或Sequence框图中的对象都应有唯一名称。尽管类名是一般性的（如Employee和Company），但对象名是很具体的（John Doe和Rational Software Corporation）。在Interaction框图中，两个对象可能是同一个类的实例。例如，在库存系统中，Part类可能有一个实例Engine，与Part的另一实例carburetor通信。可以在对象规范窗口中输入框图中每个对象的名称，也可以在框图中直接输入。

要命名对象：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Name字段中输入对象名。框图中每个对象都应有唯一名称。后面也可以用这个字段改变对象名。

或

1. 选择Collaboration或Sequence框图中的对象。
2. 右单击，使对象中显示光标。
3. 输入对象名。

要将文档加进对象：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Documentation字段中输入对象文档。

或

1. 选择Collaboration或Sequence框图中的对象。
2. 在文档窗口输入对象文档。

说明：如果对象已经映射类，则文档也会加进类中。否则文档只留在对象中。输入对象文档并不影响代码生成，输入类文档则会出现现在代码中。对象文档出现在File►Print Specifications菜单选项生成的报表中。

将对象映射类

每个Collaboration或Sequence框图中的对象都可以映射类。例如，Joe的帐号可以映射Account类。在对象规范窗口中，可以用Class字段设置对象的类。缺省情况下，类设置为(Unspecified)。

选择对象的类时，可以用模型中现有的类，也可以对对象生成新类。在下列过程中，我们介绍这两种方法。

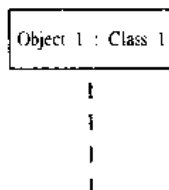
准备代码生成时，所有对象都应映射类。

要将对象映射现有类：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Class下拉列表框中，输入类名或从下拉清单选择类名。
4. 将对象映射类后，类名出现在框图的对象名后面，用冒号分开。

或

1. 选择浏览器Logical视图中的类。
2. 将类从浏览器拖动到框图中的对象上。
3. 将对象映射类后，类名出现在框图的对象名后面，用冒号分开。



说明：如果将对象映射类之后删除这个类，则类名出现在框图中，但放在括号内。

要删除对象的类映射：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Class下拉列表框中选择（Unspecified）。

要对对象生成新类：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Class下拉列表框中选择<New>，Rose进入新类规范窗口。

要保证所有对象映射类：

1. 选择Report►Show Unresolved Objects。
2. Rose列出还没有映射类的所有对象。

要在框图中只显示对象名：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Name字段中输入对象名。
4. 在Class下拉列表框中选择（Unspecified）。

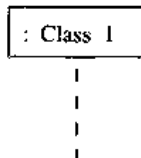
要在框图中同时显示对象名和类名：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Name字段中输入对象名。
4. 在Class下拉列表框中选择类名或输入类名。

要在框图中只显示类名：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。

3. 删除Name字段中的对象名。Rose只显示对象的类名，前面加上冒号。



设置对象持续性

Rose中可以设置对象持续性。Rose提供三个选项：

持续 (Persistent)：持续对象保存到数据库或其他形式的持续存储体中。也就是说，即使程序终止之后，对象依然存在。

静态 (Static)：静态对象保存在内存中，直到程序终止。它在Sequence框图执行之外仍然生成，但不保存到持续存储体中。

临时 (Transient)：临时对象只在短时间内保存在内存中（例如直到Open Specification框图执行逻辑结束）。

要设置对象持续性：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 在Persistence字段中选择相应单选钮：Persistent、Static和Transient。

说明：如果将对象的类持续性设置为Persistent，则可以设置对象持续性为Persistent、Static和Transient。如果将对象的类持续性设置为Transient，则可以设置对象持续性为Static或Transient。

使用对象的多个实例

Rose中可以用一个图标表示同一个类的多个实例。例如，假设要在Sequence或Collaboration框图中表示一系列员工。与其把每个员工显示为不同对象，不如用多个实例图标显示员工表。多个实例的UML图注如下：

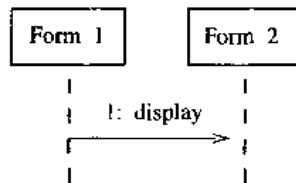


要使用对象的多个实例：

1. 右单击Collaboration或Sequence框图中的对象。
2. 从弹出菜单选择Open Specification。
3. 复选或取消Multiple Instances框图。Rose在Collaboration框图中使用相应图标（单实例或多实例）。Rose在Sequence框图中使用单实例图标。

使用消息

消息是对象间的通信，一个对象（客户）请求另一对象（供应者）做某件事。生成代码时，消息变为函数调用。本例中，一个窗体要求另一个窗体显示：



将消息加进Interaction框图

将对象放进Collaboration或Sequence框图后，下一步要加进对象之间发送的消息。在Sequence框图上，可以在两个对象的生命线之间画一个箭头以加进消息。在Collaboration框图中，则要先加进两个对象间的链接。然后在其间加上消息。

将消息加进Sequence框图

在Sequence框图中，消息在两个对象的生命线之间和一个对象自己的生命线上用箭头画成。消息从框图的上面往下按时间顺序显示。

要将消息加进Sequence框图：

1. 选择工具栏中的Object Message按钮。
2. 将鼠标从发送消息的对象或角色生命线拖动到接收消息的对象或角色生命线，如图4.9。
3. 输入消息文本。

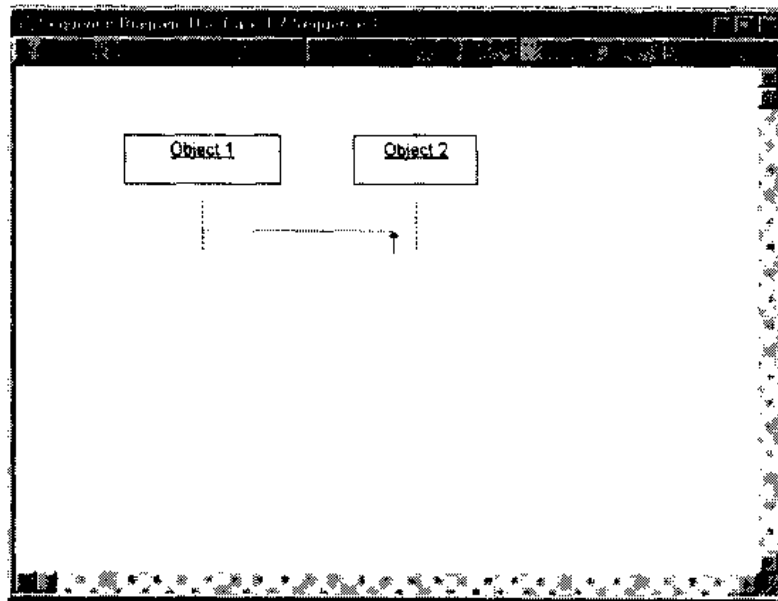


图4.9 将消息加进Sequence框图

要将反身消息加进Sequence框图：

1. 选择工具栏中的Message to Self按钮。
2. 单击收发消息的对象生命线，如图4.10。
3. 在新消息选择时，输入消息文本。

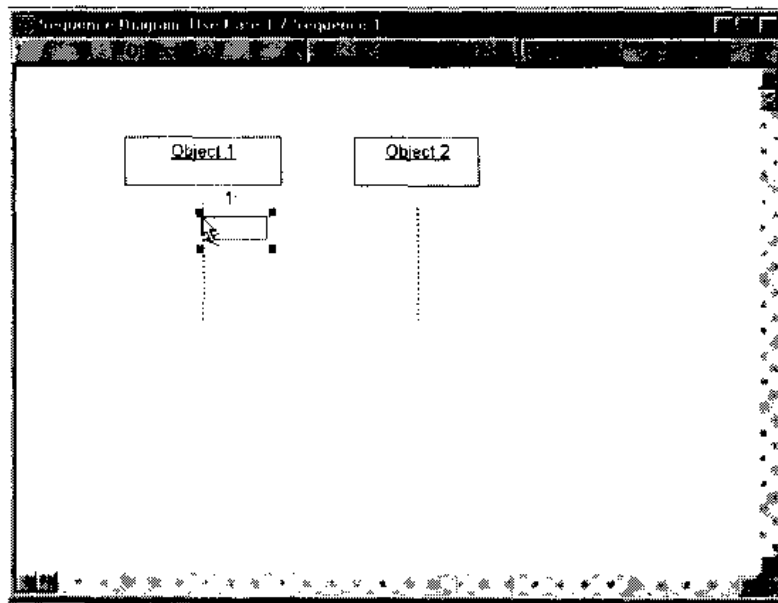


图4.10 将反身消息加进Sequence框图

从Sequence框图中删除消息

使用Sequence框图时，可能要删除一些前面绘制的消息。如果删除消息，则Rose自动将其余消息重新编号。

要从Sequence框图中删除消息：

1. 选择要删除的消息。
2. 选择Edit►Delete from Model或按Ctrl+D。

重排Sequence框图中的消息

有时需要重排Sequence框图中的消息。Rose中很容易重排消息，只要将消息拖放到所要位置即可。消息重排时，会自动重新编号。

要重排Sequence框图中的消息：

1. 选择要移动的消息。
2. 将消息拖放到所要位置。消息重排时，会自动重新编号。

Sequence框图中的消息编号

尽管消息从上往下阅读，但也可以对每个消息编号，显示消息顺序，如图4.11。消息编号在Interaction框图中是可选的。Sequence框图缺省关闭编号。

要打开或关掉消息编号：

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 复选或取消Sequence Numbering框，如图4.12。

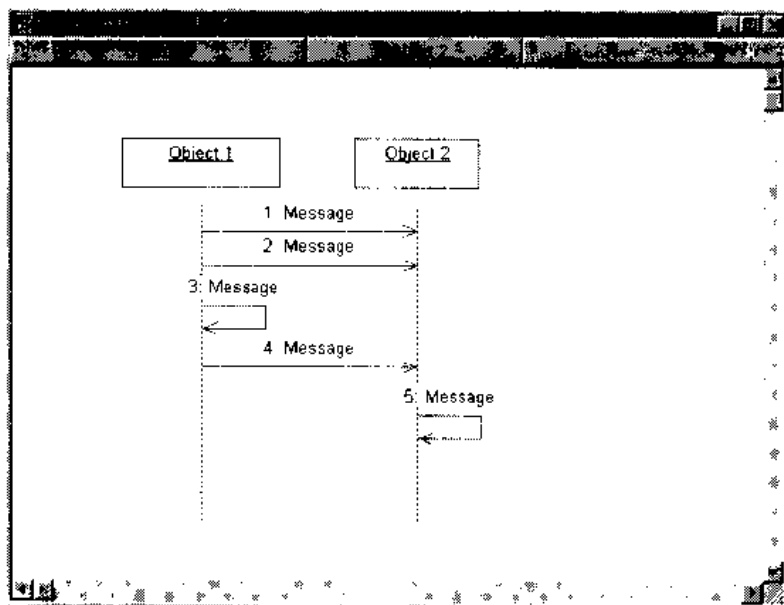


图4.11 Sequence框图中的消息编号

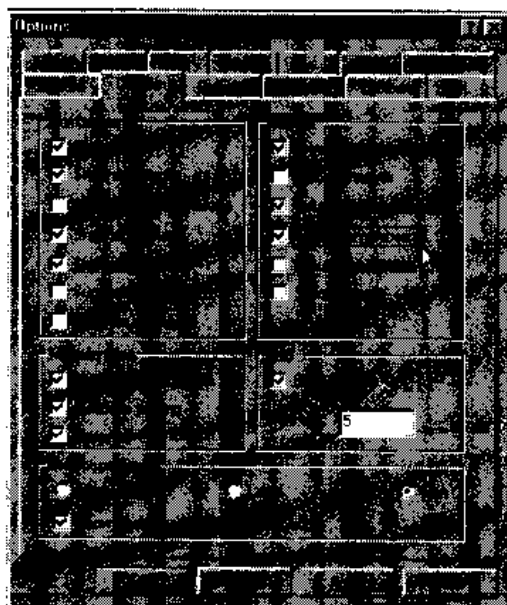


图4.12 消息编号复选框

浏览Sequence框图中的控制焦点

在Sequence框图中，可以浏览控制焦点。控制焦点是个小矩形，如图4.13，可以知道某个时间具有控制权的对象。这是Sequence与Collaboration框图的差别之一，控制焦点只在Sequence框图中显示。

要开/关Sequence框图中的控制焦点：

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 复选或取消Focus of Control框，如图4.14。

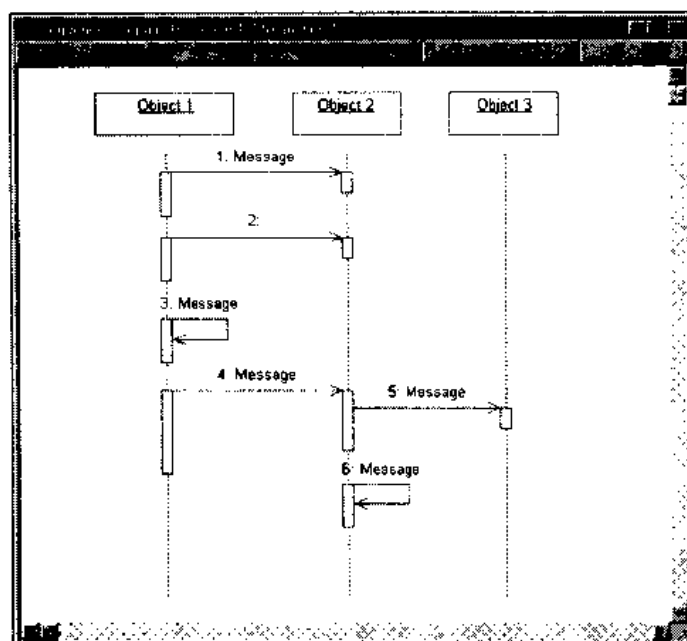


图4.13 Sequence框图中的控制焦点

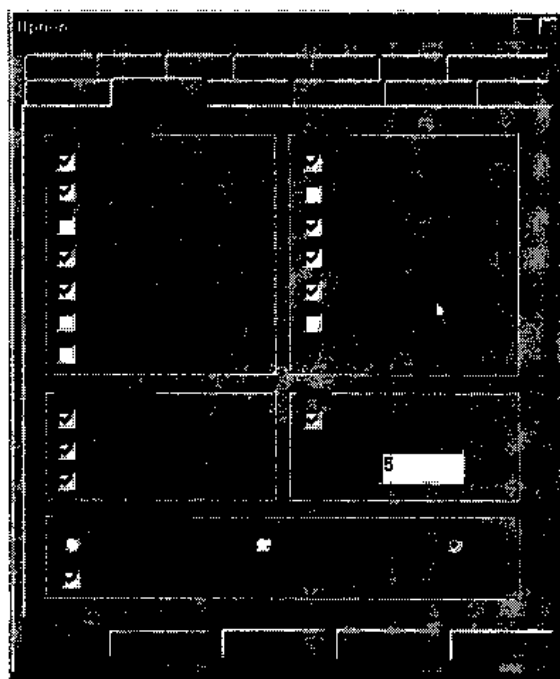


图4.14 Focus of Control复选框

将消息加进Collaboration框图

将消息加进Collaboration框图之前，要先建立对象之间的通信路径，称为链接，用Object Link工具栏按钮生成。增加链接后，可以在对象之间加上消息。

要将消息加进Collaboration框图：

1. 选择Object Link工具栏按钮。
2. 从一个对象拖动到另一对象，生成链接。

3. 选择Link Message或Reverse Link Message工具栏按钮。
4. 单击两个对象间的链接。Rose画出消息箭头，如图4.15。
5. 在新消息选择时，输入消息文本。

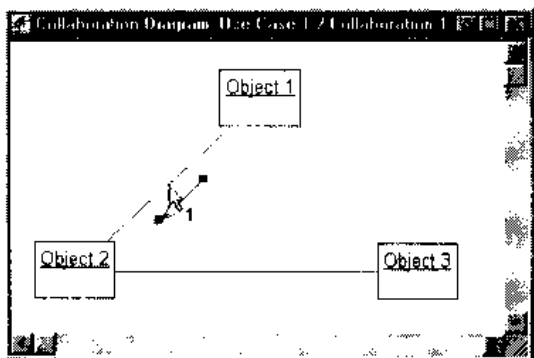


图4.15 将消息加进Collaboration框图

要将反身消息加进Collaboration框图：

1. 选择Link to Self工具栏按钮。
2. 单击收发消息的对象。Rose画一个反身链路，在对象上方，显示为半圆。



3. 选择Link Message工具栏按钮。
4. 单击对象反身链路。Rose增加消息箭头，如图4.16。
5. 在新消息选择时，输入消息文本。

说明：如果对Collaboration框图中的对象增加多个反身消息，则对其余消息跳过第一步和第二步

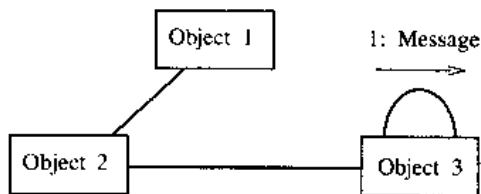


图4.16 向Collaboration框图增加反身消息

从Collaboration框图中删除消息

和Sequence框图中一样，可以从Collaboration框图中删除消息。删除消息时，Rose自动将其余消息重新编号。

要从Collaboration框图中删除消息：

1. 选择要删除的消息。
2. 选择Edit►Delete From Model或按Ctrl+D。

Collaboration框图中的消息编号

Sequence框图中，框图总是从上往下阅读，消息编号并不需要。而Collaboration框图中如果消息不编号，则没有顺序信息。

尽管我们通常不提倡，但Collaboration框图中可以关掉消息编号。

要打开或关掉消息编号：

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 复选或关闭Collaboration和Sequence Numbering框。

将数据流加进Collaboration框图

前面曾介绍过，Collaboration框图与Sequence的一个差别是使用控制焦点，另一个差别是使用数据流。Collaboration框图显示数据流，而Sequence框不显示。

数据流显示一个对象向另一对象发出消息时返回的信息。一般来说，不要对Collaboration框图中的每个消息加上数据流，否则框图中会堆满价值不大的信息。如果消息只是返回“OK，消息收到，一切顺利进行”或“OOPS，运行请求功能是出错”，则不值得在框图中显示。假设是公司的一列员工名单，则很重要，必须在框图中显示。

最终将每个消息映射类操作时，数据流中的消息加进操作细节中。一般来说，目前不必花太多时间考虑数据流。如果数据流很重要，对开发人员有帮助，则将其加进框图中，否则不要加进。

要将数据流加进Collaboration框图：

1. 选择Data Flow或Reverse Data Flow工具栏按钮。
2. 单击返回数据的消息。Rose自动在框图中加上数据流箭头，如图4.17。
3. 选择新数据流后，输入要返回的数据。

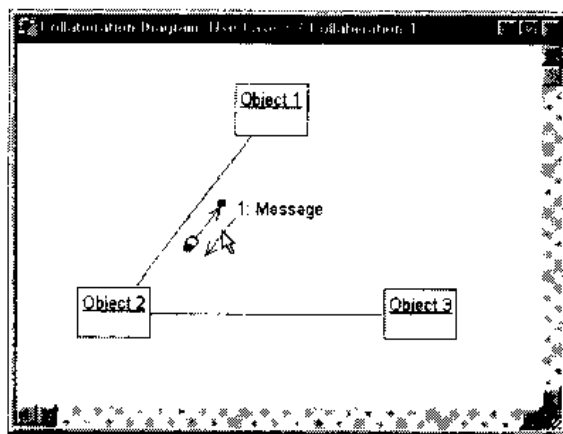


图4.17 将数据流加进Collaboration框图

消息规范

在Rose中，可以设置多个不同选项，在每个消息中增加细节。和使用案例与角色一样，可以在消息中增加名称和文档。也可以设置同步和频率选项。本节要介绍消息中可以设置的

每个选项。

要打开消息规范:

双击框图中的消息, 出现规范窗口, 如图4.18。

或

1. 选择框图中的消息。
2. 选择**Browse>Specification**或按Ctrl+B。

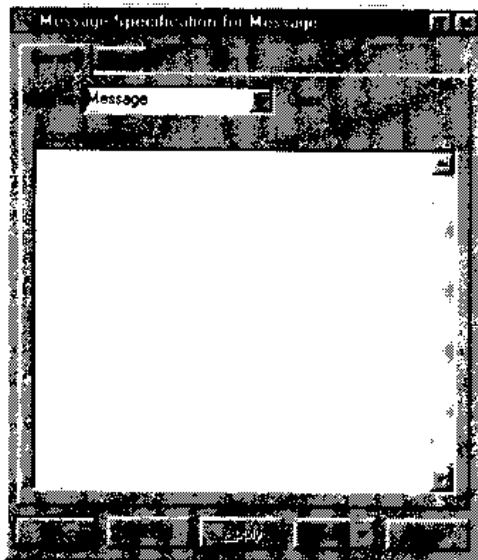


图4.18 消息规范窗口

命名消息

在消息规范窗口中, 可以命名消息或改变名称, 或增加文档。每个消息用一个名称表示其用途。后面映射每个消息到操作时, 消息名换成操作名。

要命名消息:

1. 双击**Sequence**或**Collaboration**框图中的消息。
2. 如果已经将接收对象映射为类, 则这个类的操作出现在Name下拉列表框中。选择表中项目或输入消息名。

或

1. 选择**Sequence**或**Collaboration**框图中的消息。
2. 输入消息名。

说明: 如果已经将接收对象映射为类, 则接收类的名称出现在消息名旁边的Class字段中。这个字段不能修改。要改变接收类, 在对象规范窗口中将对象映射另一个类。

要将文档加进消息中:

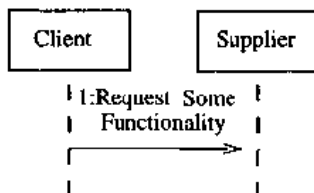
1. 双击消息打开消息规范窗口。
 2. 在**Documentation**区, 输入消息说明。例如, 可以输入一些伪代码, 描述消息的作用。
- 或
1. 选择**Sequence**或**Collaboration**框图中的消息。

2. 在Documentation窗口中输入说明。

说明：如果消息已经映射操作，则文档还会加进操作中。否则文档只保留在消息中。为消息输入的文档不影响代码生成。后面要将每个消息映射类的操作。为操作输入的文档显示为生成代码中的说明语句。

将消息映射操作

生成代码之前，应将Sequence和Collaboration框图中的每个消息映射为类的操作。本例中：消息Request some functionality映射Supplier类的操作。



要将消息映射现有操作：

1. 确保接收对象（供应者）映射一个类。
2. 右单击Sequence和Collaboration框图中的消息。
3. 出现一系列供应者操作。
4. 选择表中的操作，如图4.19。

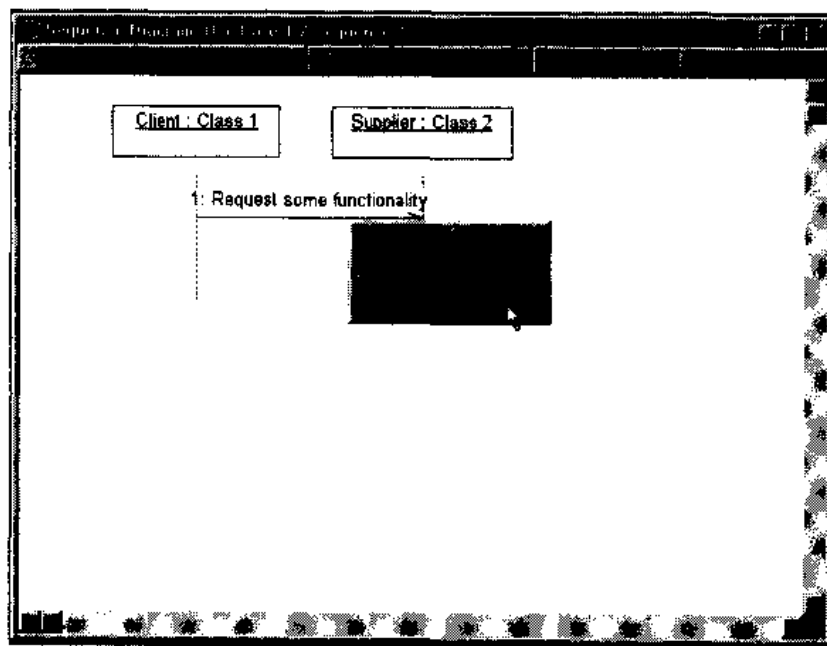


图4.19 将消息映射现有操作

要删除消息的操作映射：

1. 双击Sequence和Collaboration框图中的消息。
2. 在Name字段中删除操作名并输入新的消息名。

要对消息生成新操作：

1. 确保接收对象（供应者）映射一个类。
 2. 右单击Sequence和Collaboration框图中的消息。
 3. 选择<new operation>。
 4. 输入新操作名和细节。操作规范窗口中的选项将在第6章介绍。
 5. 单击OK关闭操作规范窗口，增加新操作。
 6. 右单击消息。
 7. 从出表的表中选择新操作。
- 要保证每个消息映射一个操作：
1. 选择Report►Show Unresolved Messages。
 2. Rose显示所有没有映射操作的消息。

设置消息同步选项

在消息规范窗口的Detail标签中（如图4.20），可以设置消息同步选项。

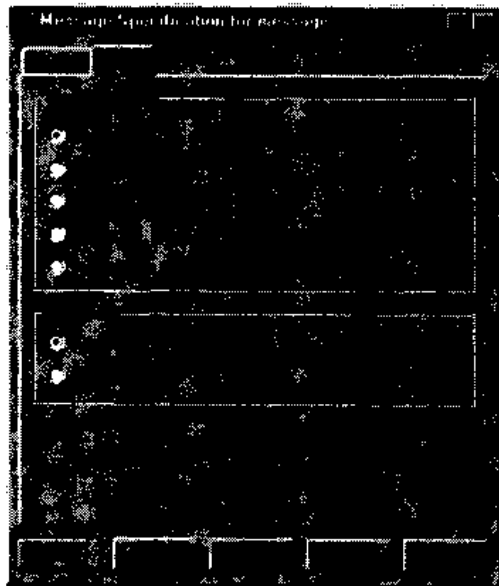
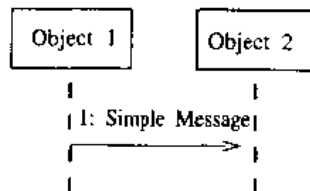


图4.20 设置消息同步选项

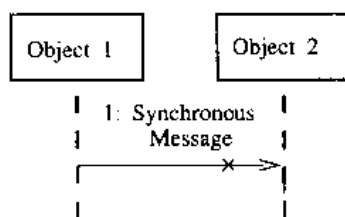
将同步性设置为阻止、超时或异步时，框图上的箭头改变。同步选项有五个：

简单（Simple）：这是消息的缺省值。这个选项指定消息在单个控制线程中运行。

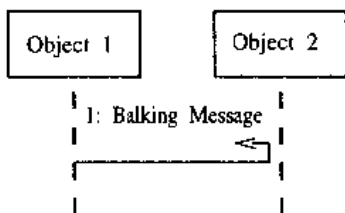
在Sequence或Collaboration框图中，简单消息用如下符号：



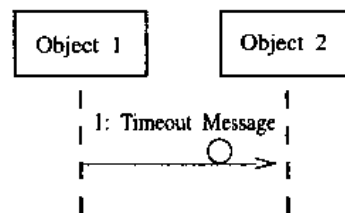
同步（Synchronous）：这个选项用于客户发出消息后等待供应者。响应这个消息时，在Sequence或Collaboration框图中，同步消息用如下符号：



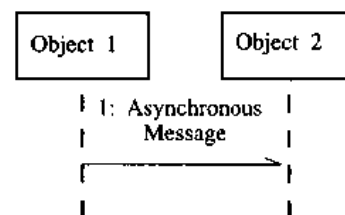
阻止 (Balking)：使用这个选项时，客户发出消息给供应者。如果供应者无法立即接收消息，则客户放弃这个消息。在Sequence或Collaboration框图中阻止消息用如下符号：



超时 (Timeout)：使用这个选项时，客户发出消息给供应者并等待指定时间。如果供应者无法在指定时间内接收消息，则客户放弃这个消息。在Sequence或Collaboration框图中超时消息用如下符号：



异步 (Asynchronous)：使用这个选项时，客户发出消息给供应者然后客户继续处理，不等待消息是否接收。在Sequence或Collaboration框图中异步消息用如下符号：



要设置消息同步性：

1. 双击Sequence或Collaboration框图中的消息。
2. 在消息规范窗口中选择Detail标签。
3. 从窗口单选钮中选择所要同步选项。

设置消息频率

消息频率可以让消息按规定间隔发送。假设消息每30秒发一次。则可以将消息设置为定期消息。频率选项在消息规范窗口Detail标签中，如图4.21。

频率选项有两个：

定期 (Periodic)：这个选项将消息设置为定期消息。

不定期 (Aperiodic)： 这个选项将消息设置为不定期消息，只发送一次，或在不规则时间发送。

说明： 消息频率不改变Sequence或Collaboration框图的样子。

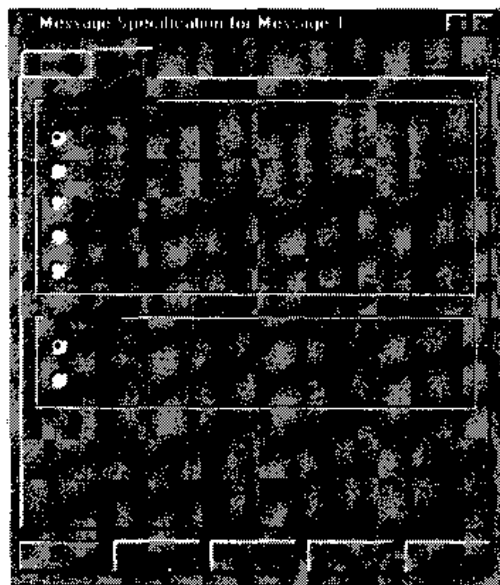


图4.21 设置消息频率

要设置消息频率：

1. 双击Sequence或Collaboration框图中的消息。
2. 在消息规范窗口中选择Detail标签。
3. 从窗口单选钮中选择所要频率选项。

使用图注说明

Interaction框图中的消息名和对象名提供了大量信息。但有时要向框图中增加其他消息。这可以用图注或脚本进行。

图注在框图中的对象上增加某种说明，可以用于澄清对象的用途。

脚本将说明连接消息。在Sequence框图中，可以用脚本增加一些条件逻辑。

将图注加进Interaction框图

可以将图注连接Interaction框图中的对象。图注可以直接在框图中增加说明，而不会影响生成的代码。可以用图注澄清对象的用途或在框图中增加说明。

要将图注加进Interaction框图：

1. 选择Note工具栏按钮。
2. 单击框图内任一位置以增加图注。
3. 在图注选择时，输入文本。

要将图注连接对象：

1. 选择Anchor Note to Item.L具栏按钮。
2. 从图注拖动到对象，将图注连接对象，如图4.22。

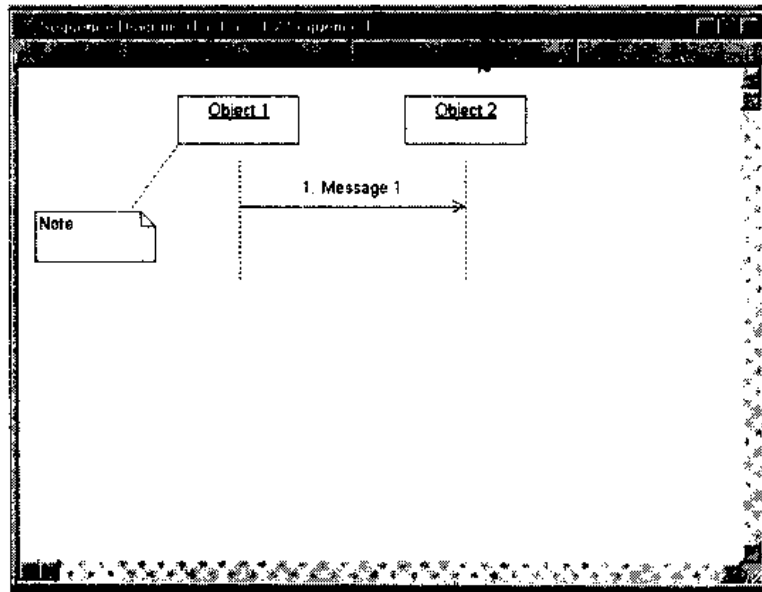


图4.22 将图注连接对象

要将文本框加进Interaction框图：

1. 选择Text Box工具栏按钮。
2. 单击框图内任一位置以增加文本框。
3. 在文本框选择时，输入文本。

使用脚本

在Rose中，图注通常用于给对象增加说明，而脚本则通常给消息增加说明。脚本只用于Sequence框图中，出现在框图左边，与所指消息相对。

可以用脚本标明消息的意义。例如对消息Validate User，可以在脚本中展开解释它的含义：Validate the ID to be sure the user exists, and that the password is correct.

也可以用脚本在框图中输入一些条件逻辑。例如，图4.23是Sequence框图中的脚本。

一般来说，要避免在框图中放入太多条件逻辑，以免失去框图的简单性。在框图中加上IF语句嵌套IF语句。再嵌套IF语句的细节时，框图的可读性就很差了。另一方面，有时需要显示一些条件逻辑。这两个极端需要平衡。只要框图容易阅读和理解，就可以了。如果条件逻辑太复杂，可以生成另一个Sequence框图：一个涉及if部分，一个涉及else部分。

除了IF语句，还可以用脚本显示框图中的循环和其他伪代码。代码不会从脚本产生，但可以通过脚本使开发人员知道逻辑流程。

要将脚本加进Sequence框图：

1. 选择Text Box工具栏按钮。
2. 单击框图中要放脚本的位置。通常是框图的左边。
3. 在文本框图选择时，输入脚本文本。

4. 选择文本框，按住Shift并选择消息。
5. 选择Edit►Attach Script。
6. 然后在框图中上下移动消息时，脚本跟着移动。

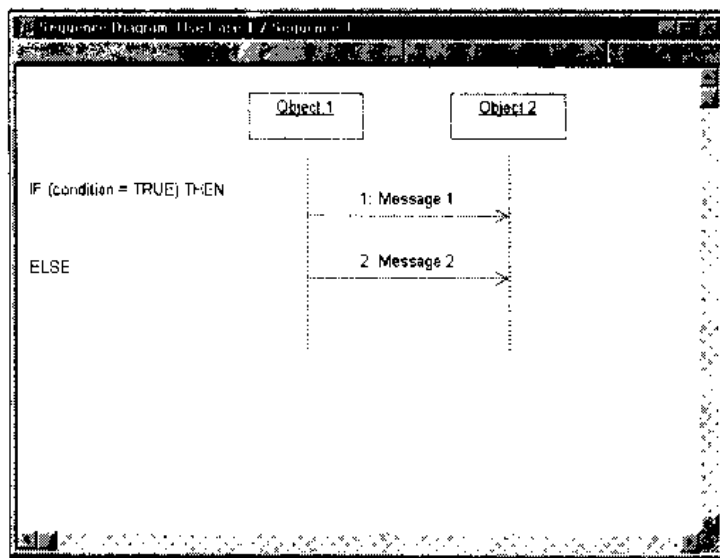


图4.23 Sequence框图中的脚本

要从消息中删除脚本：

1. 选择脚本。
2. 选择Edit►Detach Script。

在Sequence和Collaboration框图间切换

通常要对特定情形生成Sequence或Collaboration框图。如果没有Rose之类的模型工具，则同时生成两种特别费时间，而两者保存同样的信息。

但在Rose中，从Collaboration框图生成Sequence框图或从Sequence框图生成Collaboration框图非常容易。一旦有了情形的Sequence和Collaboration框图后，很容易在其间切换。

要从Sequence框图生成Collaboration框图：

1. 打开Sequence框图。
2. 选择Browse►Create Collaboration diagram或按F5。
3. Rose生成与打开的Sequence框图同名的Collaboration框图。

要从Collaboration框图生成Sequence框图：

1. 打开Collaboration框图。
2. 选择Browse►Create Sequence diagram或按F5。
3. Rose生成与打开的Collaboration框图同名的Sequence框图。

要在Sequence和Collaboration框图间切换：

1. 打开Sequence或Collaboration框图。
2. 选择Browse►Go to (Sequence or Collaboration) Diagram或按F5。
3. Rose寻找与打开的框图同名的Sequence或Collaboration框图。

Interaction框图的两步法

通常，人们用两步法生成Interaction框图。第一步关注客户关心的高级信息。消息还不映射操作，对象还不映射类。这些框图只是让分析人员、客户和让对业务流程感兴趣的其他人了解系统的逻辑流程。

Sequence框图的第一步如图4.24。

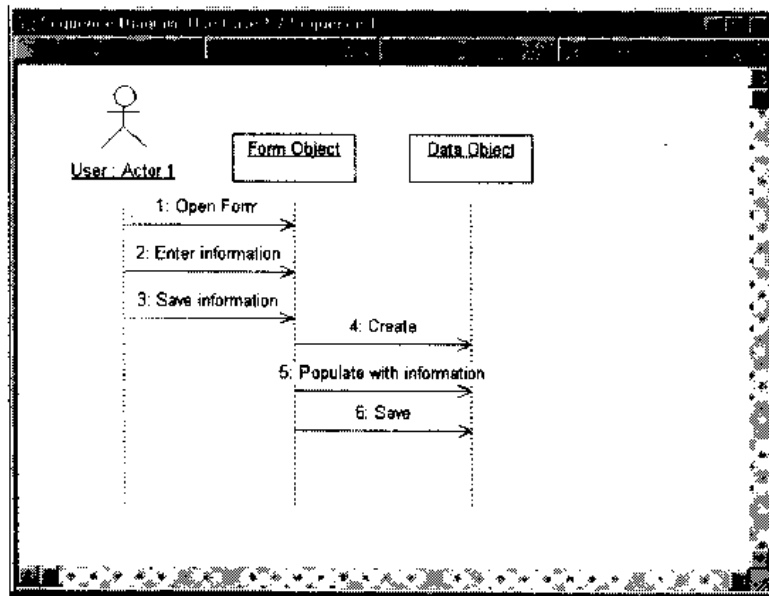


图4.24 Sequence框图的第一步

第二步，客户同意第一步框图的流程后，小组加进更多细节。这时的框图对客户可能作用不大，但对开发人员、测试人员和项目组的其他成员更有用。

首先，可能要向框图中增加一些对象。通常，每个Interaction框图有一个控制对象，负责控制情景中的程序。一个使用案例的所有Interaction框图可能共享同一控制对象，因此一个控制对象可以处理该使用案例的所有程序信息。

如果增加控制对象，则Interaction框图通常如图4.25所示。

注意，控制对象并不进行任何业务处理，只是向其他对象发消息。控制对象负责协调其他对象的效果和委托责任。为此，控制对象有时也称为管理者对象。

使用控制对象的好处是能够将业务逻辑与程序逻辑分开。如果程序需要改变，只影响控制对象。

你可能还要增加其他对象，处理安全、错误处理和数据库连接等问题。许多对象是一般性的，建立一次即可在多个应用程序中复用。例如，下面看看数据库连接问题。

将信息存入数据库和从数据库中取信息时，通常有两个选项。假设要将新员工John Doe保存数据库中。John Doe对象可能知道数据库，这时可以把自己加进数据库中。John Doe对象也可能与数据库逻辑完全分开，这时要通过另一对象将John Doe保存到数据库中。下面先假设John Doe知道数据库，如图4.26。

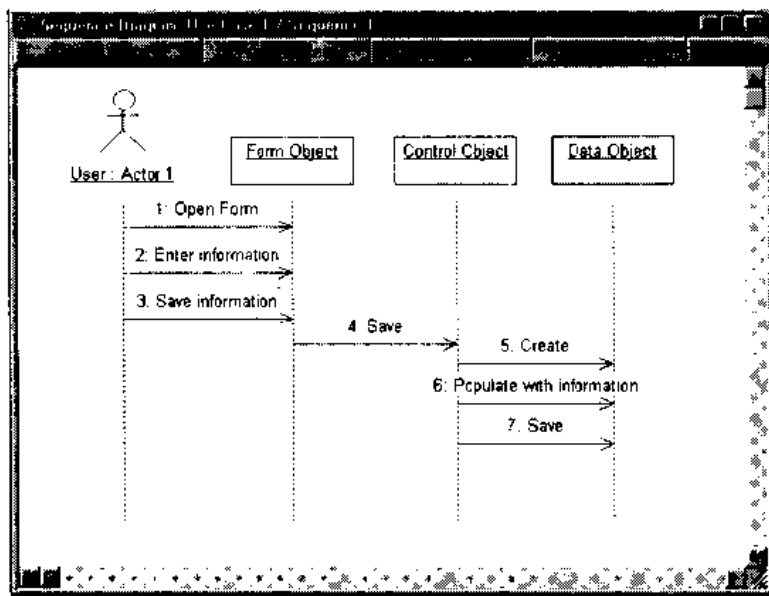


图4.25 带控制对象的Interaction框图

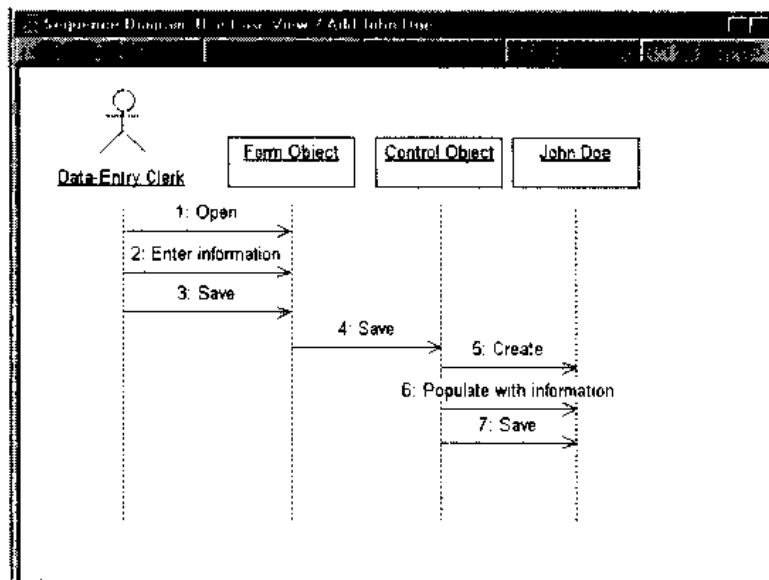


图4.26 应用程序逻辑与数据库逻辑集成

在这种情形中，应用程序逻辑与数据库逻辑是不分开的。John Doe对象负责应用程序逻辑与数据库逻辑，前者包括John Doe的雇用和解聘，后者包括将John Doe存到数据库中和从数据库中取出。如果数据库逻辑改变，则会影响应用程序逻辑，因为许多对象都包含一些数据库逻辑。另一方面，这种方法更容易建模和实现。

另一种选项是应用程序逻辑与数据库逻辑分开。这种情况下，需要生成另一对象来处理数据库逻辑。我们称这个对象为事务管理器（Transaction Manager）。John Doe对象仍然保持业务逻辑，知道如何雇用和解聘John Doe。而事务管理器对象知道如何从数据库中取出John Doe或将其存入数据库中。这时Sequence框图如图4.27。

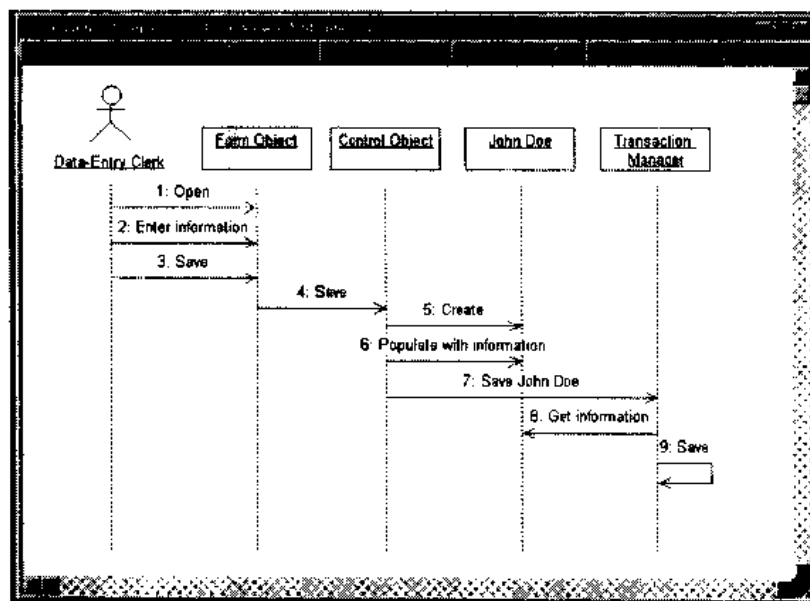


图4.27 数据库逻辑与应用程序逻辑分开

这种方法的好处是更容易在另一个具有不同数据库或根本没有数据库的应用程序中复用John Doe对象，还可以减少需求改变时的影响。数据库改变并不影响应用程序逻辑，应用程序改变也不影响数据库逻辑。缺点是造型和实现需要更费时间。

这是两种最常用的方法，还有其他处理数据库问题的方法。选哪个方法由你决定，但各个Interaction框图中应保持一致的方法。

除了数据库问题外，还可以增加错误处理、安全性、进程间通信等对象。这些细节客户不感兴趣，但对开发人员非常重要。

一旦加进所有对象后，下一步要将每个对象映射到类。可以将对象映射现有类或为对象生成新类（见上面“将对象映射到类”一节）。然后将框图中的每个消息映射操作（见“将消息映射操作”一节）。最后，还可以进入对象和消息的细节，设置对象持续性、消息同步性和消息频率等项目。

练习

本练习要建立Sequence和Collaboration框图，在订单处理系统中加进新订单。

问题

通过与Bob的交流，Susan了解了为Robertson's Cabinets公司建立的订单处理系统中要包括哪些特性。他生成一个Use Case框图。通过这个框图，每个人都可以协商系统范围。

然后Susan要分析系统组件。Enter New Order使用案例对用户具有最高优先级，是具有高风险性的项目。为了有更多时间处理这个使用案例的风险，Susan决定首先处理这个使用案例。

她与销售部负责人Carl商量。通过交谈，他们确定了这个使用案例的事件流程。利用这

个信息, Susan确定了一些情形, 建档如下:

- 销售人员增加一个新订单
- 销售人员想增加新订单, 但项目无库存
- 销售人员想增加新订单, 但存入数据库时出错

然后她对销售人员增加一个新订单的情形建立Sequence和Collaboration框图。

生成Interaction框图

生成Interaction框图, 在订单处理系统中增加新订单。完成的Sequence框图如图4.28

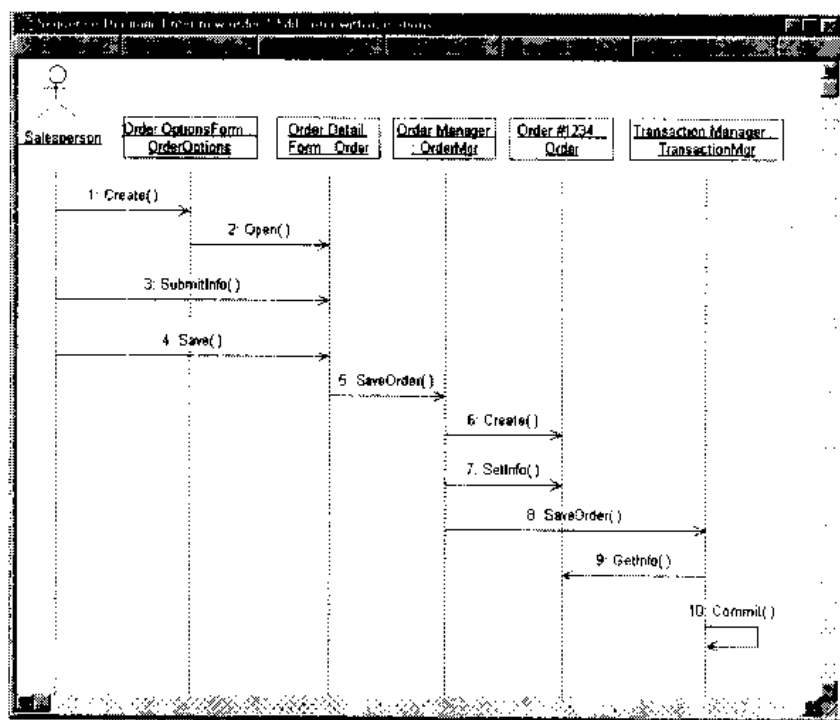


图4.28 增加新订单的Sequence框图

这只是完成Enter New Order使用案例要造型的框图之一。这个框图显示一切顺利时的情形。还要对出错时和用户选择不同选项时增加其他框图。使用案例中每个流程都有构造一个Interaction框图。

练习步骤

设置

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 确保复选Sequence Numbering、Collaboration Numbering和Focus of Control。
4. 单击OK退出Options窗口。

生成Sequence框图

1. 右单击浏览器中的Logical视图。
2. 选择New►Sequence Diagram。
3. 将新框图取名为Add Order。
4. 双击打开新框图。

将角色和对象加进框图

1. 将Salesperson角色从浏览器拖动到框图中。
2. 选择Object工具栏按钮。
3. 单击框图顶部加上对象。
4. 将新对象取名为Order Options Form。
5. 对下列对象重复第3、4步：
 - Order Detail Form
 - Order #1234

将消息加进框图

1. 选择Object Message工具栏按钮。
2. 从Salesperson角色的生命线拖动到Order Options Form对象的生命线。
3. 在消息选择时，输入Create new order。
4. 重复第2、3步，将其他消息加进框图中：
 - Open form (Order Options Form与Order Detail Form之间)
 - Enter order number, customer, order items (Salesperson与Order Detail Form之间)
 - Save the order (Salesperson与Order Detail Form之间)
 - Create new, blank order (Order Detail Form和Order #1234之间)
 - Set the order number, customer, order items (Order Detail Form与Order #1234之间)
 - Save the order (Order Detail Form与Order #1234之间)

这时如图4.29，完成了增加新订单的第一步Sequence框图。然后要考虑控制对象和数据库连接等问题。看看框图，Order Detail Form对象有大量程序责任，最好让控制对象处理。我们还让新订单自己存到数据库中，最好让另一对象负这个责任。

将其他对象加进框图

1. 选择Object工具栏按钮。
2. 单击Order Detail Form object和the Order #1234对象之间，插入新对象。
3. 将新对象取名为Order Manager。
4. 选择Object工具栏按钮。
5. 将新对象加到Order #1234右边。
6. 将新对象取名为Transaction Manager。

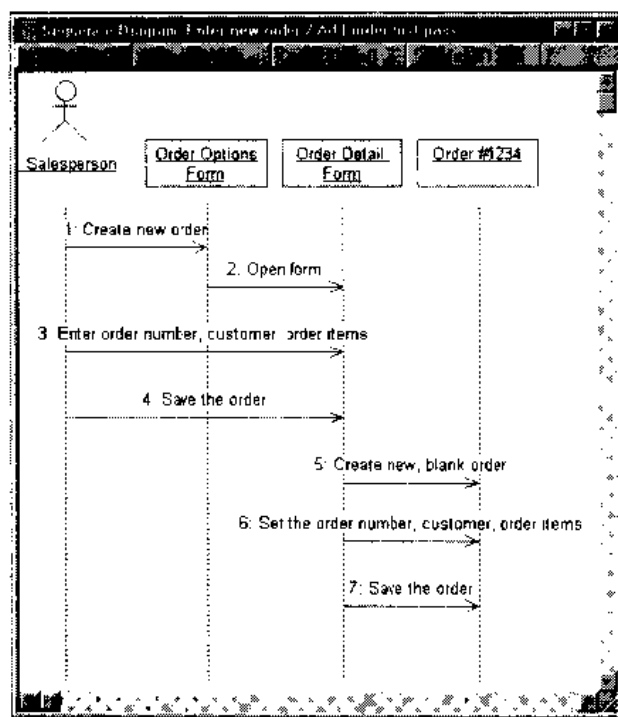


图4.29 增加新订单的第一步Sequence框图

对新对象指定责任

1. 选择消息五: Create new, blank order
 2. 按Ctrl+D删除消息。
 3. 重复第1、2步删除最后两个消息:
 - Set the order number, customer, order items
 - Save the order
 4. 选择Object Message工具栏按钮。
 5. 在消息四下面增加新消息, 放在Order Detail Form¹与Order Manager对象之间。
 6. 将新消息命令名为Save the order。
 7. 重复4到6步加进消息六到消息九如下:
 - Create new, blank order (在Order Manager¹与Order #1234之间)
 - Set the order number, customer, order items (在Order Manager和Order #1234之间)
 - Save the order (在Order Manager¹与Transaction Manager)
 - Collect order information (在Transaction Manager¹与Order #1234之间)
 8. 选择Message to Self工具栏按钮。
 9. 在消息九下面, 单击Transaction Manager对象生命线, 加进反身消息。
 10. 将新消息命名为Save the order information to the database。
- 这时Sequence框图如图4.30。

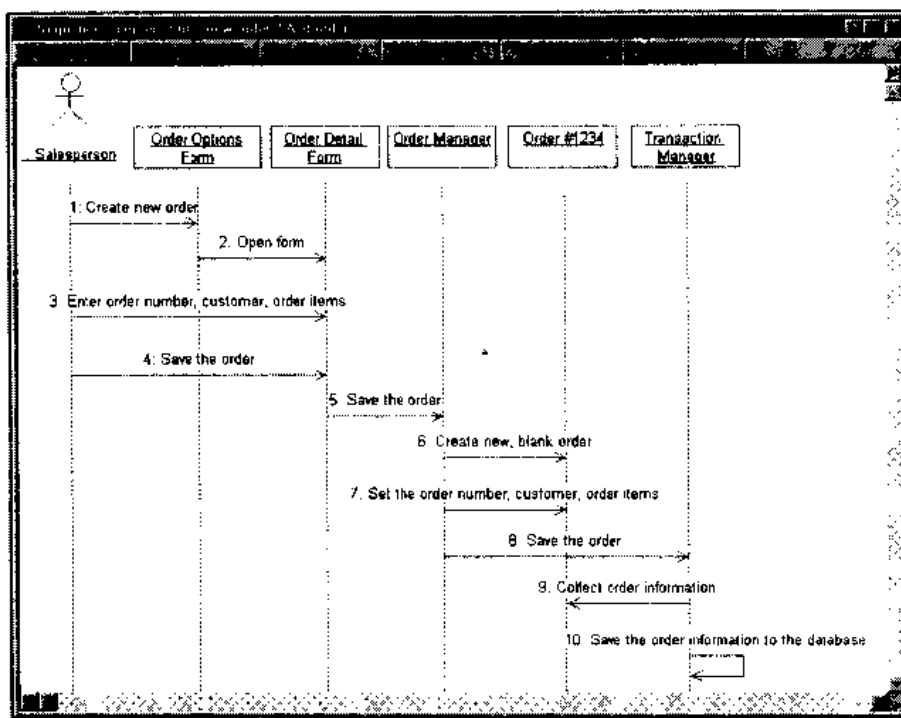


图4.30 加进其他对象后的Sequence框图

将对象映射类

1. 右单击Order Options对象。
 2. 从弹出菜单中选择Open Specification。
 3. 在Class下拉列表框图中，选择<New>，打开类规范窗口。
 4. 在Name字段中输入OrderOptions。
 5. 单击OK关闭类规范窗口，返回对象规范窗口。
 6. 在Class下拉列表框图中，选择Order Options。
 7. 单击OK返回框图。这时对象显示为Order Options Form: OrderOptions。
 8. 重复1到7步将其余对象映射为类：
 - 对Order Detail Form对象生成类OrderDetail
 - 对Order Manager对象生成类OrderMgr
 - 对Order #1234对象生成类Order
 - 对Transaction Manager对象生成类TransactionMgr
- 完成这些步骤后，Sequence框图如图4.31。

将消息映射操作

1. 右单击消息一，Create new order。
2. 从弹出菜单选择<new operation>。出现操作规范窗口。
3. 在Name字段中输入新操作名：Create。
4. 选择OK关闭操作规范窗口，返回框图。
5. 右单击消息一。

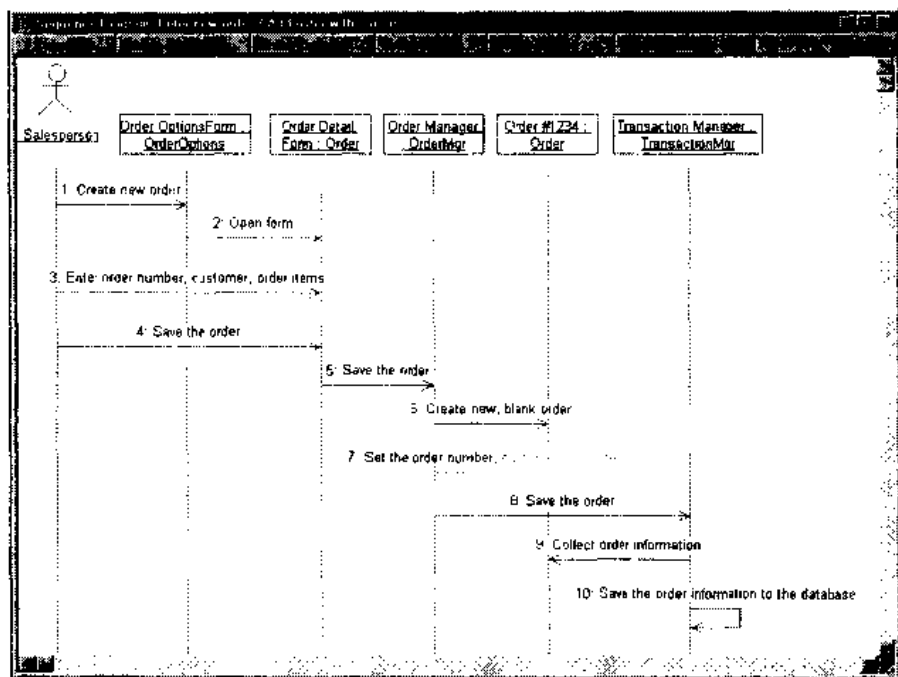


图4.31 带类名的Sequence框图

6. 从弹出菜单选择新操作Create()。
7. 重复第1步到第6步，将每个消息映射操作：
 - 映射2: Open form到Open操作
 - 映射3: Enter order number, customer, order items到SubmitInfo操作
 - 映射4: Save the order到Save操作
 - 映射5: Save the order到SaveOrder操作
 - 映射6: Create new, blank order到Create操作
 - 映射7: Set the order number, customer, order items到SetInfo操作
 - 映射8: Save the order到SaveOrder操作
 - 映射9: Collect order information到GetInfo操作
 - 映射10: Save the order information到the database到Commit操作

这时框图如图4.32。

生成Collaboration框图

要从Sequence框图生成Collaboration框图，按F5，或按下列步骤从头开始生成Collaboration框图。

生成Collaboration框图

1. 右单击浏览器中的Logical视图。
2. 选择New►Collaboration diagram。
3. 将新框图取名Add order。
4. 双击打开新框图。

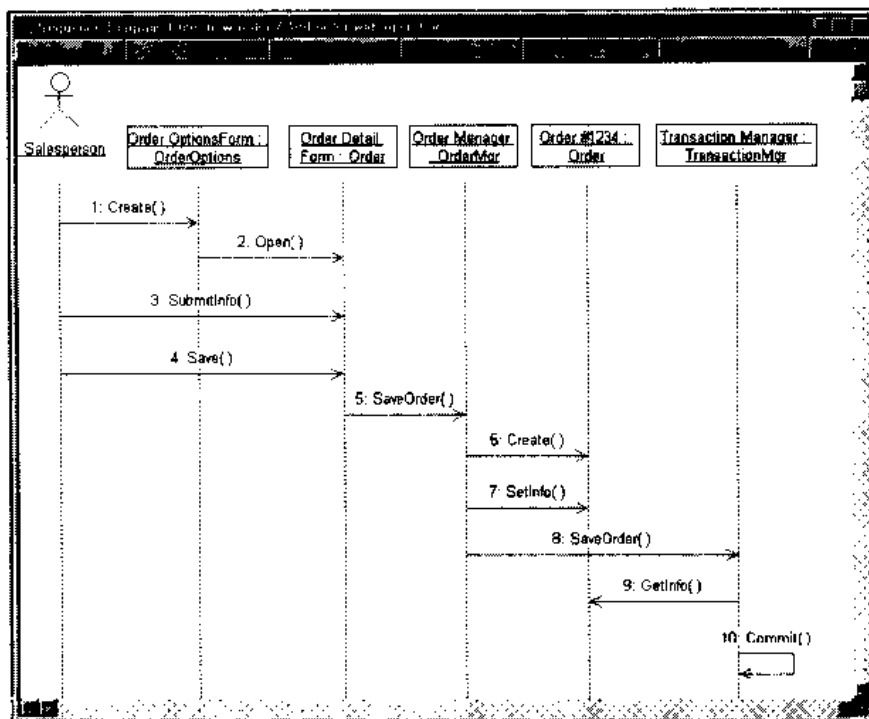


图4.32 带操作的Sequence框图

将角色和对象加进框图

1. 将Salesperson角色从浏览器拖动到框图中。
2. 选择Object工具栏按钮。
3. 单击框图内任一位置加进对象。
4. 将新对象取名Order Options Form。
5. 重复第2到第4步，将其他对象加进框图如下：
 - Order Detail Form
 - Order #1234

将消息加进框图

1. 选择Object Link工具栏按钮。
2. Salesperson角色拖动到Order Options Form对象。
3. 重复第1、2步加进下列对象间的链接：
 - Salesperson actor and Order Detail Form
 - Order Options Form and Order Detail Form
 - Order Detail Form and Order #1234 object
4. 选择Link Message工具栏按钮。
5. 单击Salesperson与Order Options Form之间的链接。
6. 在消息选择时，输入Create new order。
7. 重复第4、6步，将其他消息加进框图中：

- Open form (Order Options Form与Order Detail Form之间)
- Enter order number, customer, order items (Salesperson与Order Detail Form之间)
- Save the order (Salesperson与Order Detail Form之间)
- Create new, blank order (Order Detail Form和Order #1234之间)
- Set the order number, customer, order items (Order Detail Form与Order #1234之间)
- Save the order (Order Detail Form与Order #1234之间)

图4.33的框图是增加新订单的第一步Collaboration框图。和前面一样，后面要增加一些细节并注意对象的责任。

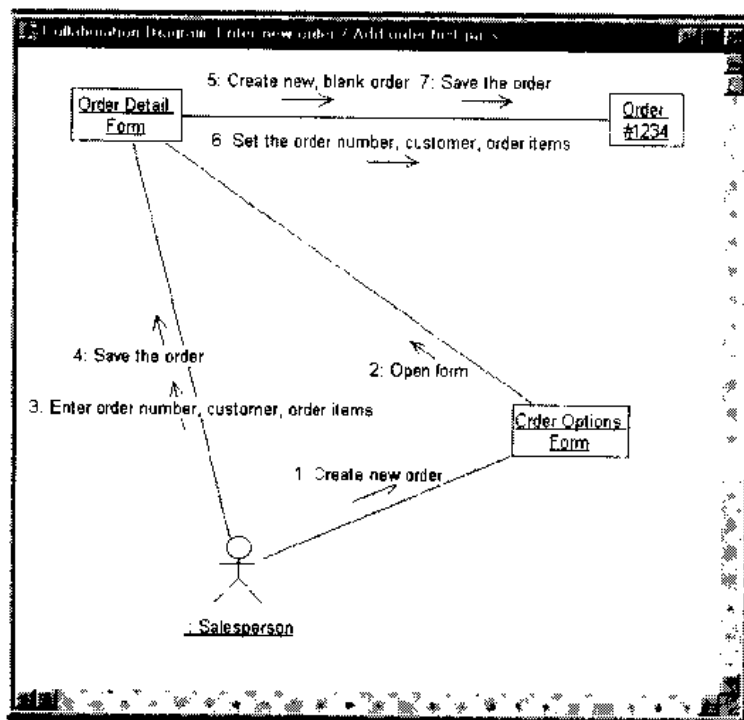


图4.33 增加新订单的第一步Collaboration框图

将其他对象加进框图

1. 选择Object工具栏按钮。
2. 单击框图内任一位置加进对象。
3. 将新对象取名Order Manager。
4. 选择Object工具栏按钮。
5. 将另一新对象加进框图。
6. 将新对象取名Transaction Manager。

向新对象指定责任

1. 选择消息五: Create new, blank order。选择文字而不是箭头。
2. 按Ctrl+D删除消息。
3. 重复第1、2步删除消息六和消息七:

- Set the order number, customer, order items
 - Save the order
4. 选择Order Detail Form和Order #1234之间的链接。
 5. 按Ctrl+D删除链接。
 6. 选择Object Link工具栏按钮。
 7. 在Order Detail Form与Order Manager对象间拖一条链接。
 8. 选择Object Link工具栏按钮。
 9. 在Order Manager与Order #1234对象间拖一条链接。
 10. 选择Object Link工具栏按钮。
 11. 在Order #1234与Transaction Manager对象间拖一条链接。
 12. 选择Object Link工具栏按钮。
 13. 在Order Manager object与Transaction Manager对象间拖一条链接。
 14. 选择Link Message工具栏按钮。
 15. 单击Order Detail Form与Order Manager对象间的链接以加进新消息。
 16. 将新消息取名为Save the order。
 17. 重复第14到第16步，加进消息六到消息九：
 - Create new, blank order (在Order Manager与Order #1234之间)
 - Set the order number, customer, order items (在Order Manager和Order #1234之间)
 - Save the order (在Order Manager与Transaction Manager)
 - Collect order information (在Transaction Manager与Order #1234之间)
 18. 选择Link to Self工具栏按钮。
 19. 单击Transaction Manager对象增加反身链接。
 20. 选择Link Message工具栏按钮。
 21. 单击Transaction Manager的反身链接以增加消息。
 22. 将机关报消息取名为Save the order information to the database。
- 这时Collaboration框图如图4.34。

将对象映射类（如果上面Sequence框图练习中已经生成类）

1. 在浏览器中找到Order Options类。
 2. 将它拖动到框图中Order Options Form对象上方。
 3. 重复第1、2步将每个对象映射相应类：
 - 类OrderDetail拖到Order Detail Form对象上方
 - 类OrderMgr拖到Order Manager对象上方
 - 类Order拖到Order #1234对象上方
 - 类TransactionMgr拖到Transaction Manager对象上方
- 将对象映射相应类（如果上面Sequence框图练习没有生成类）
1. 右单击Order Options对象。
 2. 从弹出菜单选择Open Specification。

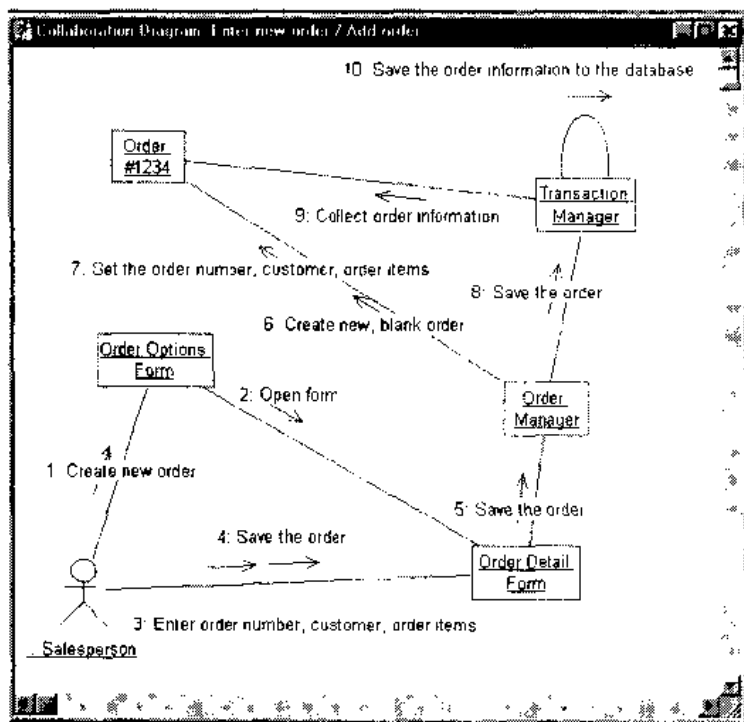


图4.34 加进其他对象后的Collaboration框图

3. 在Class下拉列表框中选择<New>。出现类规范窗口。
4. 在Name字段中输入Order Options。
5. 单击OK关闭类规范窗口，返回对象规范窗口。
6. 在Class字段中选择下拉清单的Order Options。
7. 单击OK返回框图，这时对象显示为Order Options Form : OrderOptions。
8. 重复步骤1到步骤7将其余对象映射为类：
 - 对Order Detail Form对象生成类OrderDetail
 - 对Order Manager对象生成类OrderMgr
 - 对Order #1234对象生成类Order
 - 对Transaction Manager对象生成类TransactionMgr

这时Collaboration框图如图4.35。

将消息映射操作（如果上面Sequence框图练习中已经生成操作）

1. 右单击消息 1: Create new order。
2. 从弹出菜单选择Open Specification。
3. 在Name下拉列表框中选择操作名Create。
4. 按OK。
5. 重复第1到第4步，将每个消息映射操作：
 - 映射2: Open form到Open操作
 - 映射3: Enter order number, customer, order items到SubmitInfo操作
 - 映射4: Save the order到Save操作

- 映射5: Save the order到SaveOrder操作
- 映射6: Create new, blank order到Create操作
- 映射7: Set the order number, customer, order items到SetInfo操作
- 映射8: Save the order到SaveOrder操作
- 映射9: Collect order information到GetInfo操作
- 映射10: Save the order information到the database到Commit操作

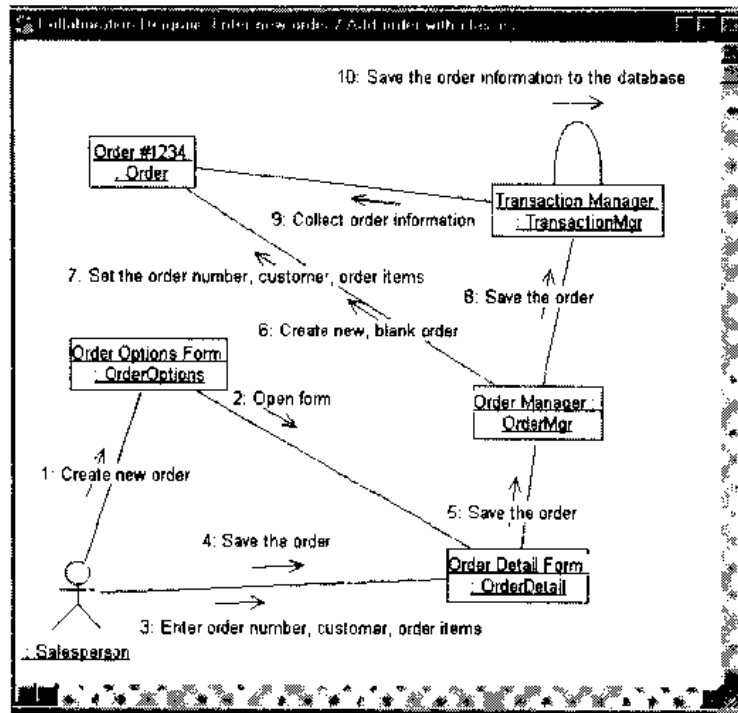


图4.35 带类名的Collaboration框图

将消息映射操作（如果上面Sequence框图练习中没有生成操作）

1. 右单击消息一，Create new order。
2. 从弹出菜单选择<new operation>，出现操作规范窗口。
3. 在Name字段中输入新操作名：Create。
4. 选择OK关闭操作规范窗口，返回框图。
5. 右单击消息一。
6. 从弹出菜单选择Open Specification。
7. 在Name下拉列表框中选择操作名Create。
8. 单击OK。
9. 重复第1到第8步，将每个消息映射操作：
 - 映射2: Open form到Open操作
 - 映射3: Enter order number, customer, order items到SubmitInfo操作
 - 映射4: Save the order到Save操作
 - 映射5: Save the order到SaveOrder操作
 - 映射6: Create new, blank order到Create操作
 - 映射7: Set the order number, customer, order items到SetInfo操作

- 映射8: Save the order到SaveOrder操作
- 映射9: Collect order information到GetInfo操作
- 映射10: Save the order information到the database到Commit操作

最后的Collaboration框图如图4.36。

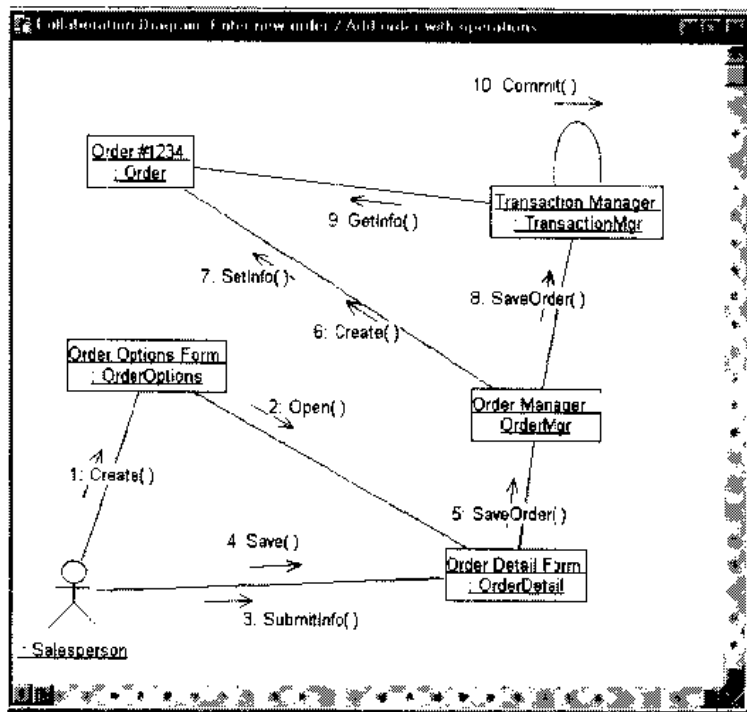


图4.36 带操作的Collaboration框图

小结

本章介绍了UML中最灵活的概念之一——Interaction框图。对象Interaction框图显示对象如何配合，实现使用案例的功能。Interaction框图有两种：Interaction和Collaboration框图。两者显示相同信息，但从不同方面显示。

Interaction框图显示按时间的信息流。Interaction框图对使用案例的每种路径生成，用于浏览使用案例进行中的功能。Collaboration框图显示信息流，但不按时间显示。Collaboration框图显示对象间的关系和对象间的消息。从Collaboration框图中，系统设计人员可以看出哪个对象是瓶颈，或发现哪些对象需要直接相互通信。Collaboration框图还可以显示对象间的数据流，而Interaction框图则没有这个功能。Rose中Interaction和Collaboration框图可以互换。一个框图作出改变时，对应框图也改变。

通常，每个Interaction框图经历两步方法。第一步，大多数技术细节不放进框图中。这些框图可以让用户验证过程捕获是否正确。一旦验证第一步框图后，就可以建立第二步框图。第二步框图的图标使用者不是用户，而是项目小组，包括设计人员、开发人员和分析人员。第二步将许多细节放进Interaction框图中。框图中每个对象映射类。框图中每个消息映射类的操作。可以产生模型质量报表，显示未映射的对象和消息。

完成第二步Interaction框图后，Rose已经生成系统所需的一些类。下一章介绍如何生成类框图，让开发人员实际开发类。

第5章 类 与 包

- 生成Class框图
- 将类加进模型
- 使用类、图注与包

上一章介绍了对象如何交互提供系统功能。现在要介绍类本身及其如何组成包。Rose中对象模型对应于Logical视图中的类。类是生成对象的蓝图，因此帐号是生成Joe存款帐号对象的类。本章要介绍如何在Logical视图中生成类和Class框图。

Rose模型的Logical视图

本章介绍Rose模型Logical视图中存放的一些项目。上一章曾介绍过，可以在Logical视图中生成Sequence和Collaboration框图。可以加进Logical视图的其他项目包括：

- Classes（类）
- Class框图
- Use Case框图
- Attributes and operations（属性与操作）
- Associations（关联）
- State Transition框图

我们首先介绍类和Class框图，下面几章将介绍增加Class框图的细节，包括增加属性与操作，类与包的关系。

Class框图

Class框图显示系统中的类与类包，提供系统组件及其相互关系的静态图形。在Rose中，Class框图旁边有下列符号：



一个系统通常要生成几个Class框图。有些显示类及其关系的子集，有些显示类的子集，包括属性和操作，还有一些显示类包及包之间的关系。可以对系统生成多个Class框图。

缺省情况下，有一个主Class框图，直接放在Logical视图下面。这个Class框图显示模型中的类包。每个包中有另一主框图，包含这个包中的所有类。在Rose中，双击Class框图中的包自动打开其主Class框图。

Class框图是项目小组的良好设计工具，有助于开发人员在编码之前显示和计划系统结构，保证系统一开始就设计合理。图5.1是Class框图。

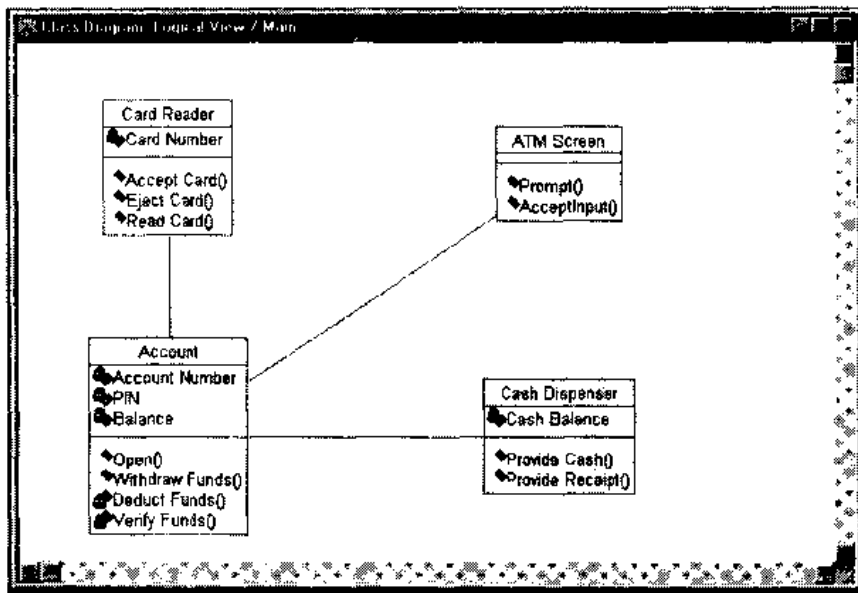


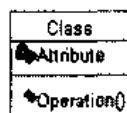
图5.1 Class框图

何谓类

类是包装信息和行为的项目。习惯上，我们把系统的信息放在数据库一方，行为放在应用程序一方。面向对象方法的特点之一就是將一小组信息和影响信息的行为连接在一起。我们将一小组信息和影响信息的行为连接在一起，包装成类。

例如，在人事系统中，我们有个**Employee**类。这个类包含一些信息，如员工号、姓名、地址和电话号码。**Employee**类还有一些行为。**Employee**类知道如何雇用和解聘员工，如何提升员工。

在UML中，类用下列图注表示：



类的上层表示类名，（可选）包括版型。中层包括属性（即类包含的信息）。下层包括操作（类的行为）。为了简化框图，可以隐藏类的属性和操作。

也可以显示每个属性与操作的可见性、每个属性的数据类型和每个操作的签名。下一章将介绍这些选项。**Employee**类是员工对象的模板。对象是类的实例。例如，**Employee**类的对象可能是John Doe、Fred Smith和公司的其他员工。

Employee类描述员工对象的信息和行为。以上例为例，John Doe对象可以保存下列信息：John Doe的姓名、地址和电话号码、工资。John Doe对象知道如何雇用和解聘John Doe、提升John Doe的工资。对象具有类中指定的信息和行为。

寻找类

要寻找类，可以从使用案例的事件流开始。看看事件流中的名词即可知道某些类。名词包括四种项目：

- 角色
- 类
- 类属性
- 除角色、类、属性以外的表达式

排除其他名词之后，就可以找出系统中的类了。

也可以检查Sequence和Collaboration框图中的对象。通过对象的共性即可寻找类。例如，Sequence框图显示工资支付过程。在这个框图中，可能演示John Doe和Fred Smith如何发工资。检查John Doe和Fred Smith对象，他们有一些相似属性：都有姓名、地址和电话号码，有一些相似操作，都知道如何雇用和解聘员工。因此，可以生成一个Employee类，作为John Doe和Fred Smith对象的模板。

Sequence和Collaboration框图中的每个对象都要映射相应的类。上一章详细介绍了如何在Interaction框图中将对象映射类。

从Interaction框图寻找类非常方便。但有些类是无法在所在这些地方找到的。寻找类时要考虑三种不同的版型：Entity、Boundary和控制（项目、边界与控制）。它们并不是都能从事件流和Interaction框图找到。下面介绍版型时会介绍Entity、Boundary和控制类。

生成Class框图

在Rose中，Class框图在Logical视图中生成。可以生成多个Class框图以完整描述你的系统。

生成新模型时，Rose自动在Logical视图中生成一个Main Class框图。通常，这个框图显示模型中的类包。其他Class框图可以直接在Logical视图中生成，也可以在包中生成。

要访问Class框图：

1. 单击浏览器Logical视图旁边的“+”号将其打开。
2. 出现Main Class框图。注意Rose的Class框图右边有下列图标：



3. 双击打开Main Class框图。

说明：首次启动Rose和装入模型时，主Class框图自动打开。

要生成新Class框图：

1. 右单击浏览器中的 Logical视图。
2. 选择弹出菜单中的New►Class diagram。
3. 输入新框图名。
4. 双击打开浏览器中的框图。

要打开现有Class框图：

1. 找到浏览器Logical视图中的Class框图。
2. 双击打开框图。

或

1. 选择Browse►Class Diagram，出现图5.2所示窗口。
2. 在Package列表框中，选择要打开的框图所在包。
3. 在Class Diagrams列表框中，选择要打开的框图。
4. 按OK。

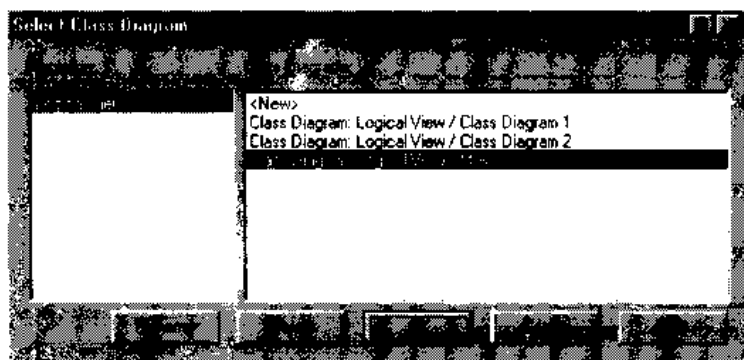


图5.2 打开现有Class框图

要将项目加进Class框图：

用Class框图工具栏按钮将项目加进Class框图。或选择Tool►Create并选择要生成的项目。下面几节介绍如何将各种项目加进Class框图。

从框图中删除项目有两种方法。要从当前框图删除项目：

1. 选择框图中的项目。
2. 按Delete键。

要从模型中删除项目：

1. 选择框图中的项目。
2. 选择Edit►Delete from Model或按Ctrl+D。

或

1. 右单击浏览器中的项目。
2. 选择弹出菜单中的Delete。

删除Class框图

在模型中增加和删除类时，可能要删除一些生成的Class框图。Rose中可以用浏览器删除Class框图。删除Class框图时，框图中的类并不删除，它们仍然在浏览器和其他框图中。

要删除Class框图：

1. 右单击浏览器中的Class框图。
2. 选择弹出菜单中的Delete。

在Class框图中组织项目

框图中加进越来越多类和关系后，会变得很乱很难看。Rose提供了在框图中自动排列所有类的选项。

将属性和操作加进类中时或调整框图中的类尺寸时，类可能变得太大或太小。Rose还可以自动调整所有类的尺寸以适合其中的文本。利用这两个选项，可以将图5.3所示框图变成图5.4的样子。

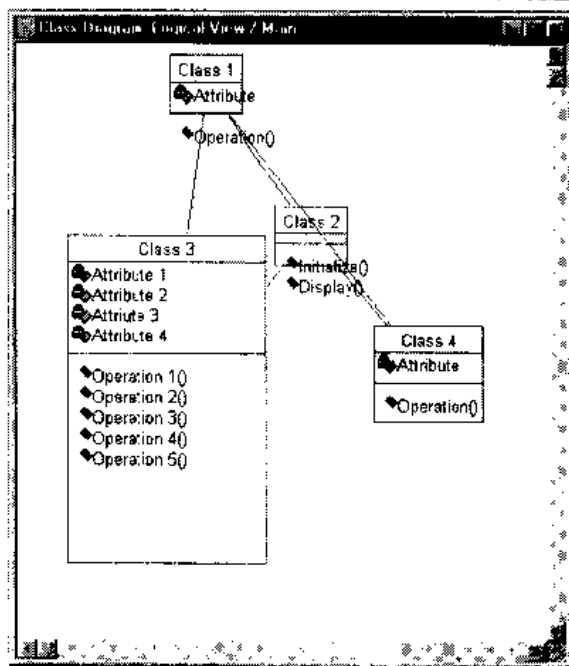


图5.3 没有缩放和自动布置的Class框图

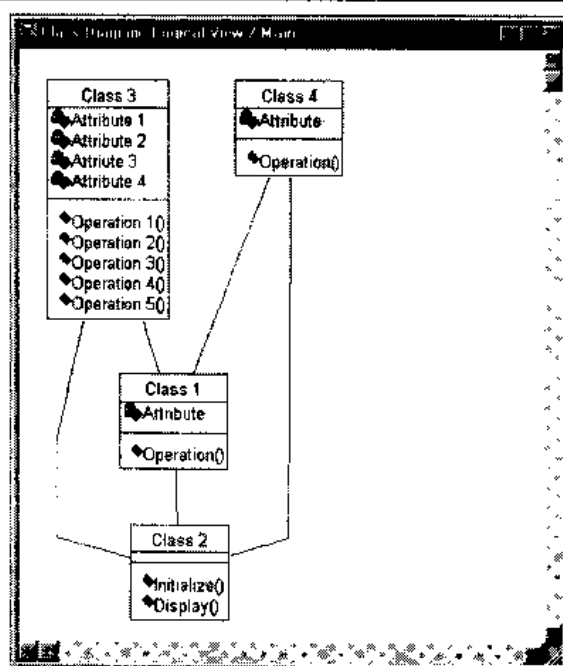


图5.4 缩放和自动布置的Class框图

要布置Class框图中的项目：

选择Tools►Layout Diagram，Rose自动在框图中对齐类。

要缩放Class框图中的项目：

选择Tools►Autosize All，Rose可以自动调整所有类的尺寸以适合其中的类名、属性和操作。

将文件与URL连接Class框图

如果Class框图中有类的补充信息，可以将文件与URL连接Class框图。

用这种方法增加的文件和URL适用于框图中的所有类。如果文件或URL只用于某个类，则应连接这个类。本章稍后会介绍如何将文件与URL连接类。

要将文件连接Class框图：

1. 右单击浏览器中的Class框图。
2. 选择New►File。
3. 用Open对话框寻找要连接的文件。
4. 选择Open连接文件。

要将URL连接类框图：

1. 右单击浏览器中的Class框图。
2. 选择New►URL。
3. 输入要连接的URL名。

要打开连接的文件：

双击浏览器中的文件名。Rose打开相应应用程序并装入文件。

或

1. 右单击浏览器中的文件。
2. 从弹出菜单选择Open。Rose打开相应应用程序并装入文件。

要打开连接的URL:

双击浏览器中的URL名。Rose打开相应应用程序并装入URL。

或

1. 右单击浏览器中的URL。
2. 从弹出菜单选择Open。Rose打开相应应用程序并装入URL。

要删除连接的文件或URL:

1. 右单击浏览器中的文件或URL。
2. 从弹出菜单中选择Delete。

Class框图工具栏

本章要介绍如何将类加进模型和框图。下面几节介绍每个工具栏按钮提供的选项（除涉及关系的选项）。第7章将介绍关系工具栏按钮。

如果工具栏中看不到所有这些按钮，可以右单击工具栏并选择Customize。从这个对话框中，可以增加表5.1所列的每个按钮。

表5.1 Class框图工具栏按钮

图标	按钮	用途
	Selects or deselects an item	将光标返回箭头以选择项目
	Text Box	将文本框加进框图
	Note	将图注加进框图
	Anchor Note to Item	将图注连接框图项目
	Class	将新类加进框图
	Interface	将新接口类加进框图
	Association	画出关联关系
	Aggregation	画出累计关系
	Link Attribute	将关联类与关联关系链接
	Package	将新包中进框图
	Dependency or instantiates	画出相关性关系
	Generalization	画出一般化关系
	Realize	画出实现关系
	Unidirectional Association	画出单向关联
	Parameterized Class	将新的参数化类加进框图
	Class Utility	将新类实用程序加进框图

(续表)

图标	按钮	用途
	Parameterized Class Utility	将新的参数化类实用程序加进框图
	Instantiated Class	将新的实例化类加进框图
	Instantiated Class Utility	将新的实例化类实用程序加进框图

使用类

生成Class框图后，下一步要将类加进模型中。可以加进几种类：普通类、参数化类、实例化类、类实用程序、参数化类实用程序、实例化类实用程序、元类。下面几节将介绍这些类。

我们还将介绍Rose提供的增加类细节的选项。可以命名类、指定其版型、设置其可见性和其他选项。下面将介绍每个选项。

本章还将介绍如何浏览类的属性、操作和关系。下面几章，介绍增加和维护属性、操作与关系的细节。

增加类

首先增加一个标准类，可以用工具栏、浏览器或菜单增加。

新类可以加进浏览器中，这里它可用于其他框图，但不是一开始就放进框图中。也可以将新类加进框图中。如果将新类加进框图中，则它自动加进浏览器中。

要将新类加进框图中：

1. 选择Class工具栏按钮。
2. 单击Class框图中任一位置，新类取名为NewClass。
3. Rose显示所有现有类清单。要将现有类加进框图中，双击清单中的现有类，如图5.5要生成新类，将NewClass换成新类名。注意新类自动加进浏览器的Logical视图中。

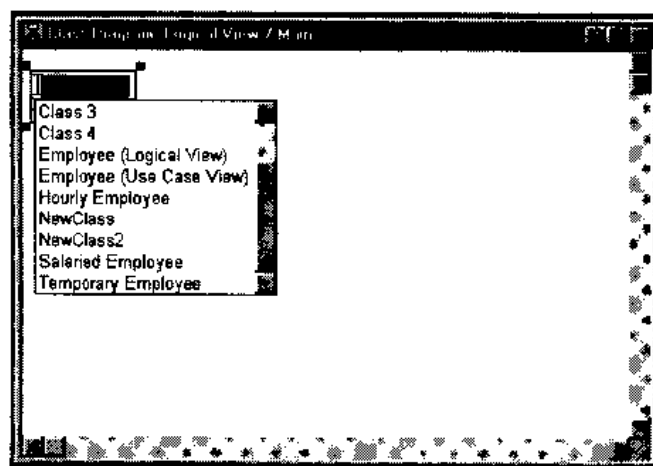


图5.5 增加新类

或

1. 选择Tools►Create►Class。
2. 单击Class框图中任一位置以放上新类，新类名为NewClass。
3. Rose显示所有现有类清单。要将现有类加进框图中，双击清单中的现有类，将NewClass换成新类名。注意新类自动加进浏览器的Logical视图中。

说明：也可以用Tools►Create菜单生成新的参数化、实例化类、类实用程序、参数化类实用程序、实例化类实用程序，详见下面几节。

要用Interaction框图加进新类：

1. 打开Sequence或Collaboration框图。
2. 右单击框图中的对象。
3. 从弹出菜单选择Open Specification。
4. 在Class下拉列表框图中选择<New>。Rose打开新类的规范窗口。
5. 在类规范窗口中Name字段输入类名。

说明：由于Interaction框图在浏览器Use Case视图中，用这个命令生成的新类在Use Case视图中生成要移到Logical视图中，在浏览器中拖放这个类。

要将现有类加进Class框图：

将类从浏览器中拖动到打开的Class框图中。

或

1. 选择Query►Add Classes，出现Add Classes对话框，如图5.6。
2. 在Package下拉列表框中选择要加进框图的类所在包。
3. 将要加的类从Classes列表框移到Selected Classes列表框。要增加所有类，按All按钮。
4. 按OK按钮。
5. Rose将所选类加进打开的框图中。

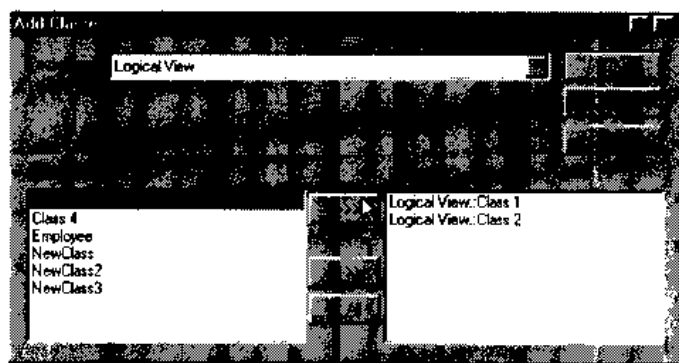


图5.6 将现有类加进Class框图

要将类加进浏览器中：

1. 右单击浏览器中的Logical视图。要将类加进包中，右单击类名。
2. 从弹出菜单选择New►Class。要加进类实用程序或接口，选择New►Class Utility或New►Interface。新类NewClass出现在浏览器窗口中。

3. 选择新类并输入新类名。
4. 然后将新类加进Class框图时，从浏览器窗口拖动到打开框图中。

删除类

和其他模型元素一样，删除类的方法有两种。可以从框图中删除，在其他框图中保留，也可以从整个模型中删除。两种方法都在下面介绍。

要从Class框图中删除类：

1. 选择框图中的类。
2. 按Delete。
3. 注意这个类从框图中删除，在浏览器和其他框图中保留。

要从整个模型中删除类：

1. 选择框图中的类。
2. 选择Edit►Delete from Model或按Ctrl+D。

或

1. 右单击浏览器中的类。
2. 从弹出菜单选择Delete。Rose从浏览器和所有Class框图中删除这个类。

类规范

类的大多数选项在类规范窗口中设置，如图5.7。例如，这个窗口可以设置类的版型、可用性和持续性。下面几节介绍窗口每个标签的选项。

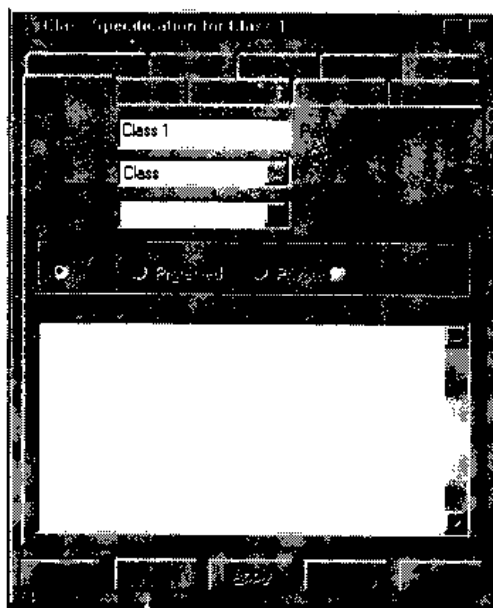
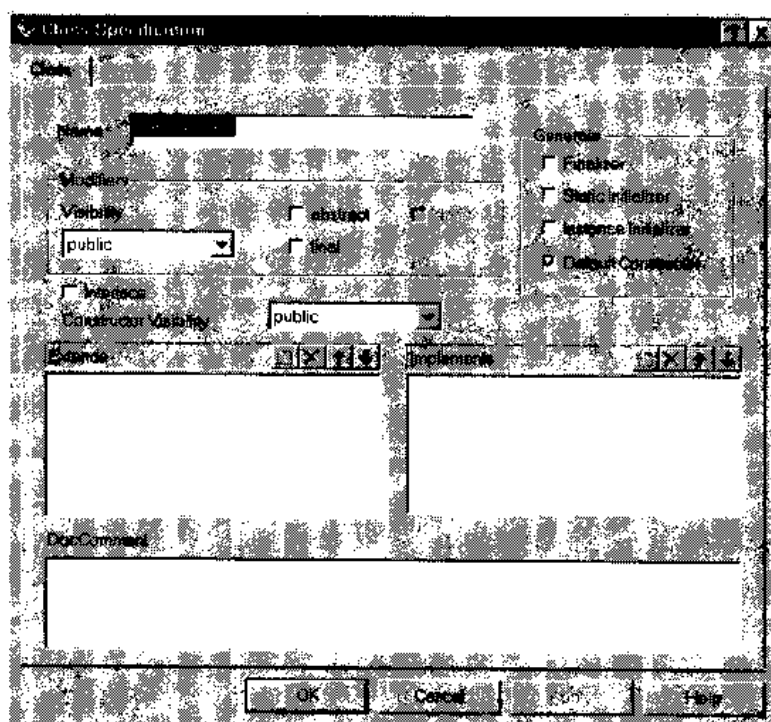


图5.7 类规范窗口

如果使用Rose 98i，检查Java或CORBA类的规范，则类规范窗口稍有不同，如下图。这个窗口的所有选项也可以通过标准规范提供。



要打开类规范窗口：

1. 右单击Class框图中的类。
2. 从弹出菜单选择Open Specification。

说明：如果使用Rose 98i，而类为Java或CORBA类，则从弹出菜单选择Open Standard Specification
选择Open Specification会打开不同的规范窗口。

或

1. 右单击浏览器中的类。
2. 从弹出菜单选择Open Specification。

说明：如果使用Rose 98i，而类为Java或CORBA类，则从弹出菜单选择Open Standard Specification
选择Open Specification会打开不同的规范窗口。

或

1. 选择Class框图中的类。
2. 选择Browse►Specification。

说明：如果使用Rose 98i，而类为Java或CORBA类，则从弹出菜单选择Open Standard Specification
选择Open Specification会打开不同的规范窗口。

或

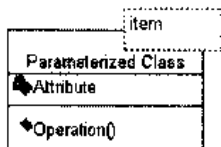
1. 选择Class框图中的类。
2. 按Ctrl+B。

说明：如果使用Rose 98i，而类为Java或CORBA类，则这种方法打开不同的规范窗口。

增加参数化

参数化类是我们介绍的第一种特殊类。参数化类用于生成一系列其他类。通常，参数化类是某种容器，也称为模板。例如，参数化类List，这个参数化类的实例可以生成EmployeeList、OrderList或AccountList等类。

在UML中，参数化类用下列图注显示：



要增加参数化类：

1. 选择Parameterized Class工具栏按钮。
2. 单击框图中任一位置加进新类。
3. 输入类名。

或

1. 用上述方法将一个类加进Class框图或浏览器中。
2. 打开类规范窗口。
3. 在Type字段中输入Parameterized Class。
4. 按OK。

或

1. 选择Tools>Create>Parameterized Class。
2. 单击框图中任一位置加进新类。
3. 输入类名。

设置参数化类变元

类的变元在虚线框中显示。变元是参数化类所包含项目的占位符。上例中，我们将参数“item”变为具体内容，如Employee，从而实例化EmployeeList类。

变元可以是另一个类、一个数据类型或一个常量表达式。可以加进多个变元。

要增加变元：

1. 打开类规范窗口。
2. 选择Detail标签。
3. 右单击Formal Arguments区内任一空白处。
4. 从弹出菜单选择Insert。
5. 输入变元名。
6. 单击Type列头下面打开变元类型下拉清单，如图5.8。选择清单中的一个类型或输入自己的类型。
7. 单击Default Value列头下面输入变元缺省值。缺省值不是必需的。

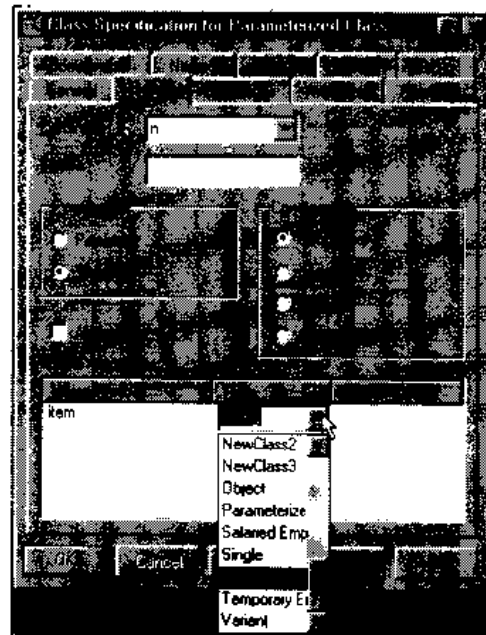


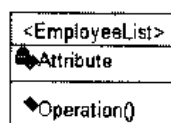
图5.8 在参数化类中增加变元

要删除变元：

1. 打开类规范窗口。
2. 选择Detail标签。
3. 右单击要删除的变元。
4. 从弹出菜单选择Delete。

增加实例化类

实例化类是具有实际变元值的参数化类。上例中我们知道有一系列项目。下面可以提供项目变元值，看看员工表。实例化类的UML图注是把变元名放在<>中的类：



要增加实例化类：

1. 选择Instantiated Class工具栏按钮。
 2. 单击框图中任一位置加进新类。
 3. 输入类名。
- 或
1. 用上述方法将一个类加进Class框图或浏览器中。
 2. 打开类规范窗口。
 3. 在Type字段中输入Instantiated Class。
 4. 按OK。

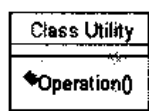
或

1. 选择Tools►Create►Instantiated Class。
2. 单击框图中任一位置加进新类。
3. 输入类名。

增加类实用程序

类实用程序是一组操作。例如，系统中要使用一些数学函数（如Squareroot(), cuberoot()等），不适合放在某个类中。这些函数可以放在一起，包装成类实用程序，让系统中的其他类使用。类实用程序常用于扩展编程语言提供的功能或放一组一般性可复用功能块，让许多系统使用。

类实用程序显示为框图中带阴影的符号如下：



要增加类实用程序：

1. 选择Class Utility 工具栏按钮。
2. 单击框图中任一位置加进新类。
3. 输入类名。

或

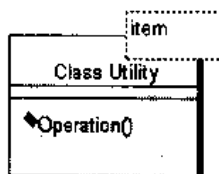
1. 用上述方法将一个类加进Class框图或浏览器中。
2. 打开类规范窗口。
3. 在Type字段中输入Class Utility。
4. 按OK。

或

1. 选择Tools►Create►Class Utility。
2. 单击框图中任一位置加进新类。
3. 输入类名。

增加参数化

参数化是个参数化类，包含一组操作，是生成类实用程序的模板，在Class框图中用下列符号表示：



要增加参数化：

1. 选择Parameterized Class Utility 工具栏按钮。

2. 单击框图中任一位置加进新类。

3. 输入类名。

或

1. 用上述方法将一个类加进Class框图或浏览器中。

2. 打开类规范窗口。

3. 在Type字段中输入Parameterized Class Utility。

4. 按OK。

或

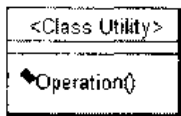
1. 选择Tools>Create>Parameterized Class Utility。

2. 单击框图中任一位置加进新类。

3. 输入类名。

增加实例化类实用程序

实例化类实用程序是设置了参数值的参数化类实用程序，在Class框图中用下列符号表示：



要增加实例化类实用程序：

1. 选择Instantiated Class Utility工具栏按钮。

2. 单击框图中任一位置加进新类。

3. 输入类名。

或

1. 用上述方法将一个类加进Class框图或浏览器中。

2. 打开类规范窗口。

3. 在Type字段中输入Instantiated Class Utility。

4. 按OK。

或

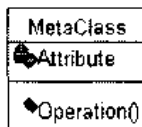
1. 选择Tools>Create>Instantiated Class Utility。

2. 单击框图中任一位置加进新类。

3. 输入类名。

增加元类

元类（metaclass）的实例是类而不是对象。参数化类和参数化类实用程序就是元类。UML中，元类显示如下：



要增加元类:

1. 用上述方法将类加进Class框图或浏览器中。
2. 打开类规范窗口。
3. 在Type字段输入MetaClass。
4. 单击OK。

命名类

Rose中类型中的每个类应有唯一名称。大多数单位都有一定的类命名规则。但一般来说,类用单数名词命名。例如,在我们的员工跟踪系统中,有个employee类,还有个position类,而不称为employee和position类。

类名通常不包括空格,这样能提高可读性,大多数编程语言不支持类名中的空格。类名应尽可能短。尽管ListOfEmployeesThatAreOnProbation能很好地描述类的作用,但会使代码可读性很差,用EmployeeList可能更好些。

使用大写或小写取决于单位的类命名规则,例如员工表的类可以命名为employeeelist、Employeeelist、EmployeeList或EMPLOYEELIST。每个公司通常有一个命名规则,但确定的方法应当用于所有类。

要命名类:

1. 选择浏览器或Class框图中的类。
2. 输入类名。

或

1. 打开类规范窗口。
2. 在Name字段中输入类名。

要将文档加进类中:

1. 选择浏览器中的类。
2. 在文档窗口输入类文档。

或

1. 打开类规范窗口。
2. 在规范窗口的Documentation区输入信息。

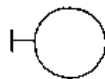
指定类版型

版型机制可以将类进行分类。例如,假设要迅速寻找模型中的所有窗体,可以生成form版型,将所有窗口指定为这个版型。要寻找模型中的所有窗体时,只要寻找form版型的类即可。

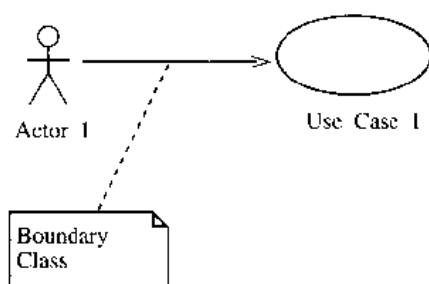
UML中有三种主要类版型: Boundary、Entity和Control。

边界类

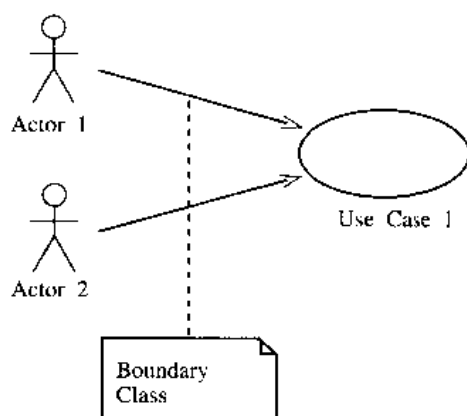
边界类(Boundary classes)位于系统与外界的交界处,包括所有窗体、报表、与打印机和扫描仪等硬件的接口,以及与其他系统的接口。UML将边界类表示如下:



要寻找边界类，可以检查Use Case框图。每个角色/使用案例交互至少要有个边界类。边界类使角色能与系统交互。



注意，并非每个角色/使用案例对要生成唯一边界类。例如，下面是两个角色启动同一使用案例，可以用同一边界类与系统通信。



通过Use Case框图可以确定需要的边界类。

实体类

实体类（Entity classes）保存要放进持续存储体的信息。在员工跟踪系统中，Employee类是实体类的范例。实体类通常在事件流和Interaction框图中，是对用户最有意义的类，通常用业务域术语命名。

在UML中，实体类用下列尺寸表示：



通常，数据库中可以对每个实体类生成一个表格。这里是从系统开发标准方法派生的中一变形。与其先定义数据库结构，也可以从对象模型中收集的信息开发数据库结构。这样就可以从数据库字段回溯要求。要求确定事件流，事件流确定对象和类，以及类的属性。实体类中的每个属性都是数据库结构中的字段。利用这个方法，可以从数据库字段回溯要求，从而避免收集没人使用的信息。

控制类

控制类（Control classes）是我们要介绍的最后一种版型，负责协调其他类的工作。每

个使用案例通常有一个控制类，控制使用案例中的事件顺序。在Interaction框图中，控制类具有协调责任，如图5.9。

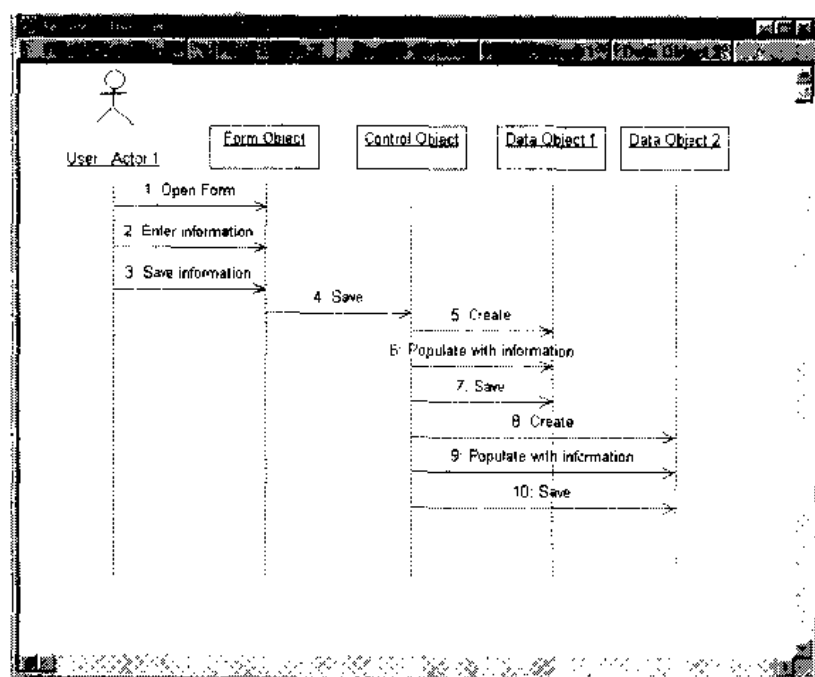


图5.9 Sequence框图中的控制类

注意控制类本身不完成任何功能，其他类并不向控制类发送许多消息，而是由控制类发出许多消息。使用案例只是向其他类委托责任。为此，控制类也称为管理者类。在UML中，控制类用下列符号表示：



可能还有许多控制类是在多个使用案例间共用的。例如，**SecurityManager**负责控制与安全有关的事件，**TransactionManager**类负责协调与数据库事务有关的消息。还有其他处理公用功能的管理者，如资源竞争、分布式处理和错误处理。

这些控制类可以分离系统使用的功能。例如，将安全协调包装成**SecurityManager**类可以减少改变的影响。如果安全逻辑的顺序需要改变，只影响**SecurityManager**类。

除了上述版型外，还可以向模型中增加自己的版型。在Stereotype字段中，可以输入新版型，使它成为当前Rose模型中的版型。如果喜欢，可以增加所有Rose模型中使用的版型，也可以对新版型生成工具栏按钮和版型图标，具体过程将在下面介绍。

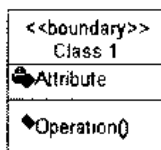
要指定类版型：

1. 打开类规范窗口。
2. 从下拉列表框图选择版型，或输入版型名。

要在框图上显示版型名：

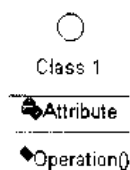
1. 右单击Class框图上的类。

2. 从弹出菜单选择Options►Stereotype Display►Label, 版型名出现在类名上面, 放在<<>>中。



要在框图上显示版型图标:

1. 右单击Class框图上的类。
2. 从弹出菜单选择Options►Stereotype Display►Icon
3. 类的表示变为相应图标。例如, 下面是接口类的图标:



说明: 并非所有版型都有图标。如果版型没有图标, 则框图中只显示版型名。

要关掉框图上显示版型:

1. 右单击Class框图上的类。
2. 从弹出菜单选择Options►Stereotype Display►None。类规范窗口中仍然有版型, 但框图中不显示版型。

要改变缺省版型显示选项:

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 在图5.10所示的Compartment区中, 选择或取消Show Stereotypes复选框, 控制是否显示版型。
4. 在Stereotype Display区中选择缺省显示类型 (None、Label或Icon)。

要在当前Rose模型中增加新版型:

1. 打开类规范窗口。
2. 在Stereotype字段中输入新版型。增加新类时, 新版型即可出现在下拉列表框中, 但只在当前Rose模型中。

要在所有Rose模型中增加新版型:

1. 退出Rose。
2. 在文件DefaultStereotypes.ini中, 寻找[Stereotyped Items]部分。
3. 在[Stereotyped Items]部分中, 增加新版型项目。例如, 要增加边界版型, 可以加上:

```

[Stereotyped Items]
Class:Boundary
  
```

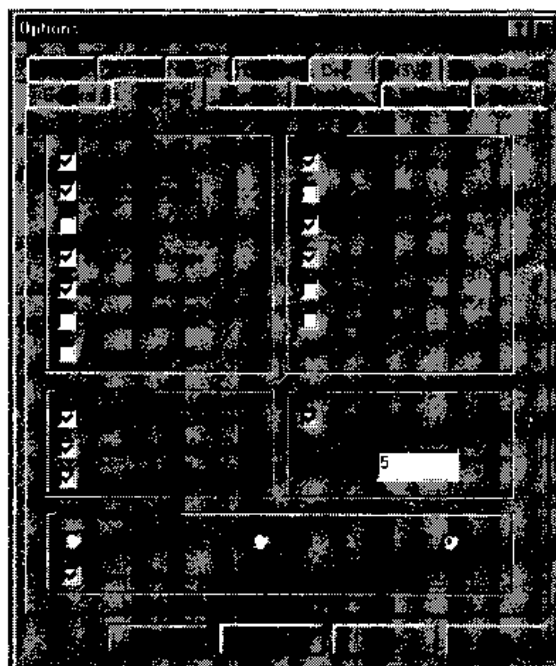


图5.10 改变缺省显示版型

4. 在DefaultStereotypes.ini文件中加入新版型项目部分。在这个部分中，要让Rose知道版型用于类、使用案例或什么，并命名版型。对于边界版型，可以增加：

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
```

要增加版型的新框图图标：

1. 退出Rose。
2. 用Windows元文件（.WMF）格式或扩展元文件（.EMF）格式生成框图图标。
3. 打开文件DefaultStereotypes.ini。
4. 到版型部分。例如，要增加边界版型的图标，到[Class:Boundary]部分。
5. 加上这个部分的项目，指定图标所在的元文件名。例如：

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
→ Metafile=c:\program files\rational rose 98\boundary.emf
```

可以用Windows元文件（.WMF）格式或扩展元文件（.EMF）格式。如果元文件与DefaultStereotypes.ini文件在同一目录，则可以用&而不用完整路径名。例如：

```
Metafile=&\boundary.emf
```

6. 如果用标准Windows元文件（.WMF）格式而不用扩展元文件（.EMF）格式，则要在这个部分加上两行ExtentX和ExtentY，让Rose知道图标的宽度和高度。例如，逻辑版型部分如下：

```

[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
→ ExtentX=20
→ ExtentY=20

```

7. 这样，在Rose模型中使用版型时，可以显示版型图标。例如，Rose中的边界版型可以使用UML边界图注：



8. 要缺省显示版型图标，加上下列行，否则版型根据模型缺省设置而显示：

```

[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
→ ExtentX=20
→ ExtentY=20
→ StereotypeIconStyle=IconTo add a small toolbar button for a stereotype:

```

要增加版型的小工具栏图标：

1. 退出Rose。
2. 用图形应用程序生成小工具栏图标。将图标保存为位图（.BMP）格式。图标为15像素高，16像素宽，灰色背景（在Rational Rose中为RGB = 192, 192, 192）。
3. 打开文件DefaultStereotypes.ini。
4. 到版型部分。例如，要增加边界版型的小工具栏图标，到[Class:Boundary]部分。
5. 加入小工具栏图标的行。如果.BMP文件与DefaultStereotypes.ini文件在同一目录中，则可以用&代替路径：

```

[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
ExtentX=20
ExtentY=20
→ SmallPaletteImages=&\smalltoolbar.bmp

```

6. 如果要用的图标是一个.BMP文件中的几个图标之一，则在DefaultStereotypes.ini文件中再加一行，告诉Rose图标在.BMP文件中的位置：

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
ExtentX=20
ExtentY=20
SmallPaletteImages=&\smalltoolbar.bmp
→ SmallPaletteIndex=1
```

说明：位图文件长度限于512KB。如果文件超过512K，则应分解成较小的文件。

7. 选择Tools►Options。
8. 选择Toolbars标签。
9. 在Customize 工具栏区按住要定制的工具栏旁边的…按钮。
10. 在Class框图工具栏中增加新的工具栏按钮。

要增加版型的大工具栏按钮：

1. 退出Rose。
2. 用图形应用程序生成大工具栏按钮。将图标保存为位图（.BMP）格式。图标为24像素高，24像素宽，灰色背景（在Rational Rose中为RGB = 192, 192, 192）。
3. 打开文件DefaultStereotypes.ini。
4. 到版型部分。例如，要增加边界版型的大工具栏按钮，到[Class:Boundary]部分。
5. 加入大工具栏按钮的行。如果.BMP文件与DefaultStereotypes.ini文件在同一目录中，则可以用&代替路径：

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
ExtentX=20
ExtentY=20
→ MediumPaletteImages=&\largetoolbar.bmp
```

6. 如果要用的图标是一个.BMP文件中的几个图标之一，则在DefaultStereotypes.ini文件中再加一行，告诉Rose图标在.BMP文件中的位置：

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
ExtentX=20
ExtentY=20
MediumPaletteImages=&\largetoolbar.bmp
→ MediumPaletteIndex=1
```

7. 选择Tools►Options。
8. 选择Toolbars标签。

9. 在Customize 工具栏区按住要定制的工具栏旁边的...按钮。

10. 在Class框图工具栏中增加新的工具栏按钮。

要增加版型的浏览器图标:

1. 退出Rose。

2. 用图形应用程序生成图标。将图标保存为位图 (.BMP) 格式。图标为16像素高, 16像素宽, 白色背景。

3. 打开文件DefaultStereotypes.ini。

4. 到版型部分。例如, 要增加边界版型的大工具栏按钮, 到[Class:Boundary]部分。

5. 加入大工具栏按钮的行。如果.BMP文件与DefaultStereotypes.ini文件在同一目录中, 则可以用&代替路径:

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
ExtentX=20
ExtentY=20
→ ListImages=&\listimages.bmp
```

6. 如果要用的图标是一个.BMP文件中的几个图标之一, 则在DefaultStereotypes.ini文件中再加一行, 告诉Rose图标在.BMP文件中的位置:

```
[Class:Boundary]
Item=Class
Stereotype=Boundary
Metafile=c:\program files\rational rose 98\boundary.wmf
ExtentX=20
ExtentY=20
ListImages=&\listimages.bmp
→ ListIndex=3
```

7. 生成这种版型的新项目时, 它在浏览器中用这个新图标显示。

设置类可见性

Visibility选项确定包外能否访问这个类, 有三个选项:

Public 系统中所有其他类都能访问这个类。

(98i) Protected, Private 这个类可以在嵌套类、朋友或同一个类中访问。

Package or Implementation 这个类只能由同一包中的其他类访问。

要设置类可见性:

1. 右单击浏览器或Class框图中的类。

2. 从弹出菜单中选择Open Specification。

3. 将输出控制设置为Protected, Private或Implementation。

设置类基数

Cardinality字段可以设置类的实例数。在员工跟踪系统中，我们的员工类可能有多个实例，如John Doe、Bill Smith等等。因此，Employee类的基数为n。

而控制类的基数通常为1。运行应用程序时，可能只要一个安全管理器。

在Rose中，下拉列表框提供下列基数选项：

Cardinality (基数)	含义
n (缺省)	多
0..0	0
0..1	0或1
0..n	0或多
1..n	1
1..1	1或多

也可以用下列格式输入基数：

格式	含义
<number>	<number>
<number 1>..<number 2>	<number 1>到<number 2>
<number>..n	<number>或多
<number 1>,<number 2>	<number 1>或<number 2>
<number 1>,<number 2>..<number 3>	<number 1>或<number 2>到<number 3>
<number 1>..<number 2>,<number 3>..	<number 1>到<number 2>或<number 3>到
<number 4>	<number 4>

要设置类的基数：

1. 打开类规范窗口。
2. 选择Detail标签。
3. 在基数下拉列表框中，选择基数，或输入下拉列表框中没有的基数选项。

设置类的存储要求

建立模型时，可能要注明每个类对象要求的相对或绝对内存量。类规范窗口中的Space字段设置类的存储要求。类实用程序、实例化类实用程序和参数化类实用程序不能使用Space字段。

要设置类的空间：

1. 打开类规范窗口。
2. 选择Detail标签。
3. 在Space字段中输入类的存储要求。

设置类持续性

Rose中可从模型产生DDL (Data Definition Language)。DDL定义数据库的结构。产生DDL时，Rose寻找设置为持续的类。类规范窗口Persistence字段可以指定类为：

Persistent 类在应用程序执行之外生存。即类对象中的信息存放在数据库或别的永久存储体中。

Transient 类对象中的信息不存到永久存储体中。

类实用程序、实例化类实用程序和参数化类实用程序不能使用Persistence字段。

要设置类持续性：

1. 打开类规范窗口。
2. 选择Detail标签。
3. 在Persistence区中选择Persistent或Transient。

设置类并发性

并发性描述类在存在多个控制线程时的表现。有四种选项：

Sequential 是缺省设置，只有一个控制线程时，类正常工作（即如期操作），而在有多个控制线程时则不能保证。

Guarded 存在多个控制线程时，类正常工作但不同线程中的类应相互协作，保证不会互相干扰。

Active 类有自己的控制线程。

Synchronous 存在多个控制线程时，类正常工作不需要与其他类相互协作，因为类本身能处理互斥情形。

要设置类并发性：

1. 打开类规范窗口。
2. 选择Detail标签。
3. 在concurrency区选择并发性单选钮。

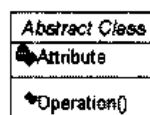
生成抽象类

抽象类（abstract class）是不能实例化的类。换句话说，如果类A是抽象的，则内存中不会有类型为A的对象。

抽象类通常在继承结构中使用，保存一些其他类共同的信息和行为。例如，Animal类有属性height、color和species。从这个类可以派生Cat、Dog和Bird等类。这些类都从Animal类继承height、color和species属性，并有自己的属性和操作。

应用程序运行时，不会生成Animal对象，所有对象都是Cat、Dog和Bird。Animal是个抽象类，只是保存Cat、Dog和Bird的共性。

在UML中，抽象类用带斜体的名称在Class框图中显示：



要生成抽象类：

1. 按上述方法生成类。
2. 打开类规范窗口。

3. 选择Detail标签。
4. 复选Abstract框。

浏览类属性

下一章要介绍增加、删除和使用类属性。类规范窗口中可以设置已经为类生成的属性。关于属性和操作的其他信息，见第6章。

要浏览类属性：

1. 打开类规范窗口。
2. 选择Attributes标签，其中列出类的属性，包括属性可见性、版型、名称数据类型和缺省值。

浏览类操作

下一章要介绍增加、删除和使用类操作。类规范窗口中可以设置已经为类生成的操作。关于属性和操作的其他信息，见第6章。

要浏览类操作：

1. 打开类规范窗口。
2. 选择Operations标签，其中列出类的操作，包括操作可见性、版型、签名和返回类型。

浏览类关系

第7章将详细介绍类中可以增加的不同关系细节。我们将介绍增加与删除关系、设置每个关系的详细信息。在类规范窗口中，可以浏览加进类中的所有关系。关于类之间关系的详细信息，见第7章。

要浏览类关系：

1. 打开类规范窗口。
2. 选择Relations标签，其中列出这个类参与的所有关系。

使用嵌套类

在Rose中，一个类可以嵌套另一个类，嵌套类中又可以嵌套类，深度随意。

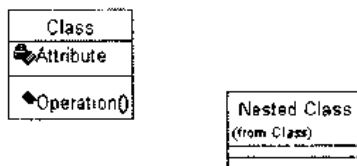
要生成嵌套类：

1. 打开父类的类规范窗口。
2. 选择Nested标签。
3. 右单击Nested标签中任一空白处。
4. 从弹出菜单中选择Insert。
5. 输入嵌套类名称。

要在Class框图中显示嵌套类：

1. 打开Class框图。
2. 选择Query►Add Classes。
3. 将嵌套类从Classes列表框移到Selected Classes列表框。嵌套类的显示格式为Parent-Class::NestedClass。

4. 单击OK，嵌套类出现在框图中，父类名放在括号内。



要从模型中删除嵌套类：

1. 打开Class框图。
2. 选择Query►Add Classes。
3. 右单击要删除的嵌套类名。
4. 嵌套类Delete，即可从所有Class框图中删除嵌套类。

将文件和URL连接类

前面曾介绍过，可以将文件和URL连接Class框图。也可以直接将文件和URL连接类。例如，可以将源代码文件连接模型中的类，或者将测试类功能的测试脚本连接类。Rose可以通过浏览器或类规范窗口将文件和URL连接类。

要将文件连接类：

1. 打开类规范窗口。
2. 选择Files标签。
3. 右单击Files标签中任一空白处。
4. 嵌套类Insert File。
5. 用Open对话框寻找要连接的文件。
6. 选择Open将文件连接类。

或

1. 右单击浏览器中的类。
2. 选择New►File。
3. 用Open对话框寻找要连接的文件。
4. 选择Open将文件连接类。

要将URL连接类：

1. 右单击浏览器中的类。
2. 选择New►File。
3. 用Open对话框寻找要连接的URL。
4. 选择Open将URL连接类。
5. 输入要连接的URL名。

或

1. 右单击浏览器中的类。
2. 选择New►URL。
3. 输入要连接的URL名。

要打开连接的文件:

1. 找到浏览器中的文件。
2. 双击文件名。Rose自动启动相应应用程序并装入文件。

或

1. 右单击浏览器中的文件。
2. 从弹出菜单中选择Open。Rose自动启动相应应用程序并装入文件。

或

1. 打开Class框图。
2. 选择Query►Add Classes。
3. 双击要打开的文件。Rose自动启动相应应用程序并装入文件。

或

1. 打开Class框图。
2. 选择Query►Add Classes。
3. 右单击要打开的文件。
4. 从弹出菜单中选择Open File/URL。Rose自动启动相应应用程序并装入文件。

要打开连接的URL:

1. 找到浏览器中的URL。
2. 双击URL名。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 右单击浏览器中的URL。
2. 从弹出菜单中选择Open。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 打开Class框图。
2. 选择Query►Add Classes。
3. 双击要打开的URL。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 打开Class框图。
2. 选择Query►Add Classes。
3. 右单击要打开的URL。
4. 从弹出菜单中选择Open File/URL。Rose自动启动Web浏览器应用程序并装入URL。

要删除连接的文件或URL:

1. 右单击浏览器中的文件或URL。
2. 从弹出菜单选择Delete。

浏览包含类的Interaction框图

要改变类时, 最好知道这个类在系统中的位置。两种Interaction框图、Sequence和Collaboration框图可以知道每个类的位置和用法。

可以用Report菜单显示哪个Sequence和Collaboration框图包含特定类的对象。

要浏览包含某个类的所有Sequence和Collaboration框图：

1. 选择Class框图中的类。
2. 选择Report►Show Instances。
3. Rose显示包含该类对象的所有Sequence和Collaboration框图，如图5.11。要打开一个框图，在清单中双击这个框图，或单击Browse按钮。

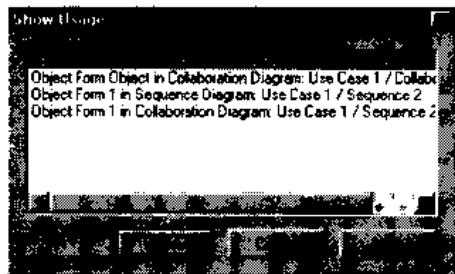


图5.11 浏览类实例

使用图注

可以在Class框图中加上图注，提供特定类、包、属性、操作或关系的详细信息。这些图注并不影响代码生成，但有助于开发人员和其他项目组成员更好地了解模型。

增加图注

有两种工具可以在Class框图中加上图注。图注是连接框图中类或其他项目的说明。文本框可以加进适用于整个框图的说明。例如，可以用文本框在框图中加上标题。

要在框图中增加图注：

1. 选择Note工具栏按钮。
2. 单击框图中任一空白处，加进图注。
3. 输入图注文本。

要将图注连接项目：

1. 选择Anchor Note to Item工具栏按钮。
2. 将鼠标从图注拖动到项目。
3. Rose在图注与项目之间画一条虚线。

要将文件框加进框图：

1. 选择Text Box工具栏按钮。
2. 单击框图中任一空白处，加进文本框。
3. 输入文本。

删除图注

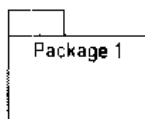
框图中的图注有时需要删除。在Rose中，可以直接在Class框图中删除图注。

要删除框图中的图注或文本框：

1. 选择框图中的图注或文本框。
2. 按Delete键。

使用包

包将具有一些共性的类组合在一起。在UML中，包用下列符号表示：



包装类时有几个常用方法，但也可以用别的方法。一个方法是按版型组合类。利用这个方法时，可以实体类一个包、边界类一个包、控制类一个包等等。这在部署时非常方便，所有要访问客户机的边界已经包装在一起。

另一种方法是按功能组合类。例如，可能有个Security包，放有所有涉及应用程序安全的类，还有Employee Maintenance、Reporting或Error Handling包。这种方法的好处是可复用。如果将类认真组合，就可以得到相互独立的包。本例中，只要选择Security包，即可在其他应用程序中复用。

最后，可以用这些方法的组合。包可以通过嵌套进一步组织类。高层可以按功能组织，生成Security包。而Security包中又可以按功能或版型组织安全类，形式子包。

增加包

生成模型的下一步是增加一些包。类包在浏览器的Logical视图中生成。

要将现有包加进Class框图中：

将包从删除器中拖动到Class框图中。

要将新包加进Class框图：

1. 选择Package工具栏按钮。
2. 单击Class框图中任一空白处放上包。
3. 输入包名。

要将包加进浏览器中：

1. 右单击浏览器中的Logical视图。要在现有包中生成子包，右单击浏览器中的既有包。
2. 选择New►Package。
3. 输入包名。

要将项目移进包中：

在浏览器中，将项目从现有位置拖动到新包中。

删除包

可以从Class框图或从整个模型中删除包。如果从整个模型中删除包，则包及其所有内容均被删除。

要从Class框图删除包：

1. 选择Class框图中的包。
2. 按Delete键。

3. 注意这个包从这个Class框图删除，但在浏览器和其他Class框图中仍然存在。

要从整个模型中删除包：

1. 右单击浏览器中的包。
2. 从弹出菜单选择Delete。

或

1. 选择Class框图中的包。
2. 选择Edit►Delete from Model或按Ctrl+D。

警告：如果从整个模型中删除包，则包及其所有内容均被删除。

练习

本练习取上次生成的类，将其组成包。然后生成Class框图，显示系统和包中的类。

问题

通过Interaction框图，Bob可以看出系统符合公司的业务需求。因此Susan¹与开发主任Karen商量。

“这是增加新订单的Interaction框图”。

“不错，我准备开发”。

Karen看看Rose模型中的类，决定将其按版型组合。因此她生成Entities、Boundaries和Control包并将每个类移到相应包中。然后她在每个包中生成一个Main Class框图，显示包；一个Enter New Order Class框图，显示该使用案例的所有类。

生成Class框图

将前面介绍的类组成包。生成Class框图以显示包，另一Class框图显示Enter New Order使用案例的所有类。

练习步骤

设置

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 确保选择Show Stereotypes复选框。
4. 确保选择Show All Attributes和Show All Operations复选框。
5. 确保取消Suppress Attributes和Suppress Operations复选框。

生成包

1. 右单击浏览器中的Logical视图。
2. 选择New►Package。
3. 将新包取名Entities。

4. 重复1~3步，生成Boundaries和Control包。
这时浏览器如图5.12。

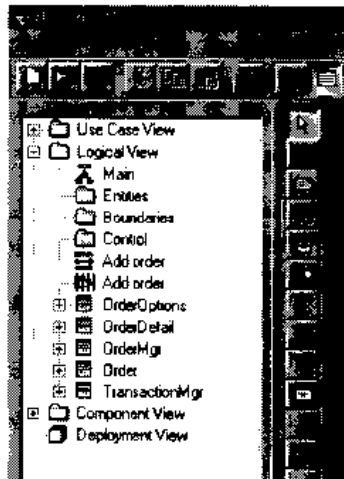


图5.12 订单处理系统的包

生成Main Class框图

1. 双击浏览器中Logical视图内的Main Class框图将其打开。
2. 将Entities包从浏览器中拖动到框图上。
3. 将Boundaries和Control包从浏览器中拖动到框图上。

Main Class框图如图5.13。

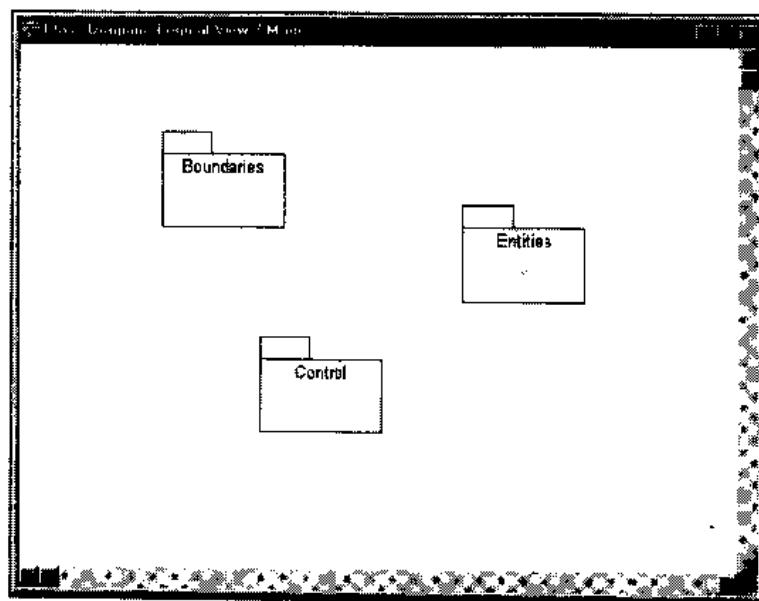


图5.13 订单处理系统的Class框图

生成包含Enter New Order使用案例中所有类的Class框图

1. 右单击浏览器的Logical视图。
2. 选择New►Class Diagram。

3. 将新的Class框图取名Add New Order。
4. 双击Add New Order Class框图将其打开。
5. 将每个类（OrderOptions、OrderDetail、Order、OrderMgr、TransactionMgr）从浏览器拖动到框图。

Class框图如图5.14。

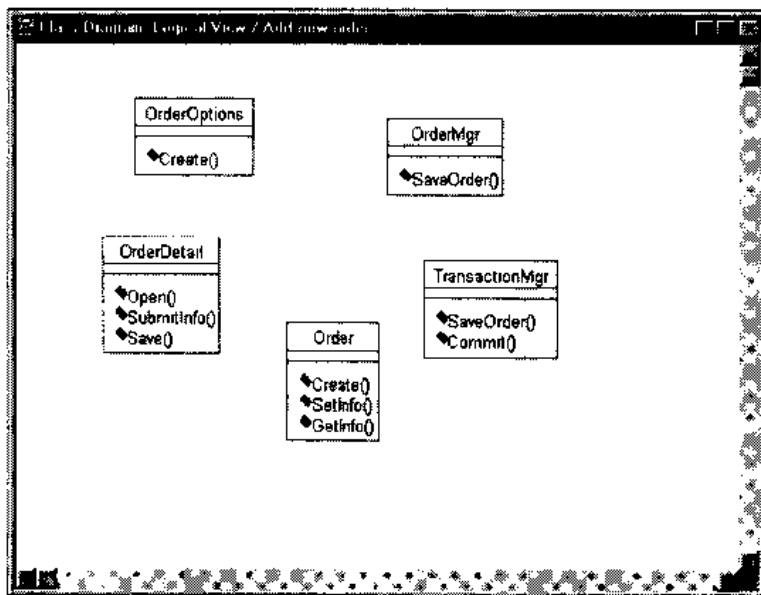


图5.14 Add New Order Class框图

将版型加进类中

1. 右单击框图中的OrderOptions类。
2. 从弹出菜单中选择Open Specification。
3. 在版型字段中输入Boundary。
4. 单击OK。
5. 右单击框图中的OrderDetail类。
6. 从弹出菜单中选择Open Specification。
7. 在版型字段下拉列表框中出现Boundary版型，选择Boundary。
8. 单击OK。
9. 重复1~4步，指定OrderMgr和TransactionMgr类为Control版型，Order类为Entity版型。

这时的Class框图如图5.15。

将类组成包

1. 在浏览器中，将OrderOptions类拖动到Boundaries包。
2. 将OrderDetail类拖动到Boundaries包。
3. 将OrderMgr和TransactionMgr类拖动到Control包。
4. 将Order类拖动到Entity包。

浏览器中的类和包如图5.16。

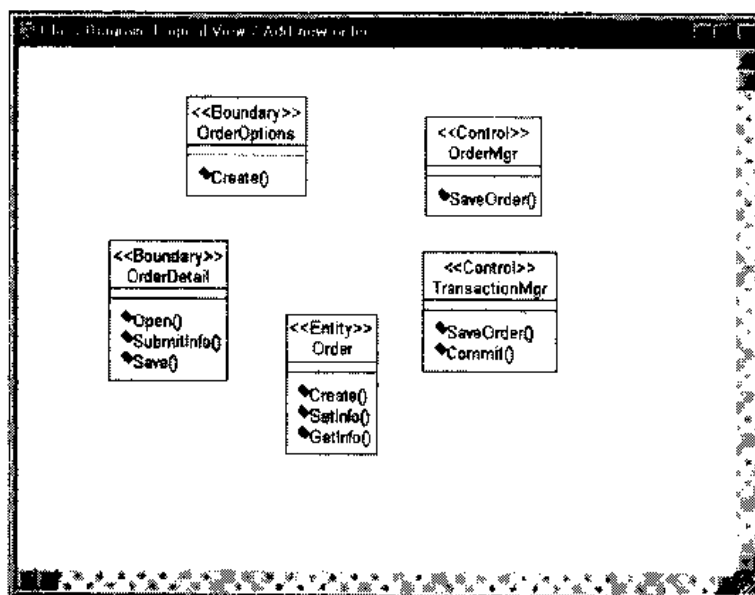


图5.15 Add New Order使用案例的类版型

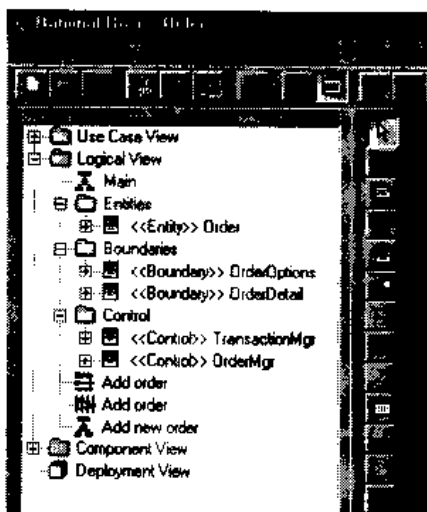


图5.16 Add New Order使用案例的类和包

将Class框图加进每个包

1. 在浏览器中，右单击Boundaries包。
 2. 选择New►Class Diagram。
 3. 将新框图取名Main。
 4. 双击打开新框图。
 5. 将OrderOptions和OrderDetail类从浏览器拖动到框图中。
- Boundaries包的Main Class框图如图5.17。
6. 关闭框图。
 7. 在浏览器中，右单击Entities包。
 8. 选择New►Class Diagram。
 9. 将新框图取名Main。

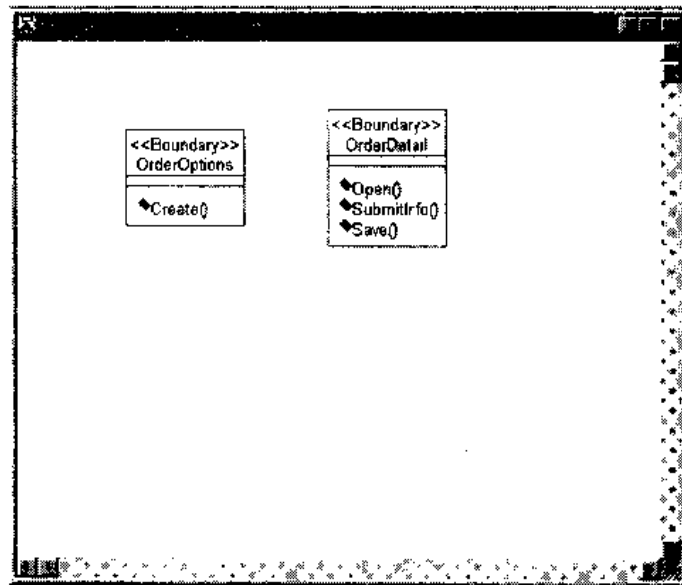


图5.17 Boundaries包的Main Class框图

10. 双击打开新框图。
 11. 将Order类从浏览器拖动到框图中。
- Entities包的Main Class框图如图5.18。

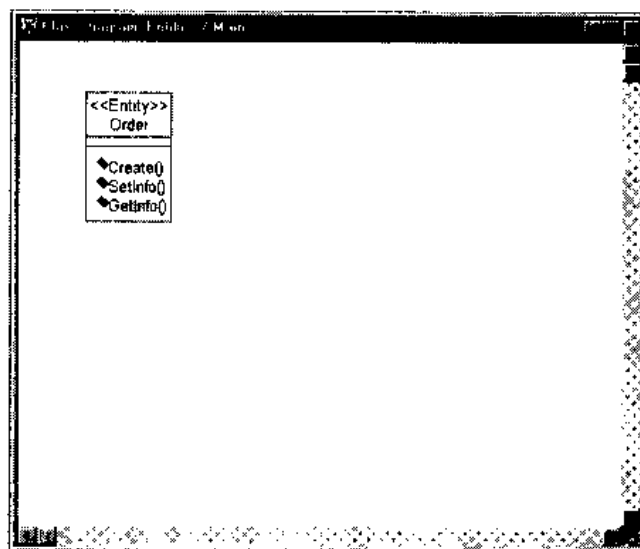


图5.18 Entities包的Main Class框图

12. 关闭框图。
13. 在浏览器中，右单击Control包。
14. 选择New>Class Diagram。
15. 将新框图取名Main。
16. 双击打开新框图。

17. 将OrderMgr和TransactionMgr类从浏览器拖动到框图中。
18. 关闭框图。

Control包的Main Class框图如图5.19。

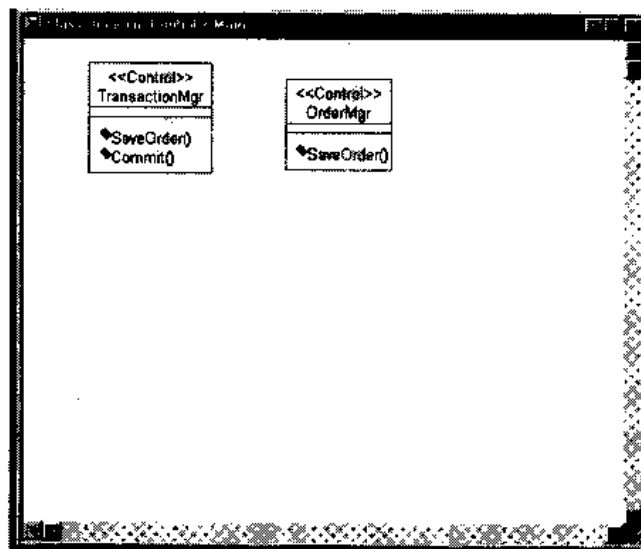


图5.19 Control包的Main Class框图

小结

本章介绍了类、Class框图和包。现在我们已经大大改进了所建系统的视图。到上一章结束时，我们已经建立了Collaboration和Sequence框图等Object Interaction框图。这些框图演示了系统完整功能所需的交互。本章介绍了Class框图，显示系统的静态行为，还介绍了将类组成包以便更好地理解系统。我们将在下一章介绍类的属性和操作。

第6章 属性与操作

- 使用属性
- 使用操作
- 在Class框图中显示属性和操作
- 将操作映射消息

上一章介绍了类与包。记注，类包装了属性（数据）和作用于这些属性的操作（行为）。现在要介绍属性和操作。首先介绍如何寻找属性，然后将其加进Rose模型中，再增加属性的细节。然后要介绍如何寻找操作，将其加进Rose模型中再增加操作的细节。接着要在Class框图中显示属性和操作。最后，我们要介绍如何在Interaction框图中将操作映射消息。

使用属性

属性（attribute）是与类相关联的信息。例如，Company类可能有属性Name、Address和NumberOfEmployees。

Rose中可以对模型中的每个类加进一个或几个属性。下面几节介绍如何寻找属性，将其加进Rose模型中，再增加属性的细节。

寻找属性

属性的来源有许多。首先可以查阅使用案例文档，寻找事件流中的名词。有些名词是对象或类，有些是角色，有些则是属性。例如，从下列事件流：“The user enters the employee's name, address, social security number, and phone number”可以看出，Employee类有属性Name、Address、SSN和Phone。

另一个来源是要求文档。要求中可能会介绍系统要收集哪些信息。收集的信息就是类的属性。

最后，可以检查数据库结构。如果已经定义数据库结构，则表中字段就是属性。通常，数据库表格和实体类具有很好的一一对应关系。以上例为例，Employee表格可能有字段Name、Address、SSN和Phone。注意，数据库表格和实体类不一定总有一一对应关系。设计数据库结构与设计类时具有不同考虑。例如，关系型数据库并不直接支持继承。

但是，定义属性时，一定要保证每个属性可以回溯要求。这可以解决应用程序捕获大量无用信息的问题。每个要求都应回溯使用案例的事件流、特定要求或现有数据库表格。如果无法回溯要求，则不能保证客户需要它。这与传统的方法有所不同，不是先建立数据库结构再处理系统，而是同时建立系统和数据库结构，以符合相同的要求。

标识属性时，应将其赋予适当的类。属性是与类相关联的信息。例如，Employee类可能有姓名和地址，但不应包括员工的公司所生产产品的信息，这个信息应放在Product类中。

类的属性不宜太多。如果某个类的属性太多，最好将其分解成更小的类。如果一个类的属性超过10到15个，则最好检查一下。这个类可能是合法的，所有属性都需要，都属于这个类。同样，属性也不要太少，但这也可能是合法的，比如控制类通常只有很少属性，但是，属性太少的类也许可以合并在一起。如果一个类只有一、两个属性，则应检查一下。

有时可能遇到一个信息，不知道是属性还是类。例如Company Name，是Person类的属性，还是Company本身是个类？答案取决于所编写的应用程序。如果要跟踪公司的信息，有一些与公司相关联的行为，则应作为类。例如，系统可能要跟踪客户。这时就要保持向其销售产品或服务的公司信息。你可能要知道该公司有多少员工，公司名和地址、公司的联系人姓名等等。

另一方面，你可能不需要知道公司的具体信息。如果所编写的应用程序向其他公司的联系人生成信函，则只要知道公司名，而不需要知道公司的具体信息，这时，可以把公司名作为Contact类的属性。

另一点要考虑的是这个信息有没有行为。如果公司在你所编写的应用程序中有一定的行为，则应作为类，否则可以作为模型。

标识属性之后，下一步要将其加进Rose模型中。下面几节介绍如何增加属性和属性细节，如数据类型与缺省值。

增加属性

标识属性之后，下一步要将其加进Rose模型中。每个属性有三个主要信息：属性名、数据类型和缺省值。生成模型代码前，必须提供每个属性的属性名和数据类型，初始值是可选的。

增加属性的方法有三种，可以直接将属性加进Class框图，用浏览器增加属性或用类规范窗口增加属性。

增加属性后，可以将文档加进其中。通常，属性文档包括属性的简短说明或定义。属性文档显示为模型所产生代码中的说明语句。通过建档属性，就开始了代码建档。

要将属性加进类中：

1. 右单击Class框图中的类。
2. 选择New►Attribute。
3. 输入属性名，用Name:Data Type=Initial value格式。例如：

Address : String

IDNumber : Integer = 0

要生成代码，必须提供每个属性的属性名和数据类型，初始值是可选的。

4. 要增加更多属性，按Enter并直接在Class框图中输入新属性。

或

1. 右单击浏览器中的类。
2. 选择New►Attribute。
3. 浏览器的类中出现新属性name。输入新属性名。数据类型与缺省值可以不在浏览器中输入，而是到Class框图中再输入，见稍后介绍。

或

1. 打开类规范窗口。
2. 选择Attributes标签。如果已经有一些属性，则会在这里列出。
3. 右单击属性区任一位置，如图6.1。

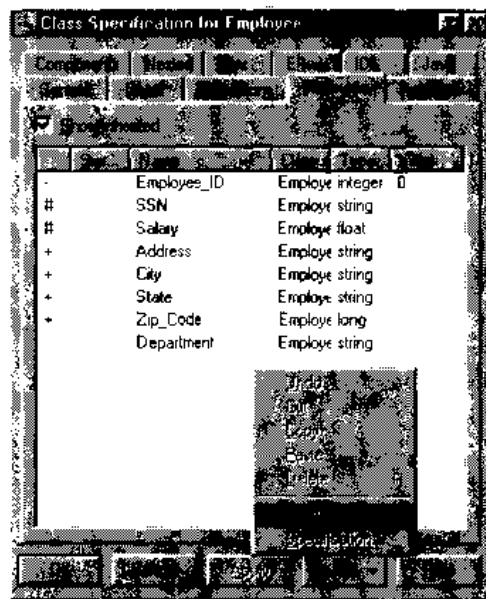


图6.1 用类规范窗口增加属性

4. 从弹出菜单选择Insert。
5. 输入新属性名。
6. 输入可见性、版型、数据类型和初始值。下面几节将介绍这些细节。
要将文档加进属性中：

1. 选择浏览器中的属性。
2. 在文档窗口中输入属性文档。

或

1. 选择Class框图中的属性。
2. 在文档窗口中输入属性文档。

或

1. 右单击浏览器中的属性。
2. 从弹出菜单中选择Open Specification。
3. 在类属性规范窗口的Documentation区输入属性文档。

或

1. 打开类文档窗口。
2. 选择Attributes标签。
3. 选择属性。
4. 在文档窗口中输入属性文档。

删除属性

有时要删除前面生成的属性。例如系统要求改变时可能要删除某个属性。Rose中删除属性的最简单方法是通过浏览器，但也可以用Class框图删除。从Class框图删除属性时，Rose自动从模型中和其他Class框图将其删除。

要删除类属性：

1. 右单击浏览器中的属性。
2. 从弹出菜单选择Delete。

或

1. 选择Class框图中的属性。
2. 用退格键删除框图中的属性名、数据类型和初始值。
3. 单击框图中任一位置。
4. Rose先确认删除，再删除属性。

或

1. 打开类文档窗口。
2. 选择Attributes标签。
3. 右单击要删的属性。
4. 从弹出菜单选择Delete。
5. Rose先确认删除，再删除属性。

属性规范

和其他Rose模型元素一样，可以在属性中增加许多细节规范，如属性可见性、版型、数据类型和初始值。下面几节介绍每个规范。

所有规范均在图6.2所示的属性规范窗口浏览和改变。

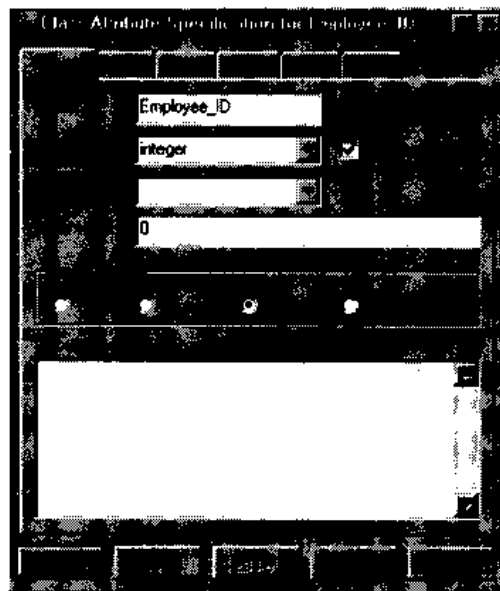


图6.2 属性规范窗口

要打开属性规范窗口:

1. 右单击浏览器中的属性。
2. 从弹出菜单中选择Open Specification。

或

1. 打开类文档窗口。
2. 选择Attributes标签。
3. 双击相应属性。

说明: 在Rose 98i中, 上述过程打开不同的规范窗口。要打开图6.2所示的属性规范窗口, 右单击浏览器中的属性并从弹出菜单中选择Open Standard Specification。

设置属性数据类型

属性要指定的信息之一是数据类型。数据类型是特定语言的类型, 如string、integer、long或boolean。要产生代码, 首先要设置属性数据类型。

设置属性数据类型时, 可以用内置数据类型(string, integer, long等等), 也可以用Rose模型中定义类名。要在下拉列表框中显示定义的类, 选择Show Classes框。

要设置属性数据类型:

1. 右单击浏览器中的属性。
2. 从弹出菜单中选择Open Specification。Rose打开类属性规范窗口。
3. 从Type下拉列表框中选择数据类型或输入新的数据类型。

或

1. 选择Class框图中的属性。
2. 在属性名后面输入冒号和数据类型。例如, 要把属性Address设置为字符串, 输入Address:String。

设置属性版型

和角色、使用案例与类一样, 属性也有版型。属性版型可以将属性进行分类。例如, 有些属性映射数据库中的字段, 有些则不映射。为此可以定义两种版型。

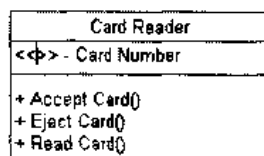
在Rose中, 不需要对属性指定版型。没有属性版型也能生成代码。但版型可以提高模型的可读性和可理解性。

要设置属性版型:

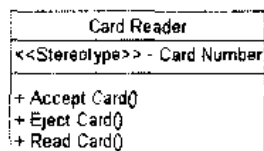
1. 右单击浏览器中的属性。
2. 从弹出菜单选择Open Specification。Rose打开类属性规范窗口。
3. 从下拉列表框图选择版型或输入新版型。

或

1. 选择浏览器中的属性。
2. 单击属性名进行编辑。名称前面出现<<>>:



3. 在<<>>中输入版型。



设置属性初始值

许多属性有相关的缺省值。例如，**Order**类保存公司购买订单的信息和行为。**Order**类的属性**TaxRate**表示购买时的销售税率。在你的城市，税率为7.5%，因此你的大多数订单用税率2.5%。可以将0.075指定为属性**TaxRate**的缺省值。

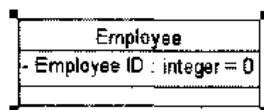
和属性版型一样，初始值也不是生成代码所必须的。但如果设置属性初始值，则Rose产生初始化属性的代码。

要设置属性初始值：

1. 右单击浏览器中的属性。
2. 从弹出菜单中选择**Open Specification**。Rose打开类属性规范窗口。
3. 在**Initial Value**字段中，设置属性初始值。

或

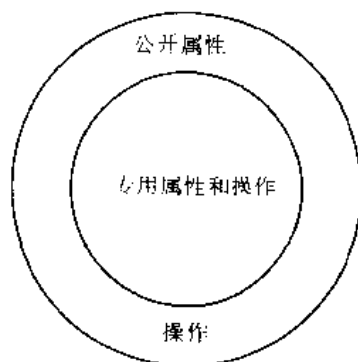
1. 选择Class框图中的属性。
2. 在属性数据类型后面，输入等于号和缺省值。缺省情况下，如果整数属性**EmployeeID**要设置属性初始值为0，则Class框图如下：



设置属性可见性

面向对象编程的核心概念之一是包装。每个类用属性和操作包装一些信息和行为。这种方法的一个好处就是能够生成小块自我包含代码。例如，**attribute visibility**类具有与员工有关的所有信息和行为。

可以把类看成如下：



属性包含在类中，其他类看不到。由于属性包含在类中，就要定义哪个类能浏览和改变这个属性，称为属性可见性（attribute visibility）。

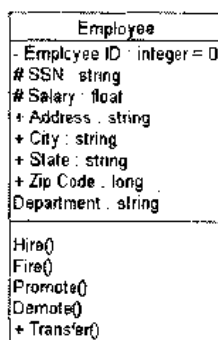
属性可见性选项有四个，下面一一举例。假设有个Employee类（带Address属性）和Company类。

Public 表示这个属性可以在所有其他类中访问。任何其他类都可以浏览或修改这个属性的值。这里，Company类可以浏览或修改Employee类的Address属性值，公开属性的UML图注为加号。

Private 表示这个属性不能在其他类中访问。Employee类知道Address属性的值，并可修改这个值，但Company类无法浏览或修改Address属性。如果Company类要浏览或修改Address属性，则请求Employee类浏览或修改Address属性。这通常通过公开操作进行。本章操作部分将介绍这个问题。专用属性的UML图注为减号。

Protected 类及其后代可以访问该属性。本例中，假设有两种员工，计时工和固定工资工。HourlyEmp和SalariedEmp继承Employee类。如果Address属性为保护可见性，则可以由Employee、HourlyEmp和SalariedEmp类浏览和修改，但Company类不能浏览和修改。保护属性的UML图注为英镑号。

Package或Implementation 表示属性公开，但只对同一包中的类公开。本例中，假设Employee类的Address属性为包可见性。则只有Company类与Employee类在同一包中时，Company类才能浏览和修改Address属性。实现可见性属性没有图标。



一般来说，属性的可见性建议用专用和保护。这些选项以更好地控制代码和属性。利用专用和保护属性，就可以避免系统中所有类都能修改属性，而把修改属性的逻辑包装在一个类中，和属性放在一起。选择的可见性选项影响生成的代码。例如，图6.3是上述类生成的Java代码。

Rose支持两种可见性符号。第一个是UML符号（+、-、#），分别表示公开、专用和保护属性。第二个是Rose图标，见表6.1。

表6.1 Rose可见性图标

图标	说明
	Public（公开）
	Private（专用）
	Protected（保护）
	Package or Implementation（包或实现）

```
// Source file: Employee.java

public class Employee {
    private integer Employee_ID = 0;
    protected string SSN;
    protected Float Salary;
    public string Address;
    public string City;
    public string State;
    public long Zip_Code;
    string Department;

    Employee() {
    }
    /**
     *Groseuid 36A95C448186
     */
    integer Hire() {
    }

    /**
     *Groseuid 36A95C5C8118
     */
    void Fire() {
    }

    /**
     *Groseuid 36A95C61889C
     */
    void Promote() {
    }

    /**
     *Groseuid 36A95C63828C
     */
    void Demote() {
    }

    /**
     *Groseuid 36A95C66889C
     */
    public void Transfer() {
    }
}
```

图6.3 可见性选项影响生成的代码

在Class框图中，可以用这些图注。下面说明介绍如何在这些图注间切换。图6.4是使用UML可见性图标例子。图6.5是使用Rose可见性图标的例子。表6.2总结了Rose和UML可见性图标。

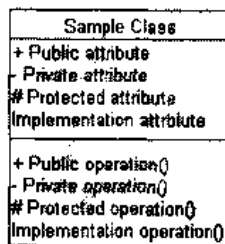


图6.4 UML可见性图标

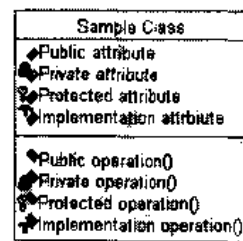


图6.5 Rose可见性图标

表6.2 Rose和UML可见性图标

需要	使用	UML图注	Rose图注
让属性向所有类显示	Public可见性	+	◆
让属性只向一个类显示	Private可见性	-	◆
让属性向一个类及其后代显示	Protected可见性	#	🔑
让属性向同一包中的所有类显示	Package或implementation可见性	<无图标>	◇

要设置属性可见性：

1. 右单击浏览器中的属性。
2. 从弹出菜单选择Open Specification，出现类属性规范窗口。
3. 在Export Control字段中选择属性可见性：Public、Protected、Private或Implementation，缺省为Private。

或

1. 选择Class框图的属性。
2. 如果使用UML可见性图标，单击属性旁边的+、-、#，选择清单中出现的Rose可见性图标选项。
3. 如果使用Rose可见性图标，单击属性旁边的Rose可见性图标，从出现的清单中选择可见性选项。



要改变可见性图标：

1. 选择Tools►Options。
2. 选择Notation标签。
3. 选择Visibility as Icons框使用Rose图注，或取消Visibility as Icons框使用UML图注。

说明：这个选项的改变影响新框图中使用的图注，现有框图不变。

设置属性包容

属性包容描述属性如何存放在类中。有三个包容选项：

By value（按数值） 属性放在类中，例如，如果属性类型为字符串，则字符串放在类定义中。

By reference（按引用） 属性放在类外，类指向这个属性。例如，如果Timecard类中有个Employee类型的属性，Timecard对象本身在Employee之外，Employee中的属性只是这个外部对象的指针。

Unspecified（未指定） 还没有指定包容类型。生成代码时，Rose假设为按数值包容。

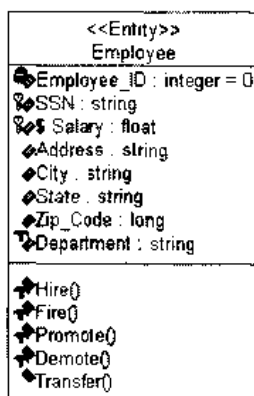
要设置属性包容选项：

1. 右单击浏览器中的属性。
2. 从弹出菜单选择Open Specification，出现类属性规范窗口（98i中为Open Standard Specification）。
3. 选择Detail标签。
4. 设置属性包容选项：By Value、By Reference或Unspecified。缺省情况下，所有属性均为Unspecified包容。

让属性为静态

属性加进类中时，类的每个实例收到自己的属性备份。例如，对于Employee类，运行时可能实例化三名员工：John Doe、Bill Jones和Jane Smith，每个对象有自己的Salary属性。

静态属性是类的所有实例共享的属性。回到上例，假设Salary属性是静态属性，则John Doe、Bill Jones和Jane Smith共享这个属性。UML静态属性名前面加上\$号。例如，Salary属性为\$Salary。



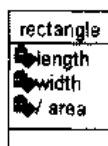
要让属性为静态：

1. 右单击浏览器中的属性。
2. 从弹出菜单选择Open Specification，出现类属性规范窗口（98i中为Open Standard Specification）。
3. 选择Detail标签。
4. 复选Static框让属性为静态，Rose在Class框图中静态属性名前面加上\$号。

指定派生属性

派生属性（derived attribute）是从一个或几个属性生成的属性。例如，Rectangle类可能有属性Width和Height，还有个Area属性是通过属性Width和Height计算出来的。由于Area是从属性Width和Height推导出来的，因此是个派生属性。

在UML中，派生属性的属性名前面加上/。上例中，Area写成/Area。



要指定派生属性：

1. 右单击浏览器中的属性。
2. 从弹出菜单选择Open Specification，出现类属性规范窗口（98i中为Open Standard Specification）。
3. 选择Detail标签。
4. 复选Derived框指定派生属性，Rose在Class框图中的属性名前面加上/。

使用操作

操作（operation）是与类相关联的行为。操作分三个部分：操作名、操作参数和操作返回值。参数是操作的输入变元。返回类型是操作的输出。

在Class框图中，可以显示操作名或操作名加操作参数和操作返回值类型。要简化Class框图，可以让一些Class框图只显示操作名，一些Class框图包括操作参数和操作返回值类型。

在UML中，操作用下列图注表示：

```
Operation Name (argument1:argument1 data type, argument2:argument2 data type, ...)  
:return type
```

操作定义类的责任。标识操作和研究类时，记注：

- 注意只有一个或两个操作的类，也许是合法的，但也许可以和其他类合并。
- 注意没有操作的类，类通常包装信息和行为，没有操作的类可以造型为一个或两个属性。
- 注意太多操作的类。一个类的责任应当限制在一定数量。如果操作太多，可能很难维护，最好分解成几个小类以便维护。

本节要介绍寻找操作，将其加进Rose模型和增加操作细节。还要介绍如何在Class框图中显示操作。

寻找操作

寻找操作很简单。生成Sequence和Collaboration框图时，就完成了寻找操作的主要工作。要考虑四种不同类型操作。

实现者操作

实现者操作（Implementor operations）实现一些业务功能。实现者操作可以从交互框图找到。Interaction框图主要关注业务功能，框图中的每个消息可能映射一个实现者操作。

每个实现者操作应能回溯要求，这是通过模型的各个部分实现的。每个操作来自Interaction框图中的消息，而这些消息又来自事件流的细节，事件流来自使用案例，而使用案例是来自要求的。这种回溯功能有助于保证每个要求均在代码中实现，每段代码均可回溯到要求。

管理者操作

管理者操作（Manager operations）管理对象的生成和构造。例如，类的构造器和删除器就属于管理者操作。

Rose中不需要对每个类手工生成构造器和删除器操作，生成代码时Rose可以提供自动生成构造器和删除器的选项。

访问操作

属性通常是专用或保护的。但其他类可能要浏览或改变某个类的属性，这可以通过访问操作（Access Operations）实现。

例如, 如果Employee类中有个Salary属性, 我们不希望所有类都能改变Salary属性, 而是在Employee类中增加两个访问操作GetSalary和SetSalary。GetSalary操作是公开的, 可以在另一个类中调用, 取得Salary属性值, 并返回调用类。SetSalary操作也是公开的, 可以在另一个类中调用, 设置Salary属性值。SetSalary中可以包含改变Salary属性值之前要检查的工资验证规则。

这种方法使属性安全地包装在类内, 防止其他类改变, 但同时允许有控制的属性访问。行业标准是对类中每个属性建立Get和Set操作。

和管理者操作一样, Rose中不需要对每个类手工生成访问操作, 生成代码时Rose可以提供自动生成访问操作的选项。

帮助器操作

帮助器操作(Helper operations)是类完成任务所需的操作, 其他类不需要知道这个操作。这是类的专用和保护操作。

和实现者操作一样, 帮助器操作可以从Sequence和Collaboration框图找到。通常, 帮助器操作显示为Sequence和Collaboration框图中的反身消息。

要标识操作, 可以完成下列步骤:

1. 检查Sequence和Collaboration框图。大多数消息是实现者操作, 而反身消息可能是帮助器操作。
2. 考虑管理者操作。可能要增加构造器和删除器, 这不是必需的, Rose产生代码时会自动产生。
3. 考虑访问操作。对需要被其他类浏览或改变的属性生成Get和Set操作。和管理者操作一样, 这不是必需的, Rose产生代码时会自动产生。

增加操作

和属性一样, 操作可以通过Class框图或通过浏览器加进Rose模型中。也可以通过类规范窗口增加类的操作。

一旦加进操作后, 可以将文档加进其中。加进操作的任何文档都会在生成的代码中显示为说明语句。操作文档通常包括操作目的、操作参数简要说明和操作返回类型等信息。

要将操作加进类中:

1. 右单击Class框图中的类。
2. 选择New►Operation。
3. 用下列格式输入操作名:

Name (Argument1:Argument1 data type) :Operation Return Type

例如:

```
Add(X : Integer, Y: Integer) : Integer
Print(EmployeeID : Long) : Boolean
Delete() : Long
```

4. 要增加更多操作, 按Enter并在Class框图中直接输入新操作。

或

1. 右单击浏览器中的类。
2. 选择New►Operation。
3. 浏览器的类下面出现新操作Opname。输入新操作名。Rose不允许在浏览器中输入操作变元和返回类型值。和属性一样，可以在Class框图中输入这些细节。

或

1. 打开类规范窗口（或98i中的标准规范窗口）。
2. 选择Operations标签，如果类已经有一些操作，则会在这里列出。
3. 右单击operations区内任一空白处，如图6.6。

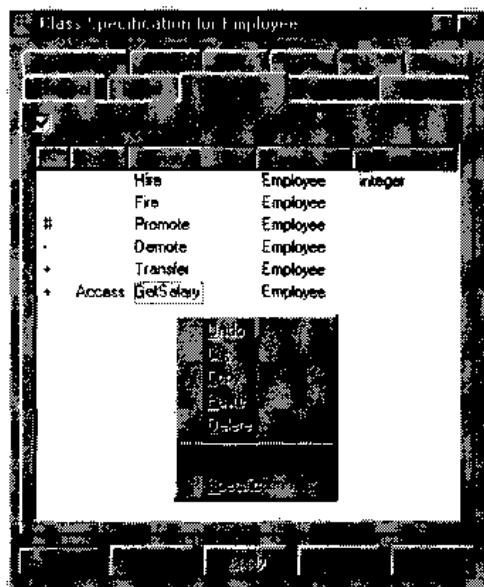


图6.6 通过类规范窗口增加类的操作

4. 从弹出菜单选择Insert。
5. 在Operation列中输入新操作名。
6. 输入可见性、版型和返回类型。

要将文档加进操作中：

1. 选择浏览器中的操作。
2. 在文档窗口输入操作文档。

或

1. 选择Class框图中的操作。
2. 在文档窗口输入操作文档。

或

1. 右单击浏览器中的操作。
2. 从弹出菜单选择Open Specification。
3. 在操作规范窗口的DocComment区输入操作文档。

或

1. 打开类规范窗口（或98i中的标准规范窗口）。
2. 选择Operations标签。
3. 选择操作。
4. 在文档窗口输入操作文档。

删除操作

如果要删除操作，可以在Class框图中或浏览器中进行。从框图中删除操作时，它自动从整个模型（包括其他框图）中删除。

删除操作时，一定要保持模型的一致性。Sequence和Collaboration框图中可能使用了这个操作。如果删除操作，则它自动变成所有框图中的消息。一定要相应更新Sequence和Collaboration框图。

要确定哪个框图引用一个操作：

1. 打开操作类的类规范窗口（或在98i中打开标准规范窗口）。
2. 选择对话框底部的Browse►Show Usage...

要从类中删除操作：

1. 右单击浏览器中的操作。
2. 从弹出菜单选择Delete。

或

1. 选择Class框图中的操作。
2. 用退格键删除框图中的操作名和签名。
3. 单击框图中任一位置。
4. Rose先确认删除，然后再删除操作。

或

1. 打开操作类的类规范窗口（或在98i中打开标准规范窗口）。
2. 选择Operations标签。
3. 右单击要删的操作。
4. 从弹出菜单选择Delete。
5. Rose先确认删除，然后再删除操作。

操作规范

在操作规范中，可以设置操作参数、返回类型、可见性等细节。下面几节介绍这些规范。所有规范均通过操作规范窗口浏览和改变，如图6.7。

要打开操作规范窗口：

1. 右单击浏览器中的操作。
2. 从弹出菜单选择Open Specification。

或

1. 打开操作类的类规范窗口。
2. 选择Operations标签。
3. 双击相应操作。

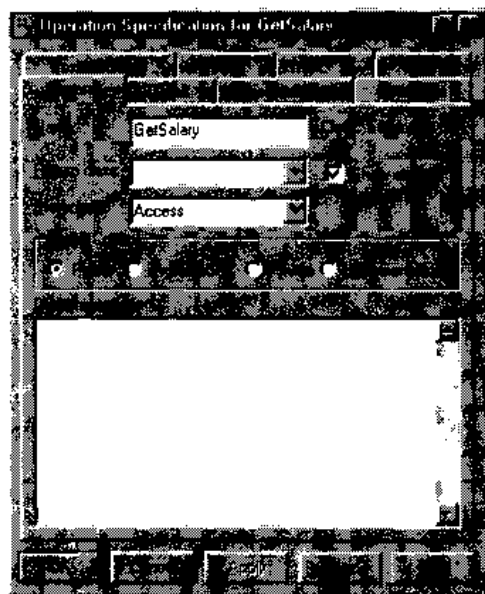


图6.7 操作规范窗口

说明：在Rose 98i中，上述过程打开不同的规范窗口。要打开图6.7所示的属性规范窗口，右单击浏览器中的属性并从弹出菜单中选择Open Standard Specification。

设置操作返回类

操作返回类是操作产生的数据类型。例如，假设操作Add取两个字符串X、Y为参数，将X、Y变为整数，然后相加，并将结果作为整数返回，则Add的操作返回类为整数。

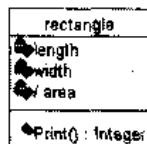
指定操作返回类时，可以用编程语言的内置数据类型（如string、char或integer）或Rose模型中定义的使用案例。

要设置操作返回类：

1. 右单击浏览器中的操作。
2. 从弹出菜单选择Open Specification（或98i中的Open Standard Specification）。
3. 从下拉列表框中选择返回类或输入新的操作返回类。

或

1. 选择Class框中的操作。
2. 在操作名后面，输入冒号和返回类型。例如，如果Print操作返回整数，则Class框图如下：



设置操作版型

和其他模型元素一样，操作可以用版型进行分类。前面曾介绍过，常见的操作版型有四种：

Implementor 实现某种业务逻辑的操作。

Manager 构造器、删除器和内存管理操作。

Access 让其他类浏览或编辑属性的操作，通常取名为Get<attribute name>和Set<attribute name>。

Helper 一个类的专用和保护操作，其他类无法访问。

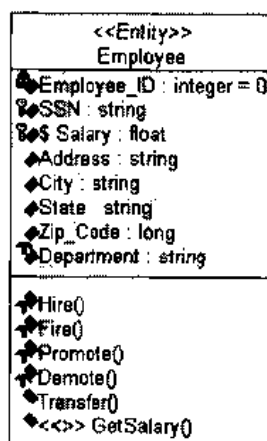
设置操作版型不是生成代码所必须的。但版型可以提高模型的可理解性，并可以避免遗漏操作。

要设置操作版型：

1. 右单击浏览器中的操作。
2. 从弹出菜单选择Open Specification（或98i中的Open Standard Specification）。Rose打开操作规范窗口。
3. 从下拉列表框选择版型或输入新版型。

或

1. 选择浏览器中的操作。
2. 再次单击操作名进行编辑。名称前面会出现<<>>。



3. 在<<>>中输入版型。

设置操作可见性

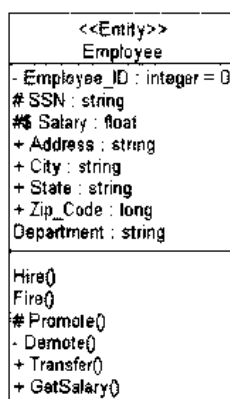
前面曾介绍过，可见性与类中包装信息和行为的方法有关。操作有四种可见性（详见表6.2）。

Public 操作可被所有其他类访问。任何其他类都可以请求执行该操作。

Private 操作不让任何其他类访问。

Protected 类及其后代可以访问该操作。

Package或Implementation 这个操作是公开的，但只对同一包中的类公开。



属性通常是专用或保护的，而操作则可以公开、专用、保护或包。作出决策时，要考虑其他哪些类（如有）要知道这个操作的存在。生成代码时，Rose会产生相应的可见性。例如，上述类生成的代码如图6.8。

```

public class Employee {
    private integer Employee_ID = 0;
    protected string SSN;
    protected static float Salary;
    public string Address;
    public string City;
    public string State;
    public long Zip_Code;
    string Department;

    Employee() {
    }
    /**
     *Roseuid 36A95C448186
     */
    integer Hire() {
    }

    /**
     *Roseuid 36A95C5C8118
     */
    void Fire() {
    }

    /**
     *Roseuid 36A95C61883C
     */
    protected void Promote() {
    }

    /**
     *Roseuid 36A95C6382BC
     */
    private void Demote() {
    }

    /**
     *Roseuid 36A95C66883C
     */
    public void Transfer() {
    }

    /**
     *Roseuid 36A96B628334
     */
    public void GetSalary() {
    }
}
  
```

图6.8 生成代码中的操作可见性

本章前面曾介绍过，Class框图中可以用UML或Rose图注。下面介绍如何在两种图注间变换。图6.4显示了使用UML可见性图注的类，图6.5使用Rose可见性图注。要回顾可见性选项，包括UML和Rose可见性图注，见表6.2。

要设置操作可见性：

1. 右单击浏览器中的操作。

2. 从弹出菜单选择Open Specification (或98i中的Open Standard Specification)。Rose打开操作规范窗口。
3. 从Export Control字段中, 选择操作可见性: Public、Protected、Private或Implementation。缺省情况下, 所有操作的操作可见性为Public。

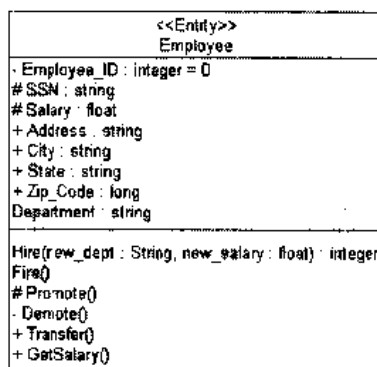
或

1. 选择Class框图中的操作。
2. 如果用UML可见性图注, 单击操作旁边的+、-或#。从出现的清单中选择可见性图标中选择可见性选项。
3. 如果用Rose可见性图注, 单击操作名旁边的可见性图标, 从出现的清单中选择可见性图标。

将变元加进操作中

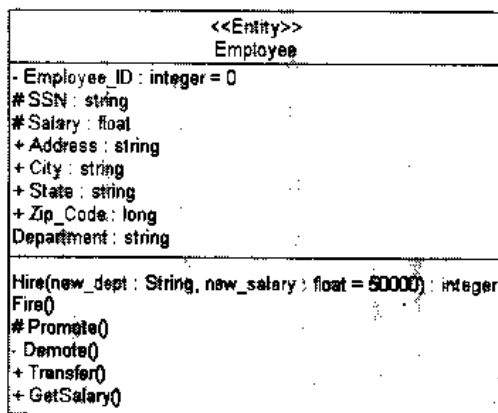
操作变元 (或参数) 是操作接收的输入数据。例如, Add操作可能取两个变元X、Y并将其相加。

每个变元要提供两个信息, 一个是变元名, 一个是数据类型。在Class框图中, 变元名和数据类型放在操作名后面的括号中。



如果喜欢, 还可以指定每个变元的缺省值。如果指定缺省值, 则UML记法如下:

Operation name (argument1:argument1 data type=argument1 default value) :operation return type



生成代码时，Rose生成操作名、变元、变元数据类型、变元缺省值和返回类型。Rose还对加进文档的操作生成说明语句。

要将变元加进操作：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 右单击变元框，并从弹出菜单选择Insert。
4. 输入变元名，如图6.9。
5. 单击数据类型列并输入变元数据类型。
6. 还可以单击缺省列，输入变元缺省值。

要删除操作的变元：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 右单击变元框中要删除的变元，并从弹出菜单选择Delete。
4. 确认删除。

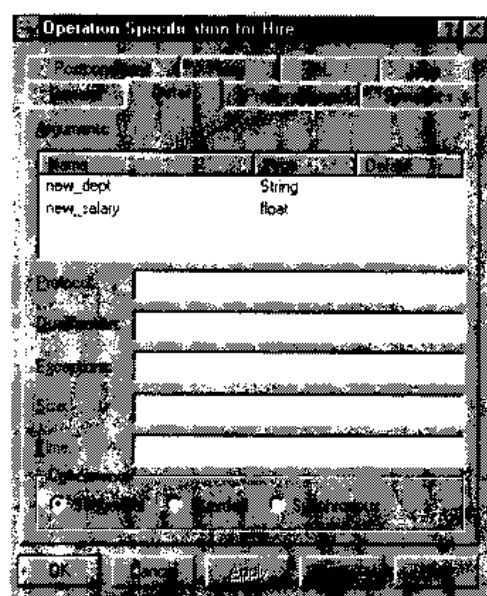


图6.9 操作变元

指定操作协议

操作协议描述客户可以对对象进行的操作，以及操作执行的顺序。例如，如果操作A要在执行操作B之后再执行，则可以在操作A的协议字段中标明。

这里输入的信息在生成代码时成为说明语句，但不会影响操作代码的生成。图6.10是操作协议屏幕。

要指定操作协议：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Protocol字段中输入协议。

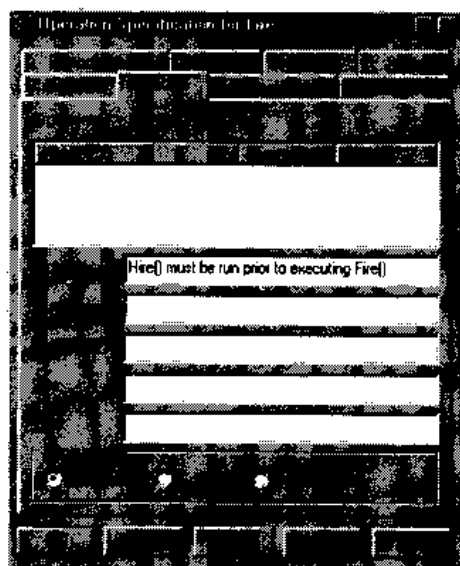


图6.10 操作协议屏幕

指定操作限定

这个字段指定操作的语言特定限定。这里输入的信息在生成代码时成为说明语句，但不会影响操作代码的生成。

要指定操作限定：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Qualification字段中指定操作限定。

指定操作异常

操作Exceptions字段可以列出操作可抛出的异常。和协议与限定一样，异常信息在生成代码时成为说明语句，并且会影响操作代码的生成。例如，图6.11是用异常信息生成的C++代码。

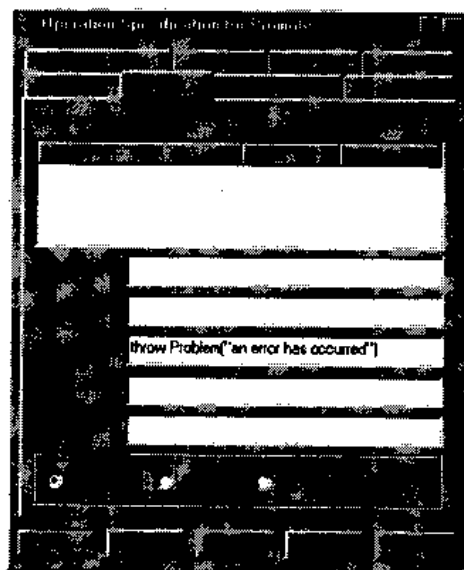


图6.11 代码中的操作异常

要指定操作异常：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Exceptions字段中指定操作异常。

指定操作长度

Size字段可以指定操作运行时所需的内存量。这个信息生成代码中的说明语句。

要指定操作长度：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Size字段中指定操作长度。

指定操作时间

操作时间是执行这个操作所需的大概时间量。这个信息生成代码中的说明语句。

要指定操作时间：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Time字段中指定操作时间。

指定操作并发性

Concurrency字段指定多控制线程中的操作行为。操作并发性有三个选项：

Sequential 是缺省设置，只有一个控制线程时，类正常工作（即如期操作），而在有多个控制线程时则不能保证。

Guarded 存在多个控制线程时，但不同线程中的类应相互协作，保证不会互相干扰。

Synchronous 存在多个控制线程时，类正常工作不需要与其他类相互协作，因为类本身能处理互斥情形。

操作并发性信息在生成代码时成为说明语句。

要指定操作并发性：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 从Concurrency字段中选择所要操作并发性。

指定操作前提条件

前提条件是运行操作之前要符合的条件。可以在操作规范窗口的Preconditions标签中指定操作前提条件，如图6.12。

前提条件不影响生成的操作代码，但会在生成代码时成为说明语句。如果Interaction框图演示操作前提条件，则可以在Preconditions标签底部输入Interaction框图名。

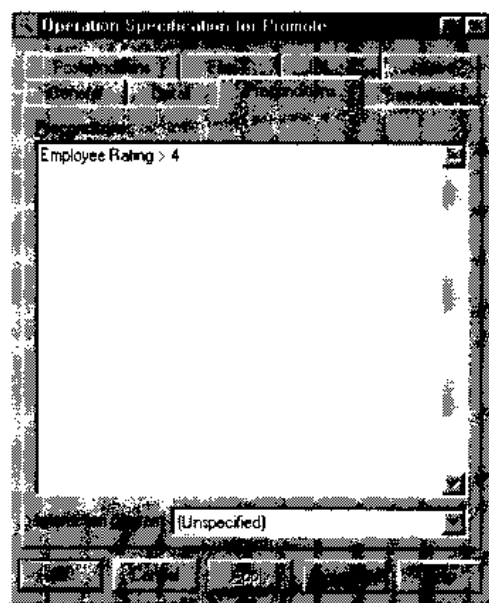


图6.12 指定操作前提条件

要指定操作前提条件:

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Preconditions字段中指定操作前提条件。

指定操作事后条件

事后条件是运行操作之后要符合的条件。可以在操作规范窗口的Postconditions标签中指定操作前提条件，如图6.13。

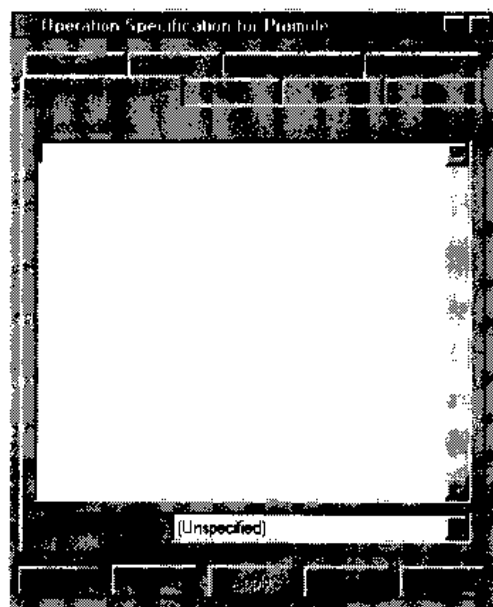


图6.13 指定操作事后条件

事后条件不影响生成的操作代码，但会在生成代码时成为说明语句。如果Interaction框图演示操作事后条件，则可以在Postconditions标签底部输入Interaction框图名。

要指定操作事后条件：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Postconditions字段中指定操作事后条件。

指定操作词法

操作规范窗口Semantics字段可以指定操作的工作，如图6.14所示。这里可以用伪代码，或用说明描述操作逻辑。Semantics字段中输入的内容会成为代码中的说明语句。如果Interaction框图与操作词法有关，可以在这个标签页面中输入框图名。

要指定操作词法：

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 选择Detail标签。
3. 在Semantics字段中指定操作词法。

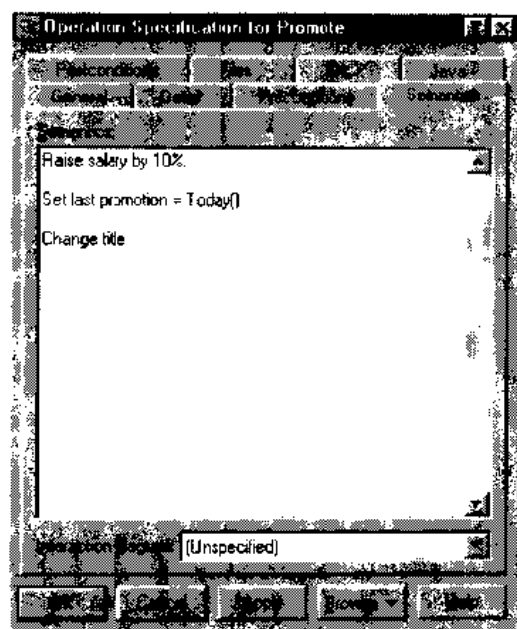


图6.14 指定操作词法

将文件与URL连接操作

外部文件或URL中也许包含一些与特定操作有关的信息。例如，要求文档中可能包含与特定操作有关的要求或描述操作的作用。

Rose模型中，可以将文件与URL连接操作。将文件与URL连接操作之后，可以直接从浏览器中打开连接的文件和URL。

要将文件连接操作：

1. 打开操作规范窗口（或98i中的标准规范窗口）。

2. 右单击Files标签中任一空白位置。
3. 从弹出菜单选择Insert File插入文件。
4. 用Open对话框寻找要连接的文件。
5. 选择Open将文件连接操作。

或

1. 右单击浏览器窗口的操作。
2. 选择New►File。
3. 从弹出菜单选择Insert File插入文件。
4. 用Open对话框寻找要连接的文件。

要将URL连接操作:

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 右单击Files标签中任一空白位置。
3. 选择Insert URL插入URL。
4. 输入要连接的URL名。

或

1. 右单击浏览器窗口的操作。
2. 选择New►URL。
3. 输入要连接的URL名。

要打开连接的文件:

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 双击Files标签中的文件名。Rose自动启动相应应用程序并装入文件。

或

1. 在浏览器的适当操作中找到文件。
2. 双击文件名。Rose自动启动相应应用程序并装入文件。

或

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 右单击Files标签中的文件。
3. 从弹出菜单选择Open File/URL。Rose自动启动相应应用程序并装入文件。

或

1. 在浏览器的适当操作中找到文件。
2. 右单击浏览器中的文件名。
3. 从弹出菜单中选择Open。Rose自动启动相应应用程序并装入文件。

要打开连接的URL:

1. 打开操作规范窗口（或98i中的标准规范窗口）。
2. 双击Files标签中的URL名。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 在浏览器的适当操作中找到URL。
2. 双击URL名。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 打开操作规范窗口（或98i中的标准规范窗口）。
 2. 右单击Files标签中的URL。
 3. 从弹出菜单选择Open File/URL。Rose自动启动Web浏览器应用程序并装入URL。
- 或
1. 在浏览器的适当操作中找到URL。
 2. 右单击浏览器中的URL名。
 3. 从弹出菜单中选择Open。Rose自动启动Web浏览器应用程序并装入URL。

在Class框图中显示属性和操作

UML非常灵活，允许在Class框图中显示所有细节，或者只显示所要的细节。Rose中可以将Class框图定制成：

- 显示所有属性和操作
- 隐藏属性
- 隐藏操作
- 显示所选属性和操作
- 显示操作签名或只显示操作名
- 显示或隐藏属性与操作可见性
- 显示或隐藏属性与操作版型

下面几节介绍每个选项。在典型项目中，有许多Class框图中。有些关注关系，显示较少属性和操作细节；有些关注类，根本不显示属性和操作；有些关注属性和操作，显示所有细节信息。在Rose中，可以将一个类放在多个Class框图中。然后可以用下列选项显示或隐藏属性与操作细节。

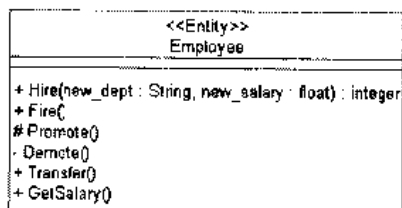
可以用Tools►Options窗口设置每个选项的缺省值。下面几节将介绍设置每个选项的缺省值的具体指令。

显示属性

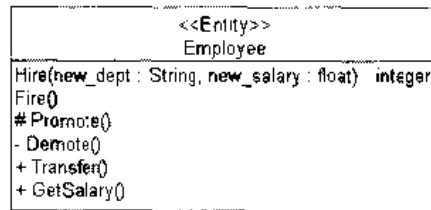
对Class框图中的某个类，可以：

- 显示所有属性
- 隐藏所有属性
- 显示所选属性
- 取消属性

取消属性不仅从框图隐藏属性，而且删除表示属性在类中的线。要演示隐藏与取消属性之间的差别，下面举一个例子。下面是隐藏属性的员工类：



下面是取消属性的员工类：



改变属性显示选项的方法有两种。可以分别访问每个类，设置相应选项；也可以先改变缺省属性显示选项再生成Class框图，改变缺省时，只影响新框图。

要显示类的所有属性：

1. 选择框图中所要的类。
2. 右单击类打开弹出菜单。
3. 选择Options►Show All Attributes。

或

1. 选择框图中所要的类。
2. 选择Edit►Diagram Object Properties►Show All Attributes。

要显示类的所选属性：

1. 选择框图中所要的类。
2. 右单击类打开弹出菜单。
3. 选择Options►Select Compartment Item。
4. 选择Edit Compartment窗口的所要属性。

或

1. 选择框图中所要的类。
2. 选择Edit►Compartment。
3. 选择Edit Compartment窗口的所要属性，如图6.15。

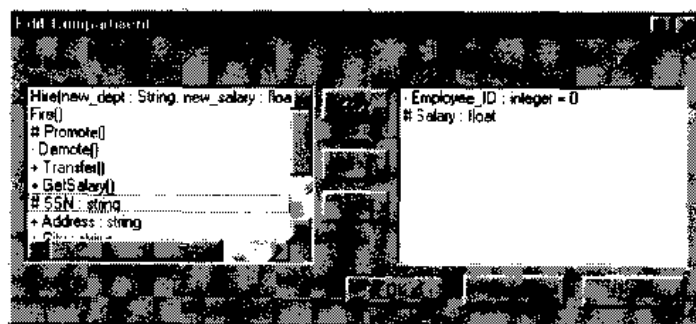


图6.15 选择Edit Compartment窗口的所要属性

要在框图上取消一个类的所有属性：

1. 选择所要类。
2. 右单击类打开弹出菜单。
3. 选择Options►Suppress Attributes。

或

1. 选择框图中所要的类。
2. 选择Edit►Diagram Object Properties►Suppress Attributes。
要改变显示属性的缺省选项：
 1. 选择Tools►Options。
 2. 选择Diagram标签。
 3. 用Suppress Attributes和Show All Attributes复选框设置缺省选项。

说明：改变显示属性的缺省选项时，只影响新框图，不影响现有Class框图。

显示操作

和属性一样，显示操作有几个选项：

- 显示所有操作
- 显示所选操作
- 隐藏所有选项
- 取消操作

此外，还有下列选项：

- 只显示操作名，即在Class框图中显示操作名，但不显示操作变元和返回类型。
- 显示完整操作签名，即不仅显示操作名，而且显示所有参数、操作数据类型和返回值。

要对类显示所有操作：

1. 选择框图中所要的类。
2. 右单击类打开弹出菜单。
3. 选择Options►Show All Operations。

或

1. 选择框图中所要的类。
2. 选择Edit►Diagram Object Properties►Show All Operations。

要显示类的所选操作：

1. 选择框图中所要的类。
2. 右单击类打开弹出菜单。
3. 选择Options►Select Compartment Item。
4. 选择Edit Compartment窗口的所要操作。

或

1. 选择框图中所要的类。
2. 选择Edit►Compartment。
3. 选择Edit Compartment窗口的所要操作，如图6.16。

要在框图上取消一个类的所有操作：

1. 选择所要类。
2. 右单击类打开弹出菜单。
3. 选择Options►Suppress Operations。

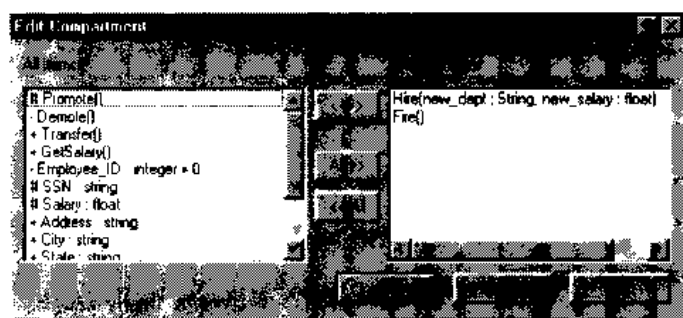


图6.16 选择Edit Compartment窗口的所要操作

或

1. 选择框图中所要的类。
2. 选择Edit>Diagram Object Properties>Suppress Operations。

要在Class框图中显示操作签名:

1. 选择框图中所要的类。
2. 右单击类打开弹出菜单。
3. 选择Options>Show Operation Signature。

或

1. 选择框图中所要的类。
2. 选择Edit>Diagram Object Properties>Show Operation Signature。

要改变显示操作的缺省选项:

1. 选择Tools>Options。
2. 选择Diagram标签。
3. 用Suppress Operations、Show All Operations和Show Operation Signatures复选框设置缺省选项。

说明: 改变显示操作的缺省选项时, 只影响新框图, 不影响现有Class框图。

显示可见性

属性和操作有四个可见性选项: public、private、protected和package。UML中用+、-、#表示公开、专用与保护, 实现没有图标。

与其用UML, 也可以用Rose图注表示属性与操作可见性。属性与操作可见性的Rose和UML图注如表6.3所示。

表6.3 Rose和UML图注

可见性	UML图注	Rose图注
Public (公开)	+	
Private (专用)	-	
Protected (保护)	#	
Package或Implementation (包或实现)	<无图标>	

可见性可以用Rose和UML图注，也可以隐藏可见性图标。

要显示类的属性与操作可见性：

1. 选择框图中所要的类。
2. 右单击类打开弹出菜单。
3. 选择Options►Show Visibility。

或

1. 选择框图中所要的类。
2. 选择Edit►Diagram Object Properties►Show Visibility。

要改变可见性显示选项：

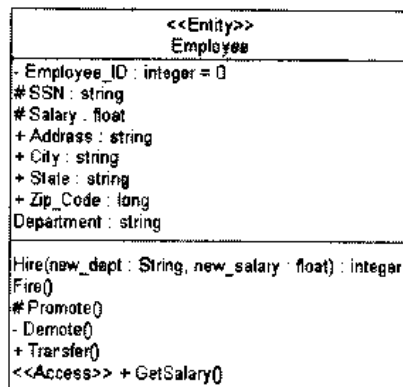
1. 选择Tools►Options。
2. 选择Diagram标签。
3. 用Show Visibility复选框设置缺省选项。

要在Rose和UML图注之间切换：

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 用Visibility as Icons复选框在Rose和UML图注之间切换。如果选择，则用Rose图注。如果取消，则用UML图注。改变缺省时，只影响新框图，不影响现有框图。

显示版型

在Rose中，可以显示或隐藏属性和操作的版型。如果显示版型，则它们出现在属性和操作名前面的<<>>中。



要显示类的属性和操作的版型：

1. 选择框图中所要类。
2. 右单击类打开弹出菜单。
3. 选择Options►Show Compartment Stereotypes。

或

1. 选择框图中所要类。
2. 选择Edit►Diagram Object Properties►Show Compartment Stereotypes。

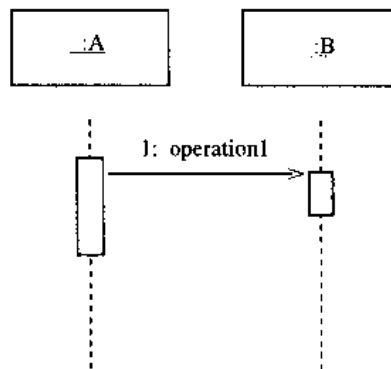
要改变缺省版型显示选项：

1. 选择Tools►Options。
2. 选择Diagram标签。
3. 用Show Stereotypes复选框设置缺省选项。

说明：改变缺省时，只影响新框图，不影响现有框图。

将操作映射消息

前面曾介绍过，Sequence或Collaboration框图中每个消息都映射操作。如果Sequence框图如下：



则操作Operation1放在类B中。首次生成Sequence或Collaboration框图时，消息名可能用英语短语，而不是操作名，如图6.17。

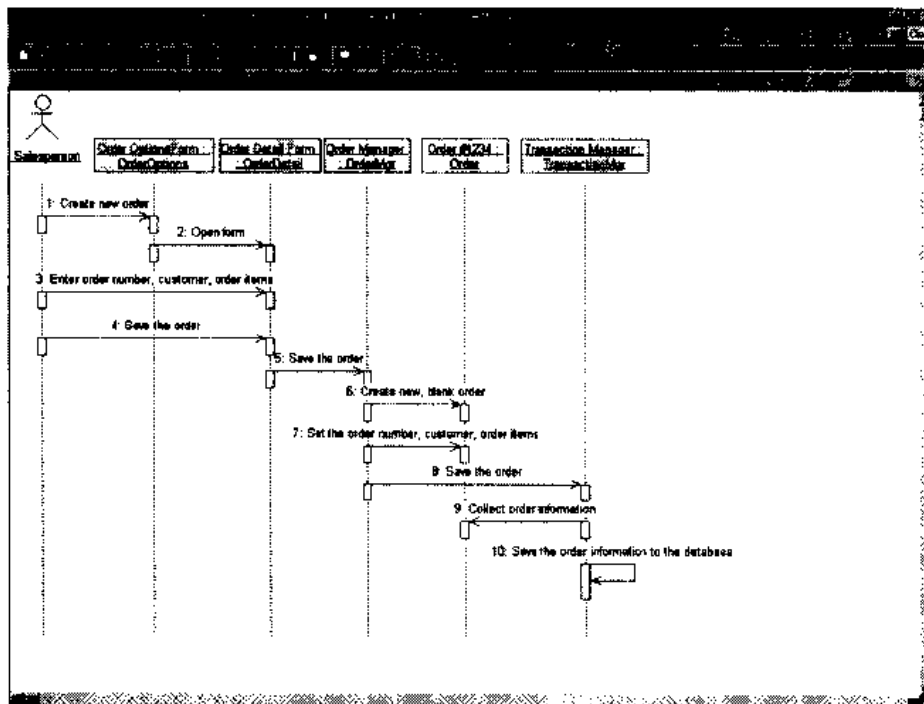


图6.17 没有操作映射的Sequence框图

但标识操作时，要将每个消息映射相应操作。Sequence框图变为图6.18。

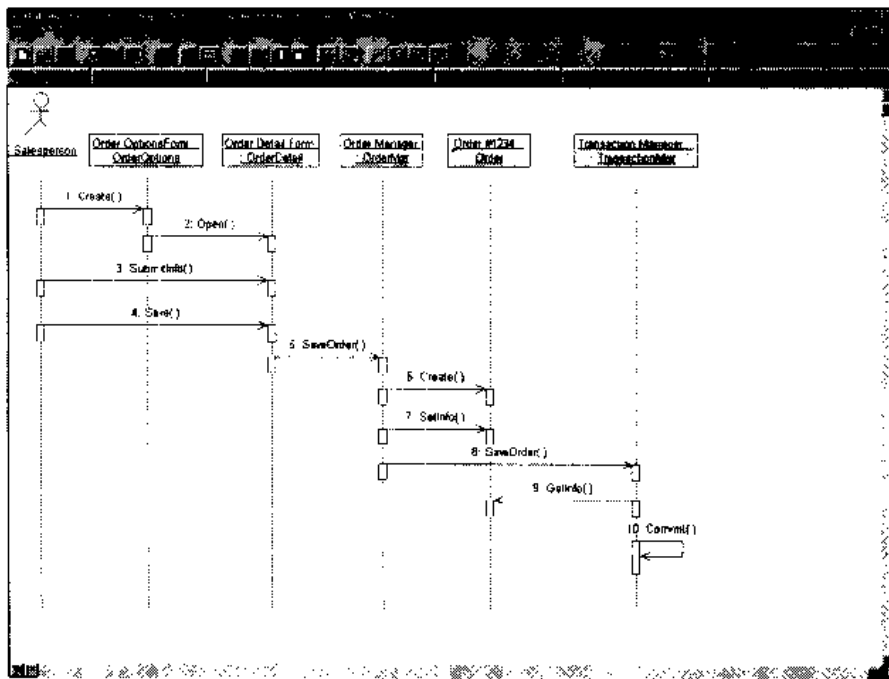


图6.18 有操作映射的Sequence框图

在Interaction框图中将操作映射消息

标识操作时，要检查Sequence或Collaboration框图中每个消息。生成代码前，要保证将每个消息映射相应操作。

说明：只有Interaction框图对象映射类时，才能将操作映射消息。

要将操作映射消息：

1. 确保接收对象（供应者）映射类。
2. 右单击Sequence或Collaboration框图中的消息。
3. 出现一系列供应者的操作，如图6.19。
4. 从清单中选择操作。

要删除消息的操作映射：

1. 双击Sequence或Collaboration框图中的消息。
2. 在Name字段中删除操作名，并输入新消息名。

要对消息生成新操作：

1. 确保接收对象（供应者）映射类。
2. 右单击Sequence或Collaboration框图中的消息。
3. 选择<new operation>。
4. 输入新操作名和细节。本章前面介绍了操作规范窗口中的选项。
5. 单击OK关闭操作规范窗口，增加新操作。
6. 右单击消息。
7. 从出现的清单中选择新操作。

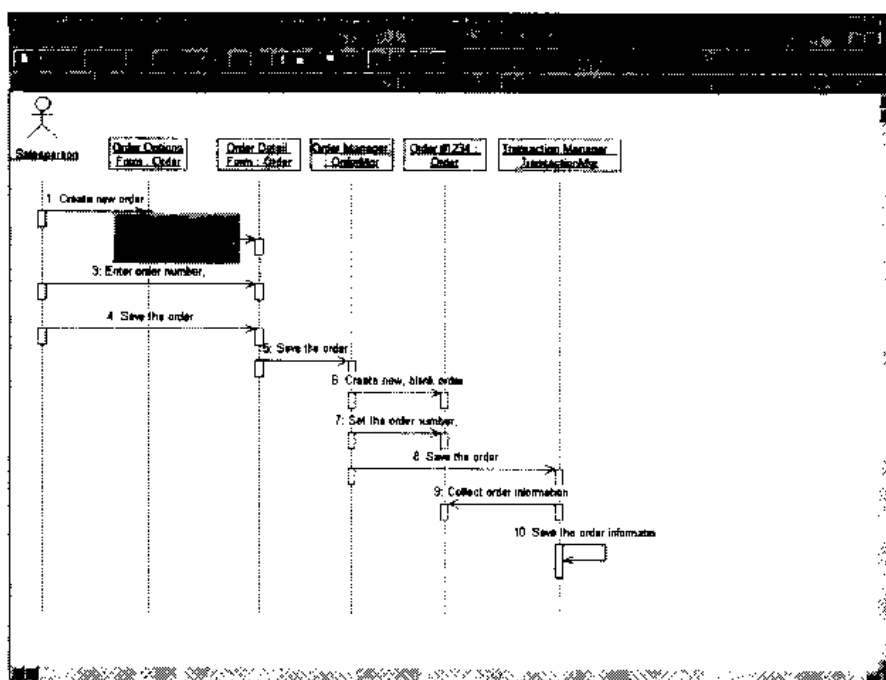


图6.19 将操作映射消息

要保证每个消息映射操作:

1. 选择Report►Show Unresolved Messages。
2. Rose显示所有未映射操作的消息, 如图6.20。

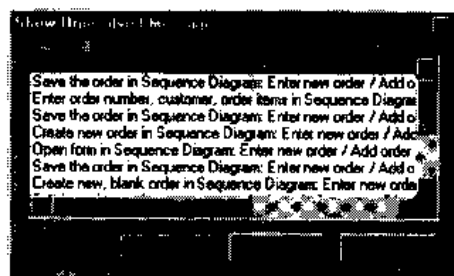


图6.20 未解析消息

练习

第4章练习生成了类的一些操作。上一练习中在Class框图上画出了类。本练习要增加操作细节, 包括参数与返回类型, 并增加属性。

问题

Karen有了Enter New Order使用案例中类的Class框图后, 开始填入细节。她选择C++编程语言, 然后在类中加进参数、数据类型和返回类型。

她还回到事件流中标识属性。将属性Order Number和Customer Name加进Class框图中的Order类中。她还检查订单项目。由于特定订单的订单项目很多。每个项目都有一些信息

和行为，因此她决定将订单项目建成类而不是Order的属性。

为了保持模型一致性，她更新Sequence框图，如图6.21。

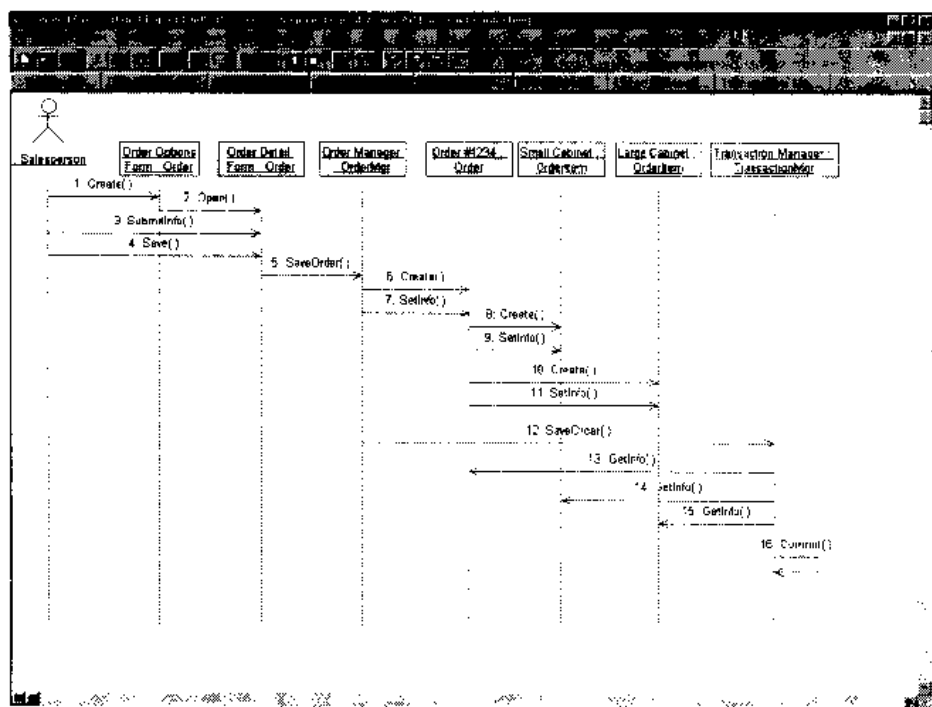


图6.21 更新Sequence框图

正在这时，Bob提出了要求改变。

“我们要开始跟踪订单日期和订单填写日期，还要补充一些新供应者，库存补货的程序也有新改变”。

Karen首先建档新的日期要求，并看看补货过程的改变。由于她目前正在处理Enter New Order使用案例，主要关心过程改变对这个使用案例的影响。她准备接下来处理Restock Inventory使用案例，到时再考虑补货过程的细节。新过程尽管对Restock Inventory使用案例影响很大，但并不影响Enter New Order使用案例。

新的日期要求需要在Order类中增加两个新属性。加进这些属性后，模型就能反映最新要求。

增加属性与操作

用Add New Order Class框图向类中增加属性与操作。增加属性与操作的特定语言细节。将选项设置成显示所有属性和所有操作，并显示操作签名。将选项设置成用UML图注显示可见性。

练习步骤

设置

1. 选择Tools►Options。
2. 选择Diagram标签。

3. 确保复选Show Visibility框。
4. 确保复选Show Stereotypes框。
5. 确保复选Show Operation Signatures框。
6. 确保复选Show All Attributes和Show All Operations框。
7. 确保取消Suppress Attributes和Suppress Operations框。
8. 选择Notation标签。
9. 确保取消Visibility as Icons框。

增加新类

1. 找到浏览器中的Add New Order Class框图。
2. 双击打开框图。
3. 选择Class工具栏按钮。
4. 单击框图中任一位置增中新类。
5. 将新类取名为OrderItem。
6. 将OrderItem类的版型设置为Entity。
7. 将浏览器中的OrderItem类拖动到Entities包中。

增加属性

1. 右单击Order类。
2. 从弹出菜单选择New Attribute。
3. 输入新属性。

OrderNumber : Integer

4. 按Enter。
5. 输入下一属性。

CustomerName : String

6. 重复4、5两步，加进下列属性：

OrderDate : Date

OrderFillDate : Date

7. 右单击OrderItem类。
8. 从弹出菜单选择New Attribute。
9. 输入新属性。

ItemID : Integer

10. 按Enter。
11. 输入下一属性。

ItemDescription : String

在OrderItem类中增加操作

1. 右单击OrderItem类。
2. 从弹出菜单选择New Operation。

3. 输入新操作。

Create

4. 按Enter。

5. 输入下一操作。

SetInfo

6. 按Enter。

7. 输入下一操作。

GetInfo

用Class框图增加操作细节

1. 单击选择Order类。
2. 再次单击Order类将光标移到类中。
3. 将Create()操作编辑如下：

Create() : Boolean

4. 将SetInfo()操作编辑如下：

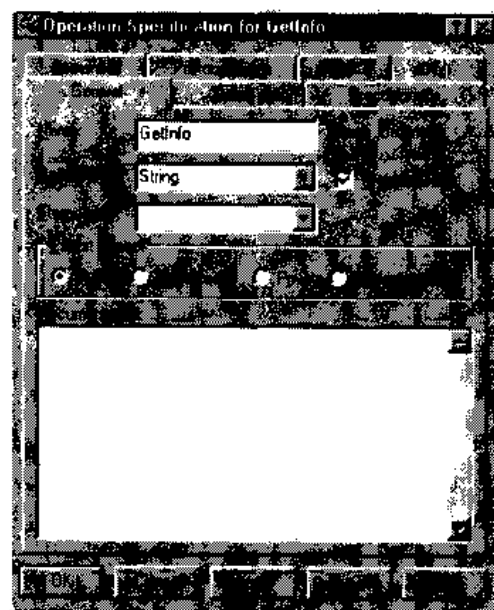
SetInfo (OrderNum : Integer, Customer : String, OrderDate : Date, FillDate : Date) : Boolean

5. 将GetInfo()操作编辑如下：

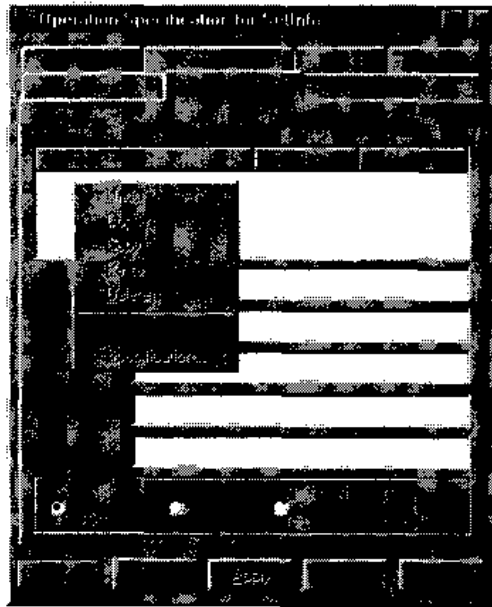
GetInfo() : String

用浏览器增加操作细节

1. 在浏览器中找到OrderItem类。
2. 单击OrderItem旁边的加号将其展开，即可在浏览器中显示属性和操作。
3. 双击GetInfo()操作展开Operation Specification窗口，如下图。



4. 在Return class下拉列表框中，选择String。
5. 单击OK关闭Operation Specification窗口。
6. 双击浏览器中OrderItem类的SetInfo()操作打开Operation Specification窗口。
7. 在Return class下拉列表框中，选择Boolean。
8. 选择Detail标签。
9. 右单击变元区空白处，加进参数，如下图。



10. 从弹出菜单选择Insert。Rose加进变元argname。
11. 单击选择argname字样，将变元名变为ID。
12. 单击Type列打开类型下拉列表框。从类型下拉列表框中选择Integer。
13. 单击Default列增加缺省值，在这个列中输入0。
14. 单击OK关闭Operation Specification窗口。
15. 双击OrderItem类的Create()操作打开Operation Specification窗口。
16. 在Return class下拉列表框中，选择Boolean。
17. 单击OK关闭Operation Specification窗口。

用两方法之一增加操作细节

1. 用浏览器或Class框图，在OrderDetail类中增加操作细节。操作签名如下：

```
Open() : Boolean
SubmitInfo() : Boolean
Save() : Boolean
```

2. 用浏览器或Class框图，在OrderOptions类中增加操作细节。操作签名如下：

```
Create() : Boolean
```

3. 用浏览器或Class框图，在OrderMgr类中增加操作细节。操作签名如下：

```
SaveOrder(OrderID : Integer) : Boolean
```

4. 用浏览器或Class框图，在TransactionMgr类中增加操作细节。操作签名如下：

```
SaveOrder(OrderID : Integer) : Boolean  
Commit() : Integer
```

小结

本章介绍了类的细节，包括属性与操作。我们介绍了增加属性、属性名、数据类型和缺省值；还介绍了操作，包括将其加进模型。我们详细介绍了操作变元、数据类型、返回值等细节。

前面介绍了单个类。下一章要介绍类间的关系。类间关系使应用程序完成其工作。

第7章 关 系

- 通过Class框图在Rose模型中增加关联、依赖、累积与一般化关系
- 增加关系名、版型、作用名、静态关系、朋友关系、限定符、链接元素和限制
- 设置倍增性、输出控制和包容

前面介绍了类、属性和操作。在Interaction框图中，我们介绍了类之间的通信。下面要介绍类之间的关系。

关系是类之间的词法连接，使一个类了解另一个类的属性、操作和关系。要让一个类向Sequence或Collaboration框图中另一个类发消息，两个类之间必须有关系。

本章介绍类之间和包之间可以建立的不同类型关系。我们介绍每个关系类型的含义以及如何把关系加进Rose模型中。

关系

本节介绍Rose模型中的四种关系，还介绍如何寻找关系。这里只是正式确定关系并加进Rose模型中。关系在Class框图中显示。

关系类型

类之间可以建立四种关系：关联、依赖性、累积和一般化。我们将一一介绍，这里先作一个简单介绍。

关联（Associations）是类之间的词法连接，在Class框图中用单线表示，如图7.1。

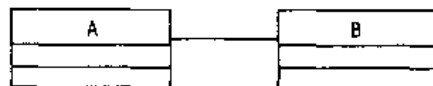


图7.1 关联关系

关联连接两个类时，如下例所示，每个类可以向Sequence或Collaboration框图中的其他类发消息。关联可以是双向的，也可以是单向的。在UML中，双向关联画成两端都有或都没有箭头，而单向关联则用箭头表示关联方向。

对于关联，Rose将属性放进类中。例如，如果House类与Person类之间有关联关系，则Rose将Person属性放进House类中，让房子知道谁是主人，并将House属性放进Person类中，让人知道拥有的房子。

依赖性（Dependencies）也是连接两个类，但与关联稍有不同。依赖性总是单向的，显示一个类依赖于另一个类的定义。Rose不对依赖性产生属性。回到上面的例子，如果Person

对House有依赖性而不是相关联, 则Rose不在House中产生Person属性, 也不在Person中产生House属性。但Person依赖于House的定义。依赖性用虚线箭头表示, 如图7.2。

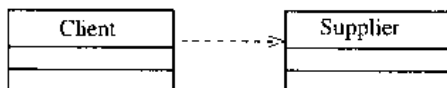


图7.2 依赖性关系

累积 (Aggregations) 是强关联。累积关系是整体与个体间的关系。例如, Car类和Engine、Tire以及其他汽车部件类之间, Car类是由一个Engine对象、四个Tire对象等组成的。累积关系在总体类旁边画一个菱形, 如图7.3。

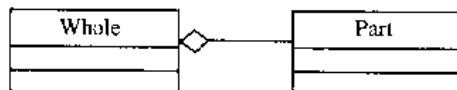


图7.3 累积关系

一般化 (Generalizations) 显示类之间的继承关系。大多数面向对象语言直接支持继承的概念。继承就是让一个类继承另一个类的所有属性、方法和关系。在UML中, 继承关系称为一般化, 显示为子类指向父类的箭头, 如图7.4。

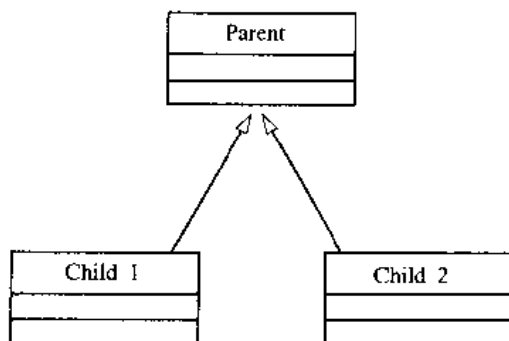


图7.4 一般化关系

寻找关系

要寻找关系, 可以检查前面建立的模型元素。大多数关系信息已经在Sequence或Collaboration框图中列出。现在可以重温这些框图, 取得关联和依赖性信息, 然后检查类中的累积和一般化信息。

要寻找关系, 做法如下:

1. 首先检查Sequence或Collaboration框图。如果类A向Sequence或Collaboration框图中的类B发出消息, 则它们必须有关系。通常, 用这个方法找出的关系是关联和依赖性。
2. 检查类的整体/部分关系。任何由其他类组成的类都参与累积。
3. 检查类的一般化关系。寻找具有不同类型的类。例如, 可能有个Employee类。你的公司有两种员工, 计时工和固定工资工。这表示有HourlyEmp和SalariedEmp类, 都继承Employee类。所有员工共同的属性、操作和关系放在Employee类中。计时工和固

定工资工特有的属性、操作和关系分别放在HourlyEmp和SalariedEmp类中。

4. 检查类中的其他一般化关系。试找出具有大量共同点的类。例如，两个类ChekingAccount和SavingsAccount有一些共同的信息和行为，可以放在第三个类Account中。生成Account，保存支票和存款帐号的共同属性、操作和关系。

关系太多的类要注意。设计良好的应用程序目标之一是减少系统中的关系。关系太多的类可能需要对许多其他类有所了解，因此很难复用，维护工作量也会很大。如果其他某个类改变，则原类会受到影响。

关联

关联是类之间的词法连接，使一个类知道另一类的公开属性和操作。例如，图7.5是House与Person间的双向关联。

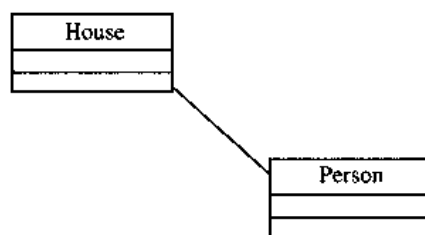
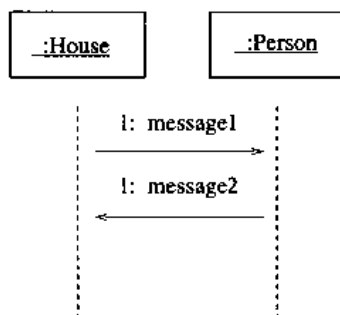


图7.5 关联关系

Person知道House的公开属性和操作，House知道Person的公开属性和操作。因此，在Sequence框图中，House可以向Person发消息，Person可以向House发消息。



上例中关系是双向的。但大多数关联应调整为单向的。单向关系更容易建立和维护，有助于寻找可复用的类。下面再看看上例，这时用单向关联，如图7.6。

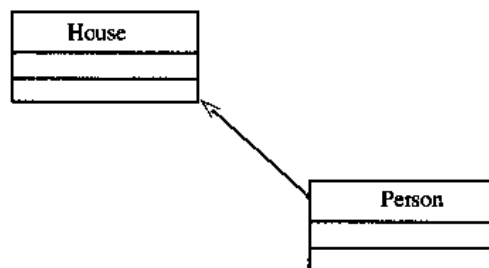


图7.6 单向关联

这里Person知道House的公开属性和操作，但House不知道Person的公开属性和操作，但House不知道Person的公开属性和操作。Sequence或Collaboration中Person可以向House发消息，但House不可以向Person发消息。

通过Sequence或Collaboration框图可以确定关联方向。如果Interaction框图中总是Person向House发消息，则是从Person到House的单向关系。如果又有从House到Person的关系，则需要双向关系。

单向关系有助于标识可复用的类。如果House与Person间关系是双向的，则每个类都需要知道对方，因此两者都不能复用。但假设是从Person到House的单向关系，则Person需要知道House，没有House就无法复用，而House不需要知道Person，因此House是可复用的。任何输出多个单向关系的类都很难复用，而只接收单向关系的类则很容易复用，如图7.7。

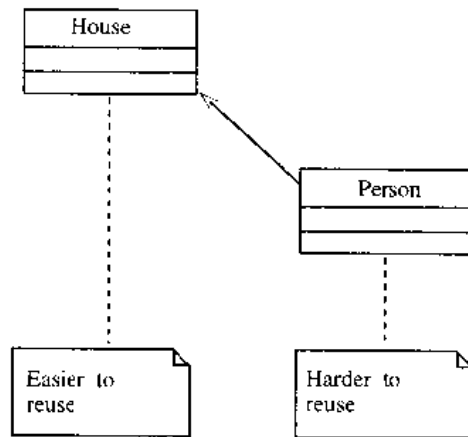


图7.7 单向关联与复用

生成双向关联的代码时，Rose在每个类中生成属性。在Person与House的例子中，Rose在House中放上Person属性，在Person中放上House属性。图7.8是对这两个类生成代码的例子。

```
// Source File: House.java
public class House {
    public Person m_Person;

    House() {
    }
}

// Source File: Person.java
public class Person {
    public House m_House;

    Person() {
    }
}
```

图7.8 生成双向关联的代码

反过来, 如果是单向关联, 则Rose在Person中放上House属性, 而不在House中放上Person属性。图7.9是单向关联的代码。

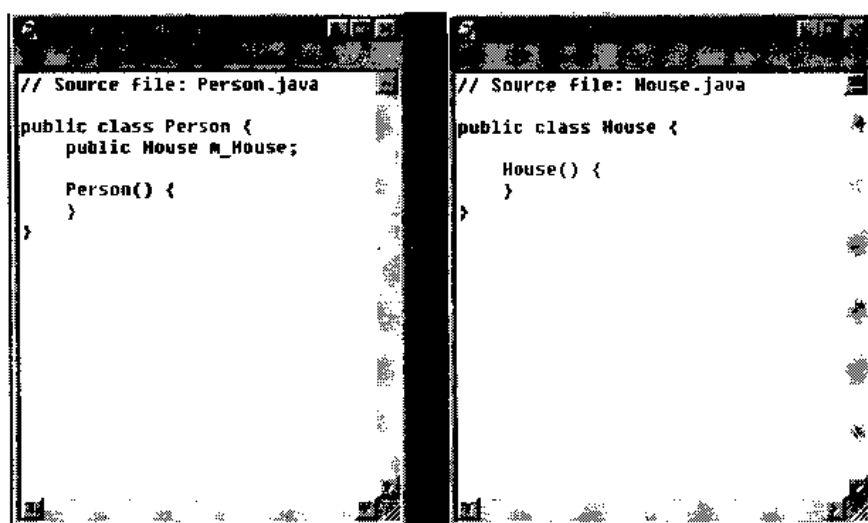


图7.9 生成单向关联的代码

关联也可以反身。反身关联让类的一个实例与同一个类的其他实例相联系。例如, Person类中一个人可能是另外几个人的父亲。由于这是Person的一个实例与其他实例相联系, 因此是反身关联, 如图7.10。

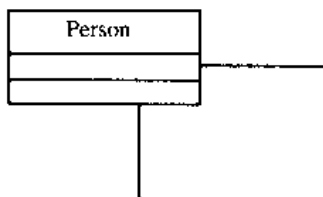


图7.10 反身关联

生成关联

Rose中可以直接在Class框图中生成关联。框图工具栏中有生成单向和双向关联的按钮。



如果生成双向关联, 则后面可以通过改变移动性将其变成单向关联。本章稍后“使用关系”中将介绍关联和移动性的细节。

要在Class框图中生成双向关联:

1. 选择工具栏中的关联图标。
2. 在一个类与另一个类之间拖动关联线。

或

1. 选择Tools>Create>Association。
2. 在一个类与另一个类之间拖动关联线。

要在Class框图中生成单向关联:

1. 选择工具栏中的单向关联 (Unidirectional Association) 图标。
2. 在一个类与另一个类之间拖动关联线。

或

1. 选择Tools►Create►Unidirectional Association。
2. 在一个类与另一个类之间拖动关联线。

要设置关系的移动性:

1. 在要移动的关系端右单击。
2. 从弹出菜单选择Navigable。

或

1. 打开所要关系的规范窗口。
2. 在要移动的关系端选择Role Detail标签。
3. 用Navigable复选框改变移动性。

要在Class框图中生成反身关联:

1. 选择工具栏中的关联图标。
2. 从类向类外拖动关联线。
3. 放开关联线。
4. 向类中拖动关联线。

或

1. 选择Tools►Create►Association。
2. 从类向类外拖动关联线。
3. 放开关联线。
4. 向类中拖动关联线。

要将文档加进关联:

1. 双击所要关联。
2. 选择General标签。
3. 在Documentation字段中输入文档。

或

1. 选择所要关联。
2. 选择Browse►Specification。
3. 选择General标签。
4. 在Documentation字段中输入文档。

要改变关联关系:

1. 选择所要关系。
2. 选择Edit►Change Into►Association。

删除关联

删除关联的方法有两种。第一种是从单个框图删除, 这时Rose知道关联还存在, 并在幕后跟踪。尽管可以从单个框图删除, 但其他框图中仍然存在。

第二种方法是从整个Rose模型中删除。这时是从所有框图中删除，Rose不再跟踪。

要从单个框图删除关联：

1. 选择所要关联。

2. 按Delete键。

或

1. 选择所要关联。

2. 选择Edit►Delete。

说明：从框图中删除关联并不从模型中删除。

要从模型中删除关联：

1. 选择所要关联。

2. 按Ctrl+D。

或

1. 选择所要关联。

2. 选择Edit►Delete from Model。

或

1. 打开参与关系的类的规范窗口。

2. 选择Relations标签。

3. 右单击关系。

4. 从弹出菜单选择Delete。

依赖性

依赖性关系显示一个类引用另一个类。因此，引用类规范改变可能影响使用类。两个类有依赖性关系时，Rose并不对关系的类增加属性。回到上例，假设Person和House有依赖性关系。依赖性关系用虚线箭头表示，如图7.11。

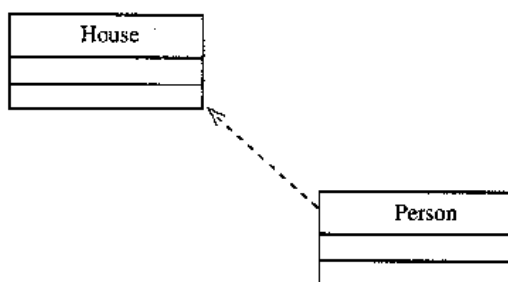


图7.11 依赖性关系

对这两个类生成代码时，并不对关系的类增加属性，如图7.12，但产生支持关系所需的特定语句。例如在C++中，生成代码中会包括必要的#include语句。

这里的含义是Person要设法知道House的存在，箭头方向表示Person依赖于House。换句话说，Sequence或Collaboration框图中Person向House发消息。如果是正常关联，则Person应有House属性。要向House发消息，Person只要查找自己的House属性。

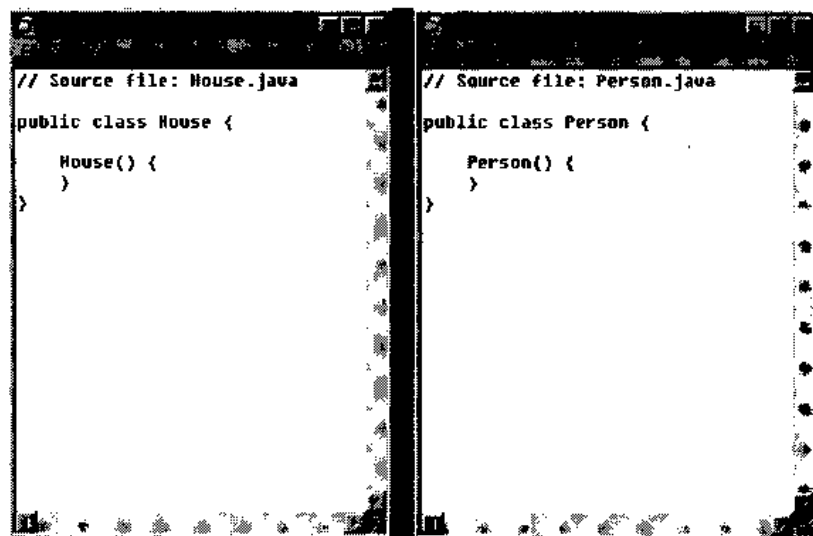
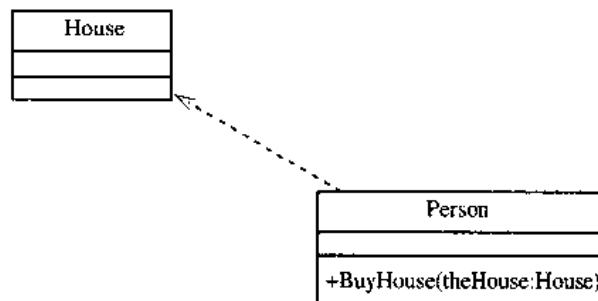


图7.12 依赖性关系的生成代码

但在依赖性关系中，**Person**没有**House**属性，因此要用其他方法查找**House**。有三种方法。如果**House**是全局的，则**Person**知道它存在。如果**House**实例化为**Person**操作中的本地变量，则**Person**知道它存在。最后，如果**House**作为参数传递到**Person**操作中，则**Person**知道它存在。在依赖性关系中，必须采用这三种方法之一。

尽管现在已经进入详细编码层，但这个决策会影响模型，我们可能需要在**Person**操作中加一个变元。



关联与依赖性的第二个差别在于方向。关联可以是双向的，而依赖性只能是单向的。依赖性也用包之间的关系，见本章稍后介绍。

生成依赖性关系

增加关联之后，可能要在具有依赖性时重新访问。这样，则可以用下列方法将关系从关联变为依赖性关系。

要生成新的依赖性关系，可以用Class框图工具栏的Dependency图标。



要在Class框图上生成依赖性关系：

1. 选择工具箱中的Dependency图标。

2. 单击依赖者的类。
3. 将依赖性关系线拖动到另一个类。

或

1. 选择Tools►Create►Dependency。
2. 单击依赖者的类。
3. 将依赖性关系线拖动到另一个类。

要将文档加进依赖性关系：

1. 双击所要依赖性关系。
2. 选择General标签。
3. 在Documentation字段中输入文档。

或

1. 选择所要依赖性关系。
2. 选择Browse►Specification。
3. 选择General标签。
4. 在Documentation字段中输入文档。

要改变Class框图中的关系为依赖性关系：

1. 选择所要关系。
2. 选择Edit►Change Into►Uses Dependency。

删除依赖性关系

和关联一样，有两种删除依赖性关系的方法。可以从一个Class框图删除，也可以从整个模型中删除。下面是删除依赖性关系过程：

1. 选择所要依赖性关系。
2. 按Delete键。

或

1. 选择所要依赖性关系。
2. 选择Edit►Delete。

说明：从框图中删除依赖性关系并不从模型中删除。

要从模型中删除依赖性关系：

1. 选择所要依赖性关系。
2. 按Ctrl+D。

或

1. 选择所要依赖性关系。
2. 选择Edit►Delete from Model。

或

1. 打开参与关系的类的规范窗口。
2. 选择Relations标签。
3. 右单击关系。
4. 从弹出菜单选择Delete。

包依赖性

包之间和类之间一样可以画出依赖性关系。事实上，依赖性关系只是包之间的关系之一。包依赖性关系和类依赖性关系一样用虚线箭头表示，如图7.13。

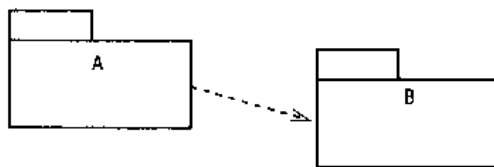


图7.13 包依赖性关系

从包A到包B的依赖性关系表示A中的某些类与B中的某些类有单向关系。换句话说，A中的某些类要知道B中的某些类。

这具有复用意义。如果依赖性关系如上，则A包依赖于B包。因此不能直接在另一应用程序中复用A包，而要同时复用B包。而B包则更容易复用，因为它没有依赖于其他包。

要确定包依赖性关系，就要检查Class框图中的关系。如果不同包中的类之间有关系，则包也有关系。

生成包依赖性关系时，要尽量避免循环依赖性。循环依赖性如图7.14。

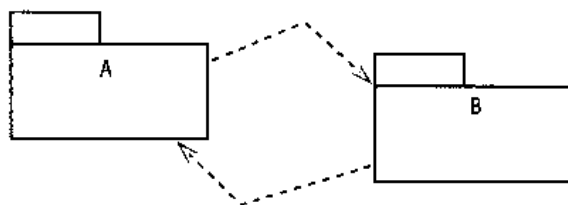
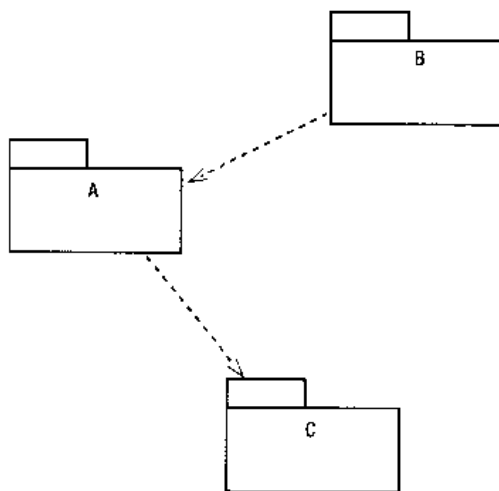


图7.14 循环依赖性

这里A中的某些类要知道B中的某些类，而B中的某些类又要知道A中的某些类。这样，两个包都不容易复用，一个包改变时会影响另一个包。这样就会失去包装类的独立性好处。要破坏循环依赖性，可以将一个包一分为二。本例中，可以将B包中A包所依赖的类移到C包中。这时包的依赖性关系如下：



生成包依赖性关系

确定包依赖性关系后，可以将其通过Class框图加进Rose模型中。通常，一个Class框图显示所有包及其相互关系。和类依赖性关系一样，用Dependency工具栏按钮画出关系。

要在Class框图中生成包依赖性关系：

1. 选择Dependency工具栏按钮。
2. 从依赖包向另一包画出依赖性关系线。

或

1. 选择Tools►Create►Dependency。
2. 从依赖包向另一包画出依赖性关系线。

删除包依赖性关系

和前面一样，有两种删除包依赖性关系的方法，从单个依赖性关系或从整个模型。

如果删除包依赖性关系之后两个包中的类仍有相互关系，则难以生成代码。可以用Report►Show Access Violations选项看看是否会有这种情况。

要在Class框图中删除包依赖性：

1. 选择所要包依赖性关系。
2. 按Delete键。

或

1. 选择所要包依赖性关系。
2. 选择Edit►Delete。

累积

累积是强关联。累积是整体与部分之间的关系。例如，EmployeeList可能是由Employees组成的。在UML中，累积显示为连接两个类的直线，整体端画一个菱形，如图7.15。

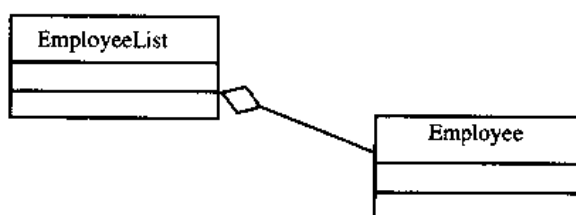
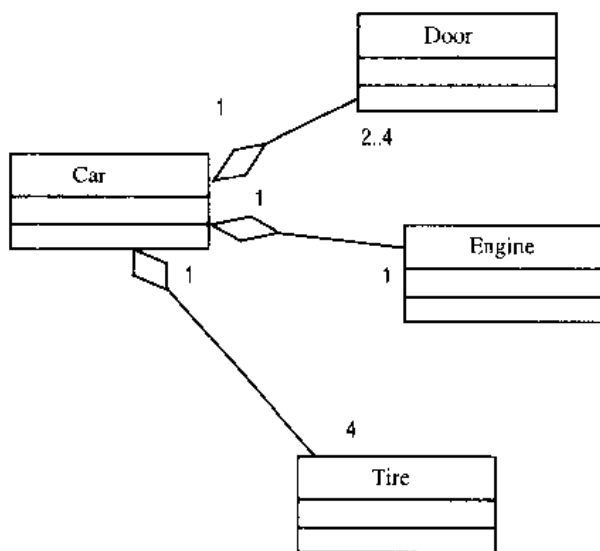


图7.15 累积关系

一个类可能与其他类有多个累积关系。例如，Car类与汽车部件的关系如下。



和关联一样，累积可以反身，如图7.16。反身累积中类的一个实例，由同一类的一个或几个其他实例构成。例如，烹调时，要组合一些调料，形成其他工序的调料。换句话说，调料可能由其他调料组成。

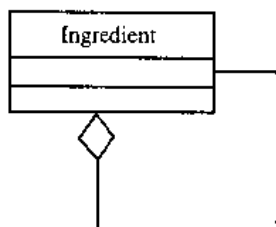


图7.16 反身累积关系

对累积关系生成代码时，Rose生成支持累积的属性。在Car例子中，Car类对累积关系中的门、发动机、轮胎和其他所有部件都生成属性。

生成累积关系

累积关系可以在Class框图中生成。要生成累积关系，可以用Class框图工具栏中的Aggregation按钮。



要在Class框图中生成累积关系：

1. 选择工具箱中的Aggregation图标。
2. 从类向父类拖动一条累积线。

或

1. 选择Tools►Create►Aggregation。
2. 从类向父类拖动一条累积线。

要在Class框图上生成反身累积关系：

1. 选择工具箱中的Aggregation图标。

2. 从类向类外拖动关联线。
3. 放开关联线。
4. 向类中拖动关联线。

或

1. 选择Tools►Create►Aggregation。
2. 从类向类外拖动关联线。
3. 放开关联线。
4. 向类中拖动关联线。

要将文档加进累积关系中:

1. 双击所要关联。
2. 选择General标签。
3. 在Documentation字段中输入文档。

或

1. 选择所要关联。
2. 选择Browse►Specification。
3. 选择General标签。
4. 在Documentation字段中输入文档。

要改变Class框图中的累积关系:

1. 选择所要关系。
2. 选择Edit►Change Into►Aggregation。

或

1. 打开所要关系的关系规范窗口。
2. 选择Role Detail标签。
3. 选择aggregate复选框。

删除累积关系

可以从单个Class框图或从整个模型中删除累积关系。下面介绍两者的过程。

要从框图删除累积关系:

1. 选择所要累积关系。
2. 按Delete键。

或

1. 选择所要累积关系。
2. 选择Edit►Delete。

说明: 从框图删除累积关系并不将其从模型中删除。

要从模型中删除累积关系:

1. 选择所要累积关系。
2. 按Ctrl+D。

或

1. 选择所要累积关系。
 2. 选择Edit►Delete from Model。
- 或
1. 打开参与关系的某个类的规范窗口。
 2. 选择Relations标签。
 3. 右单击关系。
 4. 从弹出菜单选择Delete。

一般化

一般化是两个类之间的继承关系。使一个类可以继承另一个类的公开和保护属性与操作。例如，可能有图7.17所示的关系。

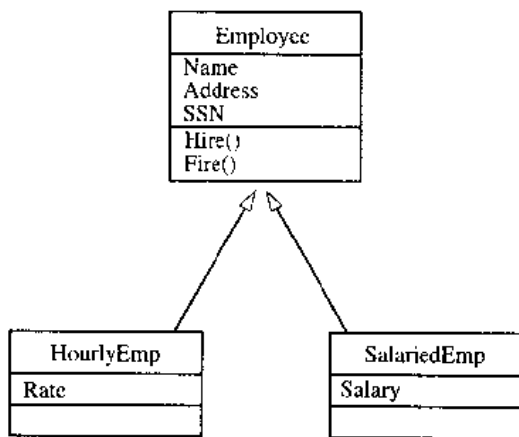


图7.17 一般化关系

本例中，有两种员工：定时工和固定工资工，两者都从Employee类继承。Employee类是上级类，HourlyEmp和SalariedEmp类是下级类。箭头从下级类指向上级类。

两个类共同的元素放在Employee类中。HourlyEmp和SalariedEmp类都继承Employee类的Name、Address和SSN属性和Employee类的Hire()和Fire()方法。

每个子类还可以有自己的独特属性、操作和方法。例如，只有计时工有小时工资，只有固定工资工有月工资。

一般化关系对开发和维护可以节省大量时间和精力。上例中，不必对HourlyEmp和SalariedEmp类分别编程和维护两个Hire()操作，而可以生成和维护一个操作。这个操作的任何改变均会在HourlyEmp和SalariedEmp类和Employee的其他子类中继承。

定义一般化关系时，可以从上向下或从下向上建立继承结构。要从上向下建立结构，可以看看具有不同类型的类。例如，可以从Employee类开始，认识到有不同类型的员工。要从下向上建立结构，寻找具有共性的类。这里从HourlyEmp和SalariedEmp类开始，认识到它们的相似性，然后生成Employee类，保存公共元素。

注意别生成不可维护的继承结构。继承层数太多时很难维护。向上层结构的每个改变

都会影响到下层的类。尽管这有好处，但这样会使分析和控制变得更重要。在某些语言中，层数太多会使应用程序减慢。

生成一般化关系

发现一般化关系后，可以将其用Class框图加进Rose模型中，通常，公司生成一个或两个针对继承结构的Class框图。这些Class框图通常只显示有限的属性和操作信息。

增加一般化关系时，可能需要移动一些属性和操作。例如，可以从HourlyEmp和SalariedEmp类开始，每个类都有Address属性。现在要将Address属性移到Employee类中。在浏览器中，可以将Address属性从HourlyEmp和SalariedEmp类拖动到Employee类。一定要记住从HourlyEmp和SalariedEmp类中删除Address属性。

要从Class框图生成一般化关系：

1. 选择工具箱中的Generalization图标。
2. 从子类向父类拖动一般化关系线。

或

1. 选择Tools►Create►Inherits。
2. 从子类向父类拖动一般化关系线。

要将文档加进一般化关系：

1. 双击所要一般化关系。
2. 选择General标签。
3. 在Documentation字段中输入Documentation。

或

1. 选择所要一般化关系。
2. 选择Browse►Specification。
3. 选择General标签。
4. 在Documentation字段中输入Documentation。

要将关系变成一般化关系：

1. 选择所要关系。
2. 选择Edit►Change Into►Inherits。

删除一般化关系

和其他关系一样，可以从单个框图或从模型中删除一般化关系。如果从整个模型删除一般化关系，则记住父类的属性、操作、和关系不再让子类继承。因此，如果子类需要这些属性、操作和关系，则要将这些属性、操作和关系加进子类中。

要从框图删除一般化关系：

1. 选择所要一般化关系。
2. 按Delete键。

或

1. 选择所要一般化关系。
2. 选择Edit►Delete。

说明：从框图删除一般化关系并不将其从模型中删除。

要从模型中删除一般化关系：

1. 选择所要一般化关系。

2. 按Ctrl+D。

或

1. 选择所要一般化关系。

2. 选择Edit►Delete from Model。

或

1. 打开参与关系的某个类的规范窗口。

2. 选择Relations标签。

3. 右单击关系。

4. 从弹出菜单选择Delete。

使用关系

本节要介绍Rose中关系的详细规范。在模型中，可以加进关联名、作用名、限定符等项目，指定关系存在的原因。

生成代码之前，要先指定关系倍增性，否则Rose会提供缺省。本节提供的大多数其他规范都是可选的。要生成代码时，如果没有设置所要规范，则Rose会指出这点。

设置倍增性

倍增性表示某个时刻一个类的几个实例与另一类的一个实例相联系。

例如，如果看看大学的课程注册系统，则可能有Course和Student类，两者之间有关系，学业有课程，课程有学生。问题是“一个学生同期选几门课程”，“一个课程同期有多少学生选”。

由于倍增性能回答这两个问题，因此倍增数指示符在关系两端都要用列。在课程注册例子中，我们决定每个学生可以注册0到4门课程，每个课程可以取10到20个学生。在Class框图中，表示为图7.18。

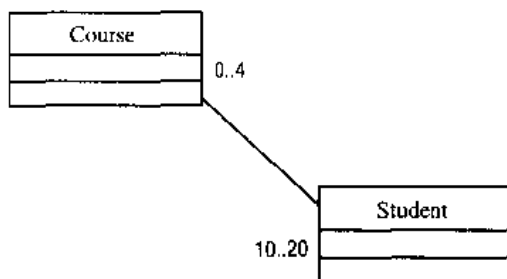


图7.18 关系倍增性

0..4表示每个学生可以注册0到4门课程，10..20表示每个课程可以取10到20个学生。倍增性的UML图注如下：

倍增性	含义
*	Many (多)
0	Zero (0)
1	One (1)
0..*	Zero or more (0或多)
1..*	One or more (1或多)
0..1	Zero or one (0或1)
1..1	Exactly one (1)

也可以用下列格式输入倍增性:

格式	含义
<number>	Exactly <number>
<number 1>..<number 2>	Between <number 1>到<number 2>
<number>..n	<number>或多
<number 1>,<number 2>	<number 1>或<number 2>
<number 1>, <number 2>..<number 3>	Exactly <number 1>或<number 2>到<number 3>
<number 1>..<number 2>,<number 3>..<number 4>	Between <number 1>到<number 2>或<number 3>到<number 4>

记住, 在倍增性设置中, 要指出关系是否可选的。在上例中, 每个学生可以注册0到4门课程, 因此, 停课的学生仍然是个学生。如果倍数为1..4, 则每个学生每学期至少要选1门课。因此, 倍数实现“每个学生每学期至少要选1门课”之类的业务规则。

通常, 窗体、屏幕或窗口之间的倍数在关系两端为0..1。尽管这并不总是正确, 但这个倍数表示每个窗体可以独立于其他窗体而存在。

要设置关系倍增性:

1. 右单击所要关系一端。
2. 从弹出菜单中选择**Multiplicity**。
3. 选择所要倍数。
4. 对关系另一端重复1~3步。

或

1. 打开所要关系的规范窗口。
2. 选择一端的Role Detail标签。
3. 用cardinality字段改变倍数。
4. 对关系另一端重复1~3步。

使用关系名

关系可以用关系名或角色名调整。关系名通常是个动词或动词短语, 描述关系的作用。例如, **Person**类和**Company**类可能有关联。但为什么要有这个关系呢? 这个人是公司客户、员工还是所有者? 我们可以将关系命名为**employs**, 表示关系的作用。图7.19显示了关系名的例子。

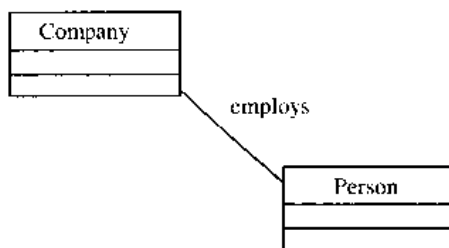


图7.19 关系名

关系名是可选的，通常只在关系的作用不明显时使用。关系名在关系线上显示。

在Rose中，还可以设置关系方向。上例中我们可以是公司雇人，但不能说人雇公司。在关系规范窗口中，可以设置关系名方向。

要设置关系名：

1. 选择所要关系。
2. 输入所要名称。

或

1. 打开所要关系的规范窗口。
2. 选择General标签。
3. 在name字段中输入名称。

要设置关系名方向：

1. 打开所要关系的规范窗口。
2. 选择Detail标签。
3. 用name direction字段设置关系名方向。

使用版型

和其他模型元素一样，可以指定关系版型。版型用于将关系分类。例如，可能有两种常用关联。可以对这些关联类型生成版型。版型在关联线上显示，放在<<>>中。要设置关系版型，可以用关系规范窗口的General标签，如图7.20。

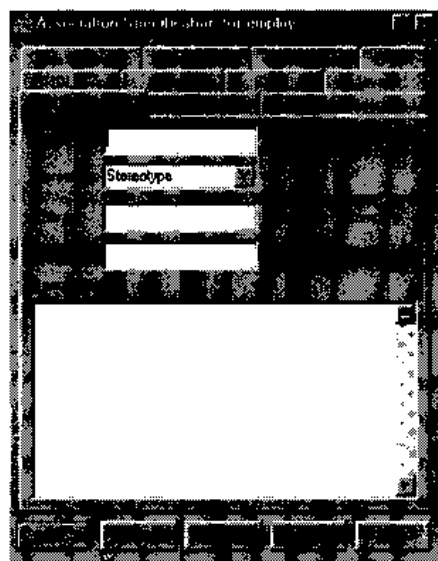


图7.20 关系规范窗口

要设置关系版型:

1. 打开所要关系的规范窗口。
2. 选择General标签。
3. 在stereotype字段中输入版型。

使用作用

作用名可以在关联或累积关系中代替关系名,描述关系的作用。回到Person和Company的例子,假设Person在Company中起员工的作用,作用名通常是名词或名词短语,显示在起这个作用的类旁边。通常,可以用关系名或作用名,但不能同时使用。和关系名一样,作用名是可选的,通常只在关系的作用不明显时使用。图7.21显示了作用的例子。

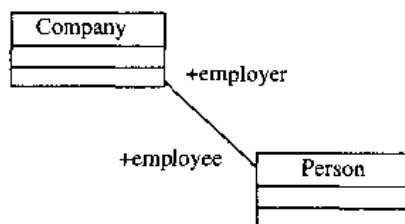


图7.21 关系中的作用

利用关系规范窗口可以将文档加进作用中。加进作用中的文档在生成代码时成为说明语句。要浏览框图中的作用,右单击关系并选择Role名,如图7.22。

要设置作用名:

1. 右单击所要关联的作用名端。
2. 从弹出菜单中选择Role name。
3. 输入作用名。

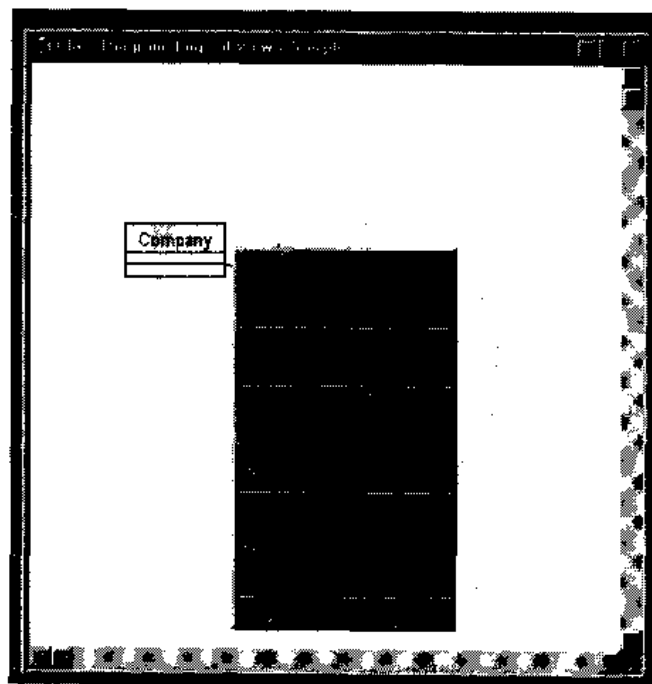


图7.22 设置作用文档

或

1. 打开所要关联的规范窗口。
2. 对要命名的作用选择Role General标签。
3. 在Role字段中输入作用名。

要增加作用文档：

1. 打开所要关联的规范窗口。
2. 对所要作用选择Role General标签。
3. 在Documentation字段中输入文档。

设置输出控制

在关联关系中，Rose在生成代码时生成属性。生成属性的可见性通过Export Control字段设置。和其他属性一样，有四个可见性选项：

Public 表示属性可以在所有其他类中显示。

Private 表示任何其他类中都无法显示该属性。

Protected 表示属性只在类及其子类中显示。

Package或Implementation 表示属性在同一包中所有其他类中显示。

在双向关系中，可以设置两个属性的输出控制，各在关系的一端生成。在单向关系中，只需设置一个输出控制。输出控制可以用关系规范窗口的Role A General和Role B General标签设置。

要设置输出控制：

1. 右单击所要作用名。
2. 从弹出菜单中选择Export Control。

或

1. 打开所要关系的规范窗口，如图7.23。
2. 对所要作用选择Role General标签。
3. 将Export Control设置为Public、Protected、Private或Implementation。

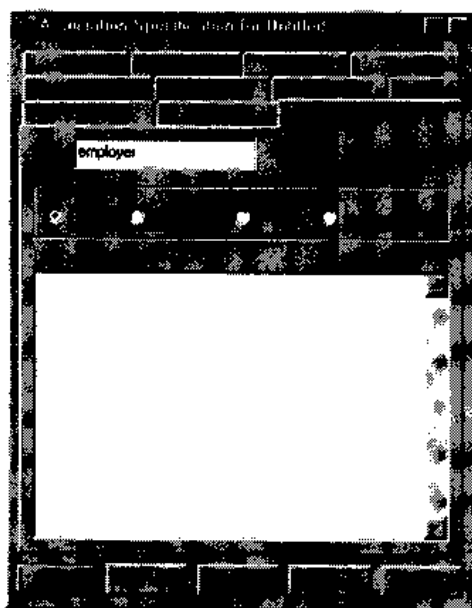


图7.23 设置输出控制

使用静态关系

前面曾介绍过，Rose对关联和累积关系产生属性。Static字段确定生成的属性是否静态的。静态属性是类的所有实例共享的属性。

如果将一个作用设置为静态的，则产生的关联属性为静态的。在Class框图中，静态作用前面显示\$，如图7.24。

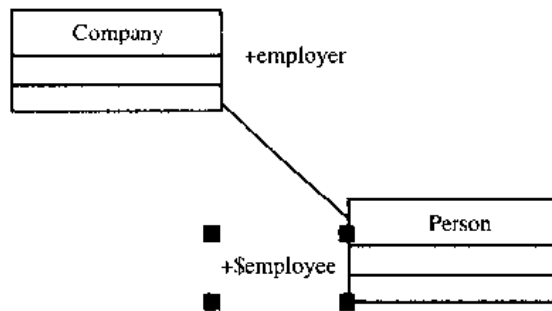


图7.24 静态作用

要将关联分类为静态：

1. 右单击要成为静态的关联端。
2. 从弹出菜单中选择**Static**。

或

1. 打开所要关联的规范窗口。
2. 对作为静态的作用选择**Detail**标签。
3. 选择**Static**复选框。

使用朋友关系

朋友关系表示Client类能访问Supplier类的非公开属性和操作。可以对关联、累积、依赖和一般化关系设置朋友属性。Supplier类的源代码包括让Client类具有朋友可见性的逻辑。

例如，假设从Person到Company类有单向关联，我们对这个关系设置Friend复选框。生成C++代码时，Company.H文件包括**friend class Person**语句，表示Person类能访问Company类的非公开属性和操作。

要将关系划分为朋友关系：

1. 右单击所要关系的相应端。
2. 从弹出菜单中选择**Friend**。

或

1. 打开所要关联的规范窗口。
2. 选择相应端的**Role Detail**标签。
3. 选择**Friend**复选框。

设置包容

Containment of Button字段确定、累积关系生成的属性按值或按引用包容。在累积关系中，整体类要加进表示每个部件类的属性。这里设置这些属性按值或按引用（by value或by reference）。

按值包容属性同时生成和部署整体及其部件。例如，如果窗口与按钮之间为按值累积，则窗口与按钮同时生成和部署。在UML中，按值累积显示为实心菱形，如图7.25。

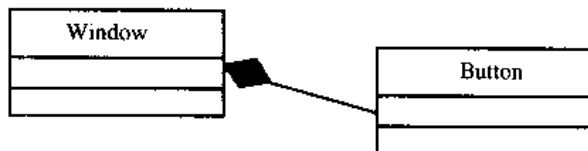


图7.25 按值累积

按引用累积的整体及其部件不是同时生成和部署。如果两者都在内存中，则通过累积相联系。EmployeeList和Employees不是同时生成和部署。按引用累积显示为空心菱形，如图7.26。

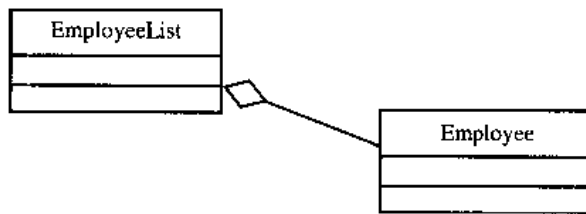


图7.26 按引用累积

要设置包容：

1. 右单击关系所要端以设置包容。
2. 从弹出菜单选择Containment。
3. 将包容设置为By Reference、By Value或Unspecified。

或

1. 打开所要关系的规范窗口。
2. 选择所要作用的Role Detail标签。
3. 选择包容为By Reference、By Value或Unspecified，如图7.27。

使用限定符

限定符用于缩小关联的范围。例如，我们可能有个Person和Company类之间的关联。假设对某个Person ID值，有两个相关的公司，则可以在Person类中加上限定符Person ID。框图如图7.28。

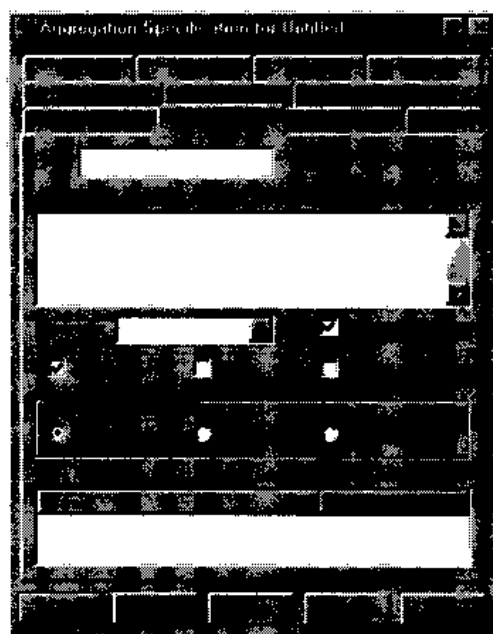


图7.27 设置包容

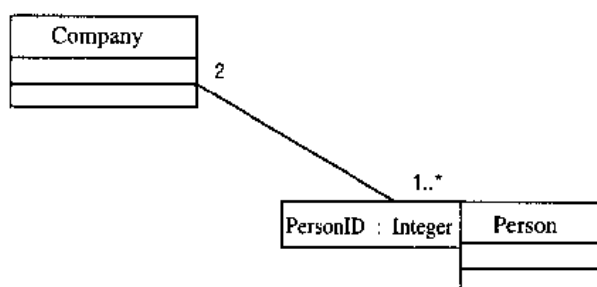


图7.28 使用限定符

要使用限定符：

1. 右单击所要关联中要增加限定符的端。
2. 从弹出菜单选择New Key/Qualifier。
3. 输入新的限定符数值和类型。

或

1. 打开所要关系的规范窗口。
2. 选择所要作用的Role Detail标签。
3. 右单击Role Detail框。
4. 从弹出菜单选择Insert。
5. 输入新的限定符数值和类。

要删除限定符：

1. 打开所要关系的规范窗口。
2. 选择所要作用的Role Detail标签。
3. 右单击要删除的限定符。
4. 从弹出菜单选择Delete。

使用链接元素

链接元素（link element）也称为Association类，可以放置与关联相关的属性。例如，两个类Student和Course，要把Grade属性放在哪里呢？如果放在Student内，则需要对学生注册的每门课程加一个属性到Student中，这样就会使Student类非常复杂。如果将属性加进Course类中，则需要对每个选该课的学生加一个属性。

要解决这个问题，我们生成一个Association类。属性Grade与学生与课程的链接有关，而不是与两者各自有关，因此可以放在这个新类中。关联中的UML图注如图7.29。

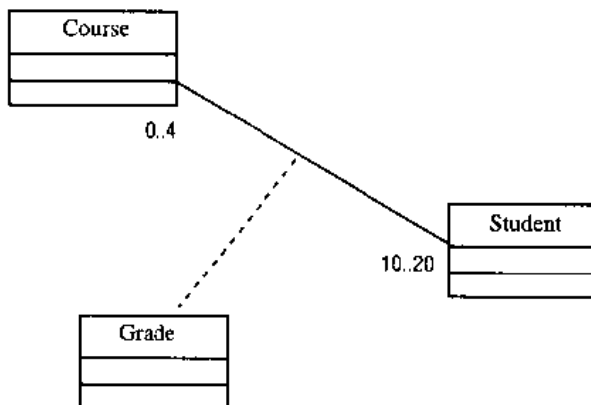


图7.29 链接元素（关联类）

要设置关系的链接元素：

1. 打开所要关系的规范窗口。
2. 选择Detail标签。
3. 用Link Element字段设置链接元素。

使用限制

限制是必须符合的条件。在Rose中，可以设置关系或单个作用的限制条件。输入的限制在生成代码时成为说明语句。

要设置关系限制：

1. 打开所要关系的规范窗口，如图7.30。
2. 选择Detail标签。
3. 在Constraints字段中输入限制。

要设置作用限制：

1. 打开所要关系的规范窗口。
2. 选择所要作用的Role Detail标签。
3. 在Constraints字段中输入限制，如图7.31。

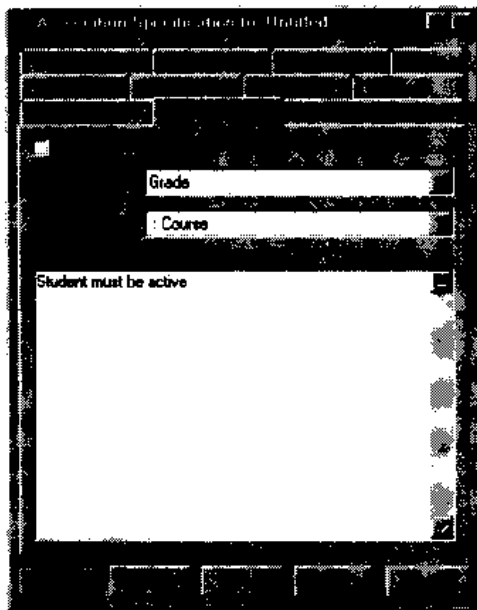


图7.30 关系限制

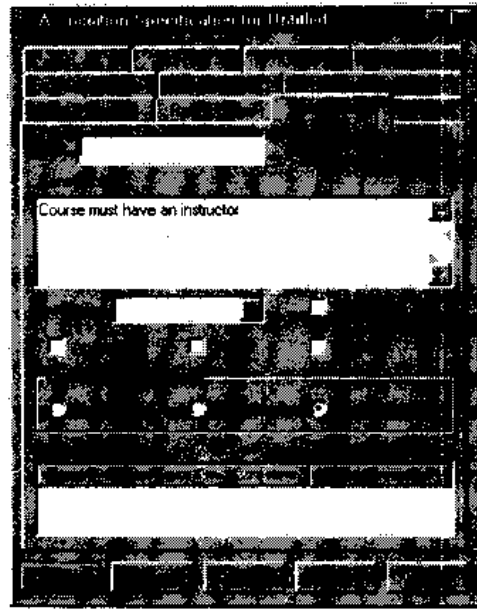


图7.31 作用限制

练习

本练习在参与Enter New Order使用案例的类之间增加关系。

问题

Karen在类中增加属性操作之后，就可以生成代码了。但她首先还要检查类之间的关系。

她从增加订单的Sequence框图中寻找关系。任何在Sequence框图中通信的类都要在Class框图中有个关系。一旦标出关系后，就要将其加进模型中。

增加关系

在参与Enter New Order使用案例的类之间增加关系。

练习步骤

设置

1. 在浏览器中找到Add New Order Class框图。
2. 双击打开框图。
3. 在框图工具栏中寻找Unidirectional Association按钮。如果没有，则继续第5步，否则跳到练习下一部分。
4. 右单击框图工具栏中并从弹出菜单选择Customize。
5. 在工具栏中增加按钮Creates A Unidirectional Association。

增加关联

1. 选择Unidirectional Association工具栏按钮。

2. 从OrderOptions到OrderDetail类拖动一个关联。
3. 重复1、2步，画出下列关联：
 - 从OrderDetail到OrderMgr
 - 从OrderMgr到Order
 - 从OrderMgr到TransactionMgr
 - 从TransactionMgr到Order
 - 从TransactionMgr到OrderItem
 - 从Order到OrderItem
4. 右单击OrderOptions与OrderDetail之间的单向关联，在OrderOptions类旁边单击。
5. 从弹出菜单选择Multiplicity►Zero or One。
6. 右单击单向关联的另一端。
7. 从弹出菜单选择Multiplicity►Zero or One。
8. 重复第4到第7步，将其余倍增性加进框图中，如图7.32。

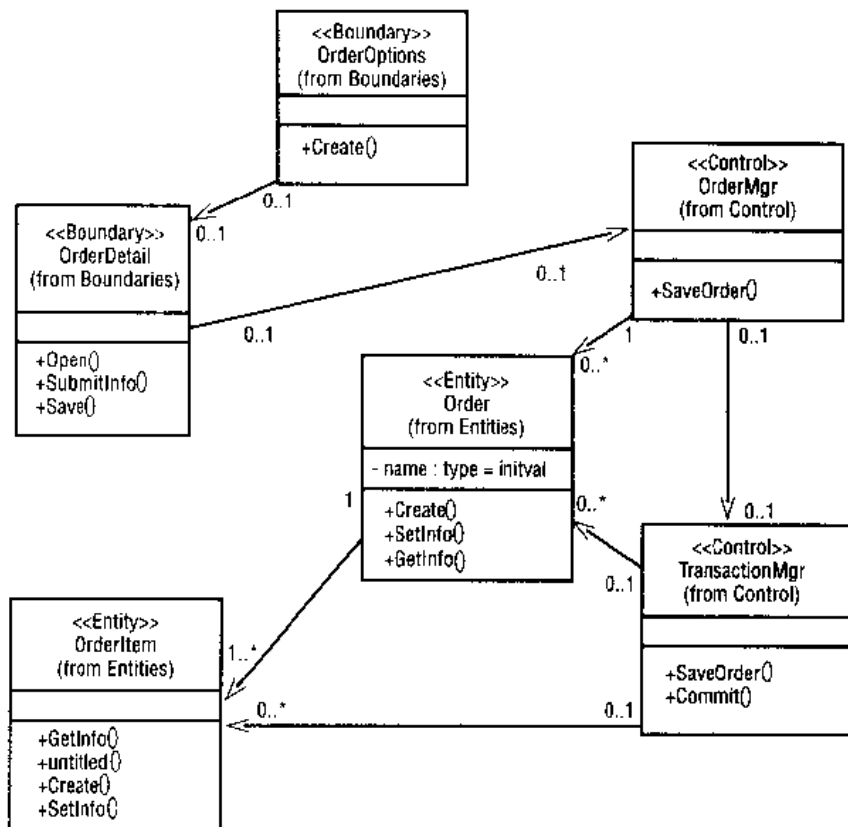


图7.32 Add New Order使用案例的关联

小结

本章介绍了UML中不同类型的关系——关联、依赖性、累积和一般化。每种关系都可以通过Class框图加进Rose模型中。加进关系后，就可以加进各种细节，如关系名、作用名、限定符和倍增性。下一章介绍对象行为，通过单个类检查类的各种状态，以及类如何从一种状态过渡到另一种状态。

第8章 对象行为

- 生成State Transition框图
- 将活动、进入操作、退出操作、事件和状态历史加进状态中
- 将事件变元、保证条件、操作和发送事件加进过度中
- 增加起、停、嵌套状态

前面介绍了类及其相互关系，这里要介绍一个对象的行为。一个对象有一种或多种状态。例如，员工可以雇用、解聘、试用、离职或退休。**Employee**类在这几种状态中的表现各不相同。

本章介绍State Transition框图，其中包括对象存在的不同状态信息、对象如何从一种状态过渡到另一种状态、以及对象在不同状态中的不同行为。

State Transition框图

State Transition框图显示一个对象从生成到删除的生命周期。这些框图可以造型类的动态行为。在一般项目中，不对每个类生成State Transition框图。事实上，许多项目根本不用State Transition框图。图8.1是Course类的State Transition框图。

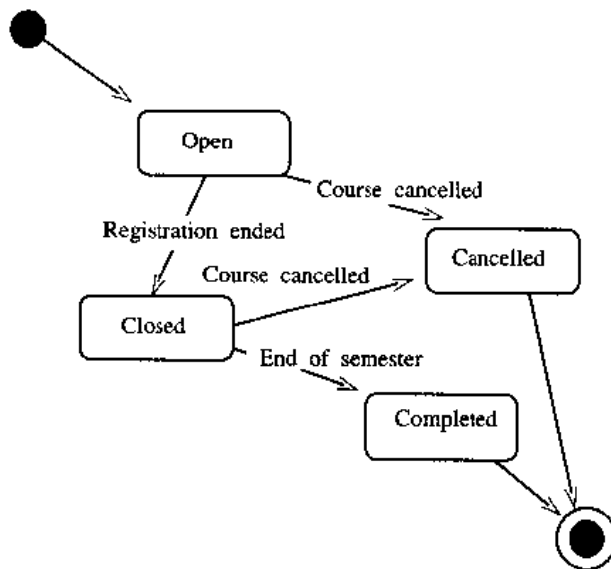


图8.1 Course类的State Transition框图

如果一个类有一些重要的动态行为，则可以生成State Transition框图。一个类有一些重要的动态行为时具有多种状态。在上面的Course类例子中，Course对象可以开课、关闭、取消或完成。在本章的练习中，我们要介绍Order类。Order对象有多种状态，包括等待、填写和取消。订单在不同状态中的行为各不相同。

要确定哪个类有重要的动态行为，首先要看它的属性。考虑一个类的实例在属性值不同时如何表现。如果有个属性**Status**，则表示不同状态。这个属性取不同值时，表现有什么不同呢？

也可以检查类的关系。看看倍增性中带0的关系。0表示这个关系是可选的。关系存在和不存在时类的实例是否表现不同呢？如果是，则可能有多种状态。例如，看看人员与公司间的关系，如果有关系，则人员处于雇用状态，否则此人被解聘或退休。

Rose中，**State Transition**框图不生成源代码。这些框图只是建档类的动态行为，使开发人员和分析人员能更好地了解这个行为。开发人员最终要负责实现这个框图中描述的逻辑。和其他UML框图一样，**State Transition**框图让小组有机会先讨论和建立逻辑再进行编码。

生成State Transition框图

Rose中可以对每个类生成**State Transition**框图。类的所有状态和过渡都体现在这个框图中。在浏览器中，**State Transition**框图放在类下面。**State Transition**框图的Rose图标如下：



要生成**State Transition**框图：

1. 右单击浏览器中所要类。
2. 从弹出菜单选择**Open State Diagram**。

或

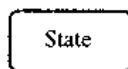
1. 在**Class**框图中选择所要类。
2. 选择**Browse►State Diagram**。

增加状态

状态是对象存在的可能条件。前面曾介绍过，可以从两个地方确定对象状态：属性值和与其他对象的关系。考虑属性可取的不同值和关系存在与不存在时的对象状态。

和其他Rose元素一样，可以在状态中增加文档。但由于这些框图不生成代码，因此生成代码时不插入状态文档的说明语句。

在UML中，状态用圆角矩形表示：



要增加状态：

1. 从工具箱工具栏中选择**State**。
2. 单击**State Transition**框图中状态要出现的位置。

或

1. 选择**Tools►Create►State**。
2. 单击**State Transition**框图中状态要出现的位置。

要将文档加进状态中：

1. 双击所要状态打开状态规范窗口。

2. 选择General标签。
 3. 在Documentation字段中输入文档。
- 或
1. 选择所要状态。
 2. 选择Browse>Specification。
 3. 选择General标签。
 4. 在Documentation字段中输入文档。

增加状态细节

对象处于特定状态时，可能要进行一些活动，例如生成报表。进行计算或向另一对象发送事件。Rose中可以通过状态规范窗口将这类信息加进模型中。

一个状态可以加进五种信息：活动、进入操作、退出操作、事件和状态历史。下面要举例说明这五种信息。图8.2是ATM系统Account类的State Transition框图。

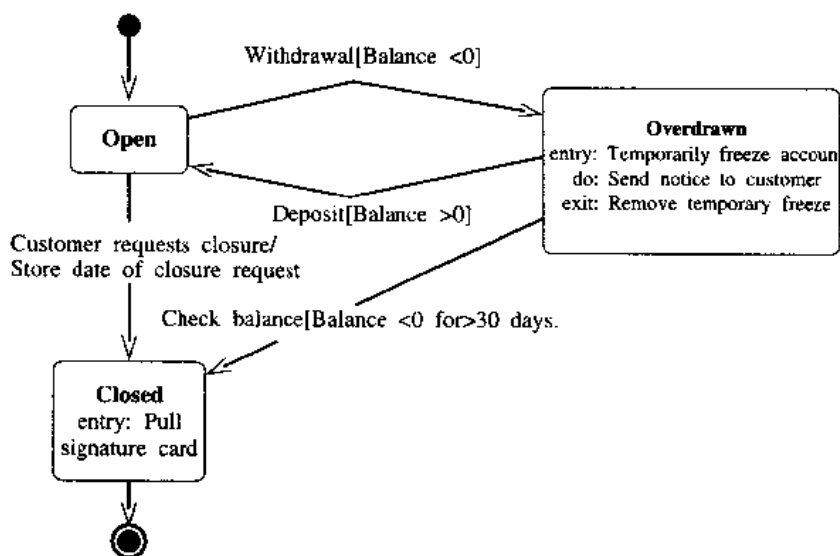
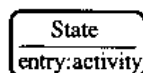


图8.2 ATM系统Account类的State Transition框图

活动

活动是对象在特定状态时进行的行为。例如，帐号处于关闭状态时，取帐号持有者的签名卡。活动是可中断的行为，可以在对象处于该状态时运行完毕，也可以在对象转入另一状态时中断。

活动在状态内显示，前面加上do和冒号。

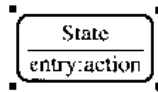


进入操作

进入操作（entry action）是对象进入某个状态时发生的行为。下面以帐号为例。帐号进入透支状态时，不管来自哪种状态，都发生“Temporarily freeze account”（暂时冻结帐

号)的进入操作。但这个操作在对象进入透支状态后并不发生,而是发生在状态过渡时,与活动不同的是,进入操作是不可中断的。

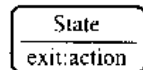
进入操作在状态内显示,前面加上entry和冒号。



退出操作

退出操作(exit action)与进入操作相似,但它在退出某个状态时发生。在帐号例子中,帐号离开透支状态时,不管来自哪种状态,都发生Remove Temporary Freeze(取消暂时冻结)退出操作。和进入操作一样,退出操作也是不可中断的。

退出操作在状态内显示,前面加上exit和冒号。



活动、进入操作和退出操作中的行为可以包括向另一对象发送事件。例如,帐号对象可能发出一个事件给读卡机对象,这时活动、进入操作和退出操作前面加上^。框图变成:

Do: ^Target.Event(Arguments)

其中Target是接收事件的对象,Event是要发的消息,Arguments是所发消息的参数。在Rose中,可以向发出的事件增加这些细节。

活动还可能在收到某个事件后发生。例如,帐号处于打开状态,发生某个事件时,会进行活动。

所有这些项目都可以通过状态规范窗口的Detail标签加进Rose模型,如图8.3。

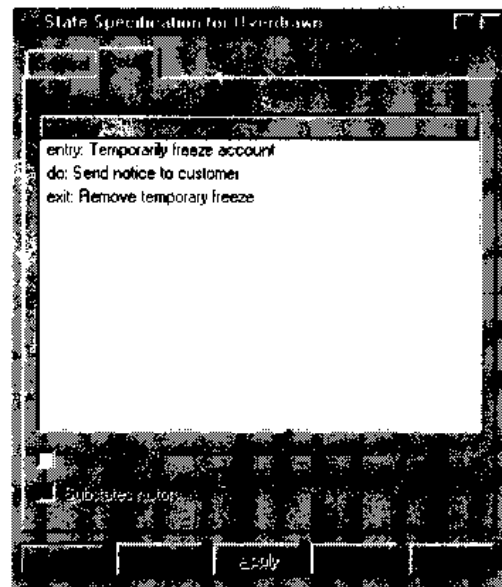


图8.3 状态规范窗口

要加进活动:

1. 打开所要状态的规范窗口。

2. 选择Detail标签。
3. 右单击Actions框。
4. 从弹出菜单选择Insert。
5. 双击新操作。
6. 在Actions字段中输入操作。
7. 在When框中选择Entry Until Exit, 使新操作成为活动。

要增加进入操作:

1. 打开所要状态的规范窗口。
2. 选择Detail标签。
3. 右单击Actions框。
4. 从弹出菜单选择Insert。
5. 双击新操作。
6. 在Actions字段中输入操作。
7. 在When框中选择On Entry。

要增加退出操作:

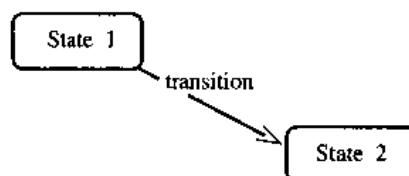
1. 打开所要状态的规范窗口。
2. 选择Detail标签。
3. 右单击Actions框。
4. 从弹出菜单选择Insert。
5. 双击新操作。
6. 在Actions字段中输入操作。
7. 在When框中选择On Exit。

要发出事件:

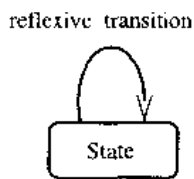
1. 打开所要状态的规范窗口。
2. 选择Detail标签。
3. 右单击Actions框。
4. 从弹出菜单选择Insert。
5. 双击新操作。
6. 选择Send Event类型。
7. 在相应字段中输入事件、变元和目标。

增加过渡

过渡 (transition) 是从一种状态变到另一种状态。框图上的过渡显示对象如何从一种状态变到另一种状态。框图中每个过渡画成从源状态指向新状态的箭头。



过渡也可以反身。发生某种事件时，对象可能过渡回当前状态。反身过渡显示为指向同一状态的箭头。



要增加过渡：

1. 从工具箱工具栏中选择Transition。
2. 单击过渡开始的状态。
3. 向过渡结束状态拖动一条过渡线。

要增加反身过渡：

1. 从工具箱工具栏中选择Transition to Self。
2. 单击反身过渡所在的状态。

或

1. 选择Tools►Create►Loop。
2. 单击反身过渡所在的状态。

或

1. 从工具箱工具栏中选择Transition。
2. 单击过渡开始的状态。
3. 向该状态之外拖动过渡线并放开。
4. 将过渡线拖回这个状态。

要将文档加进过渡中：

1. 双击所要状态打开过渡规范窗口。
2. 选择General标签。
3. 在Documentation字段中输入文档。

增加过渡细节

每个过渡可以指定各种规范。包括事件、变元、保证条件、操作和发送事件。下面在ATM例子中介绍这些规范。图8.2是Account类的State Transition框图。

事件

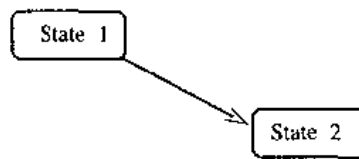
事件导致类从一种状态变到另一种状态。在帐号例子中，Customer Requests Closure（客户请求关闭）事件使帐号从打开状态变成关闭状态。事件在框图中显示在过渡箭头上。

在框图中，事件可以用操作名或英语短语描述。在帐号例子中，事件用英语短语。如果用操作名，则Customer Requests Closure事件可以写成RequestClosure()。

事件可以有变元，例如，存款事件将帐号从透支状态变到打开状态，可能有变元Amount，表示存款金额，在Rose模型中，可以将变元加进事件中。

大多数过渡都有事件，导致从一种状态变到另一种状态。但也可以没有事件而自动过

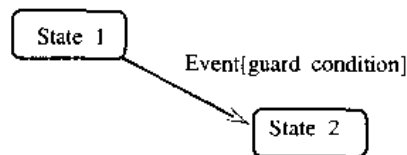
渡。对于自动过渡，对象在发生所有进入操作、活动和退出操作后自动从一种状态移到另一种状态。



保证条件

保证条件 (guard condition) 控制过渡何时发生或不发生。在帐号例子中，存款事件在帐号结余大于0时将帐号从透支状态变到打开状态。如果结余不大于0，则发生过渡。

保证条件在过滤线上画出，放在事件名后面的方括号中。



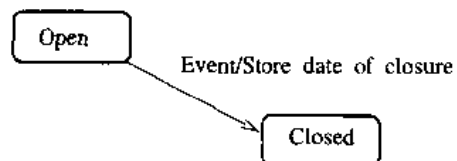
保证条件是可选的，但如果有多状态自动过渡结果，则要对每个自动过渡提供互相排斥的保证条件，使框图读者能知道自动取哪个路径。

操作

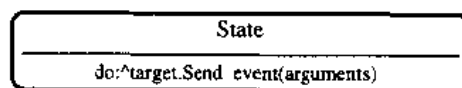
操作是过渡中发生的不可中断行为。进入和退出操作在状态内显示，定义每次对象进入或退出状态时的行为。但大多数操作是在过渡线上画出的，因为不是在每次对象进入或退出状态时发生。

例如，从打开状态过渡到关闭状态时，发生Store Date of Closure Request (存放关闭请求日期) 操作。这是个不可中断操作，只在从打开状态过渡到关闭时发生。

操作在过渡线上显示，放在事件名和斜杠后面。



事件和操作可能是对象内发生的行为，也可能是向另一对象发送的消息。如果事件和操作向另一对象发送，则框图中要在其前面加上^。



要增加事件：

1. 双击所要过渡打开规范窗口。
2. 选择General标签。
3. 在Event字段中输入事件。

要将变元加进事件中：

1. 双击所要过渡打开规范窗口。
2. 选择General标签。
3. 在Arguments字段中输入变元。

要增加保证条件：

1. 双击所要过渡打开规范窗口。
2. 选择Detail标签。
3. 在Condition字段中输入保证条件。

要增加操作：

1. 双击所要过渡打开规范窗口。
2. 选择Detail标签。
3. 在Action字段中输入操作。

要发出事件：

1. 双击所要过渡打开规范窗口。
2. 选择Detail标签。
3. 在Send Event字段中输入事件。
4. 在Send Arguments字段中输入变元。
5. 在Send Target字段中输入目标。

增加特殊状态

框图中可以加入两个特殊状态：开始状态和结束状态。

开始状态

开始状态是对象初始生成时的状态。在帐号例子中，开始状态打开新帐号。开始状态在框图中显示为实心圆。从这个圆向初始状态画一条过渡。



开始状态是强制的，框图读者要知道新对象的状态。框图中只能有一个开始状态。

停止状态

停止状态是对象删除时的状态。停止状态在框图中显示为牛眼。停止状态是可选的，可以有多个停止状态。



要增加开始状态：

1. 从工具箱工具栏中选择Start State。
2. 单击State Transition框图要出现开始状态处。

要增加停止状态：

1. 从工具箱工具栏中选择**End State**。
2. 单击**State Transition**框图要出现停止状态处。

使用嵌套状态

要减少框图的复杂性，可以在一个状态中嵌套一个或几个状态。嵌套状态称为子状态，而大状态称为父状态。

如果两个或几个状态有相同的过渡，则可以组成父状态。然后不是对每个状态维护一个过渡，而是把过渡移到父状态的。图8.1是不带嵌套状态的框图。

图8.4是带嵌套状态的框图。

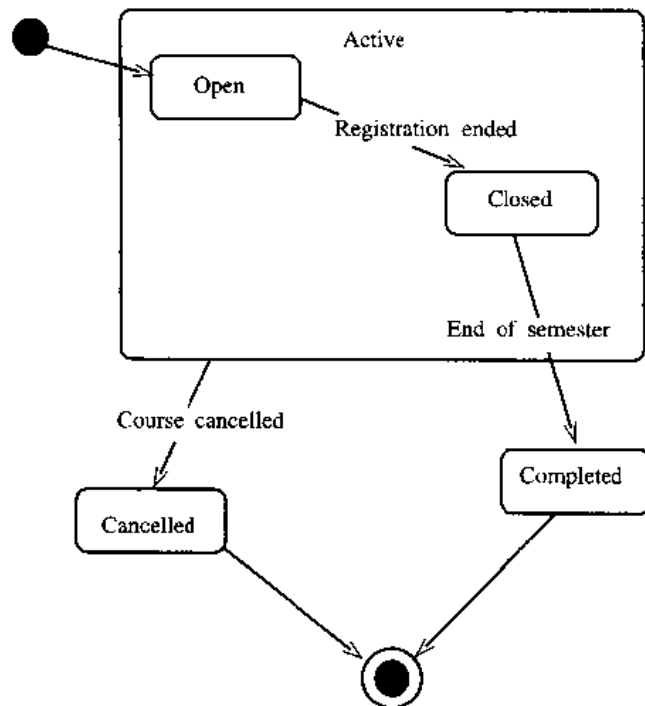


图8.4 带嵌套状态的框图

可以看出，父状态可以减少**State Transition**框图的复杂性。

有时要让系统记住最后状态。如果父状态中有三个状态，然后离开父状态，可能要让系统记住最后离开时的状态。

解决这个问题的方法有两种。第一是在父状态中增加一个开始状态。开始状态表示父状态中的缺省起点。对象首次进入父状态时，处于这种开始状态。

第二是用状态历史记住对象状态。如果设置历史选项，则对象可以离开父状态，然后返回正确的状态。历史选项显示为框图角上的“H”字样，放在圆圈中，如图8.5。

要嵌套状态：

1. 从工具箱工具栏中选择**State**。
2. 单击要嵌套新状态的状态。

要使用状态历史：

1. 打开所要状态的规范窗口。
2. 选择Detail标签。
3. 选择tates History框。
4. 如果状态中的状态中有状态，则可以对父状态中的所有嵌套状态采用历史特性。为此，选择tates History框。

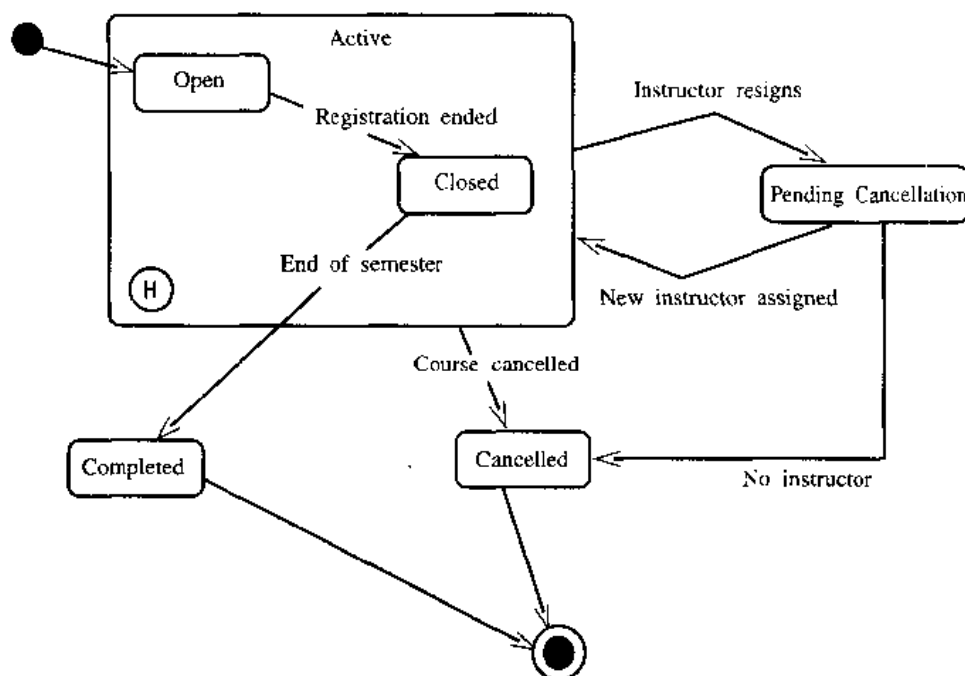


图8.5 父状态历史

练习

本练习对Order类生成State Transition框图。

问题

设计Order类时，Karen认识到这个类很重要。订单状态改变时，许多要求大大改变。待结订单与填充订单不同，填充订单又与取消订单不同。

要保证设计合理，她和其他开发人员一起构造这个类的State Transition框图。利用这个信息，开发人员就能明确了解如何编写这个类。

生成State Transition框图

对Order类生成图8.6所示的State Transition框图。

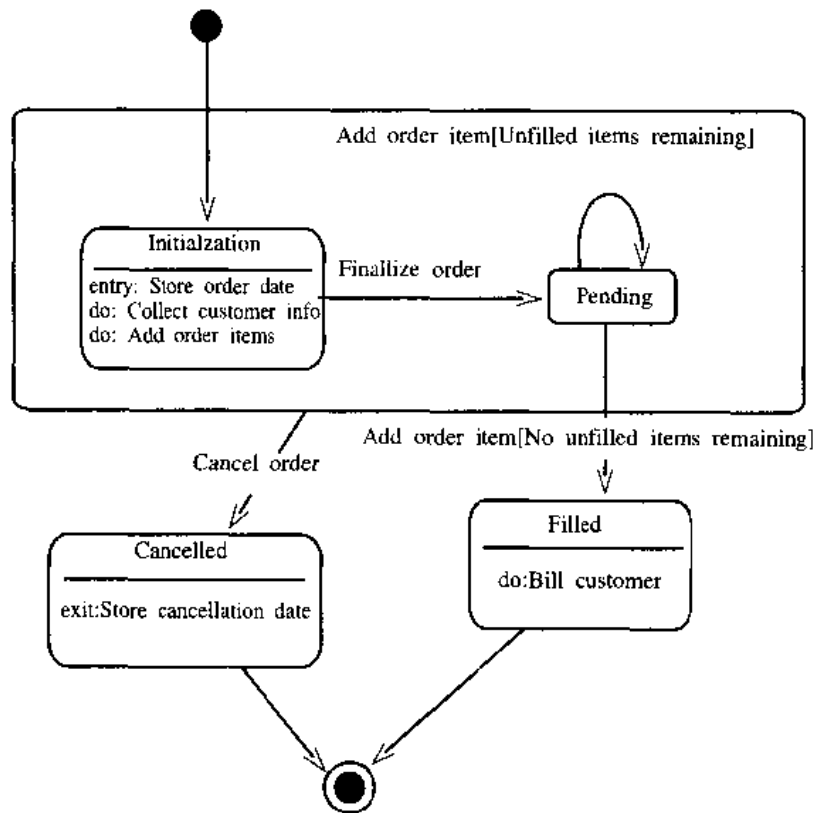


图8.6 Order类的State Transition框图

练习步骤

生成框图

1. 在浏览器中找到Order类。
2. 右单击这个类并选择Open State Diagram。

增加开始和停止状态

1. 从工具箱中选择Start State。
2. 在框图中放上状态。
3. 从工具箱中选择End State。
4. 在框图中放上状态。

增加父状态

1. 从工具箱中选择State。
2. 在框图中放上状态。

增加其他状态

1. 从工具箱中选择State。
2. 在框图中放上状态。

3. 将状态取名Cancelled。
4. 从工具箱中选择State。
5. 在框图中放上状态。
6. 将状态取名Filled。
7. 从工具箱中选择State。
8. 在框图中放上状态。
9. 将状态取名Initialization。
10. 从工具箱中选择State。
11. 在框图中放上状态。
12. 将状态取名Pending。

增加状态细节

1. 双击Initialization状态。
2. 选择Detail标签。
3. 右单击Actions框。
4. 从弹出菜单选择Insert。
5. 双击新操作。
6. 将新操作命名为Store Order Date。
7. 保证When框选择On Entry。
8. 重复第3到第7步，加入下列操作：
 - Collect Customer Info, Entry until Exit
 - Add Order Items, Entry until Exit
9. 单击OK两次，关闭规范窗口。
10. 双击Cancelled状态。
11. 重复第2到第7步，加入下列操作：
 - Store Cancellation Date, On Exit
12. 单击OK两次，关闭规范窗口。
13. 双击Filled状态。
14. 重复第2到第7步，加入下列操作：
 - Bill Customer, Entry until Exit
15. 单击OK两次，关闭规范窗口。

增加过渡

1. 从工具箱中选择Transition。
2. 单击Start State。
3. 向Initialization状态拖动过渡线。
4. 重复第1到第3步增加下列过渡：
 - Initialization to Pending
 - Pending to Filled

- Superstate to Cancelled
 - Cancelled to End State
 - Filled to End State
5. 从工具箱中选择Transition to Self。
 6. 单击Pending状态。

增加过渡细节

1. 双击Initialization to Pending过渡打开规范窗口。
2. 在Event字段中输入Finalize Order。
3. 单击OK关闭规范窗口。
4. 重复第1到第3步，将事件Cancel Order加进父状态与Cancelled state状态间的过渡。
5. 双击Pending to Filled过渡打开其规范。
6. 在Event字段中输入Add Order Item。
7. 选择Detail标签。
8. 在Condition字段中输入No unfilled items remaining。。
9. 单击OK关闭规范窗口。
10. 双击Pending状态中的反身过渡(Transition to Self)。
11. 在Event字段中输入Add order item。
12. 选择Detail标签。
13. 在Condition字段中输入Unfilled items remaining。
14. 单击OK关闭规范窗口。

小结

本章介绍了Rose支持的另一个UML框图——State Transition框图。尽管源代码不从这些框图产生，但它们在检查、调试和建档类的动态行为时非常有用。

State Transition框图显示对象存在的各种状态，对象如何从一种状态变到另一种状态。所有这些信息都在类的详细设计中。开发人员可以用这个信息编制类。

在Rose中，可以从一个类生成State Transition框图。至多可以对每个类生成一个State Transition框图，显示类的状态和过渡。并非每个类都需要State Transition框图，只有具有重要动态行为的类才需要。要确定一个类是否具有重要动态行为，可以检查其属性的取值和具有的关系。

下一章介绍源代码生成，介绍Rose的Component视图。在Component视图中，我们要从逻辑设计转入实际设计，介绍代码库、执行文件和系统的其他组件。

第9章 Component视图

- 探索组件类型
- 生成组件和将类映射组件
- 使用组件框图

现在转入Rose的Component视图。在Component视图中，我们着重考虑系统的实际结构。首先，我们要确定类如何组成代码库，然后要介绍不同的执行文件、数据链接库（DLL）文件和系统中的其他运行文件。我们目前还不关心不同文件在网络上的位置，这些问题要放到Deployment视图中介绍。

何谓组件

组件是代码的物理模块。组件可以包括代码库和运行文件。例如，如果使用C++，则每个.CPP和.H文件是单独的组件。编译代码后生成的.EXE文件也是组件。

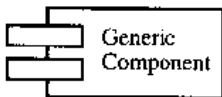
生成代码之前，将每个文件映射相应组件。例如，在C++中，每个类映射两个组件，一个表示类的.CPP文件，一个表示类的.H文件。在Java中，每个类映射一个组件，表示这个类的.JAVA文件。生成代码时，Rose用组件信息生成相应的代码库文件。

一旦生成组件后，它们加进Component框图中，并在其间画出关系。组件间唯一的关系类型是依赖性关系。依赖性关系要求一个类要在另一个类之前编译。稍后将对此详细介绍。

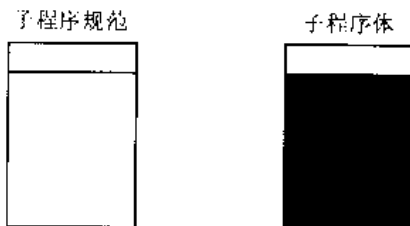
组件类型

Rose中可以用几种不同图标表示不同类型的组件。前面介绍了两种主要组件：源代码库和运行组件。在这两种组中各有几个不同的图标。下面先介绍源代码库组件：

组件（Component） 组件图标表示具有定义接口的软件模块。在Component规范中，可以用Stereotype字段指定组件类型（如ActiveX、Applet、Application、DLL和Executables等等）。下面版型一节介绍这个图标可以适用的不同版型。



子程序规范和体（Subprogram Specification和Body） 这些图标表示子程序的显示规范和实现体子程序通常是一组子程序集名。子程序不包含类定义。



主程序 (Main Program) 主程序是包含程序根的文件。例如，在PowerBuilder中，这个文件包含应用程序对象。

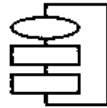


包规范和体 (Package Specification和Body) 包是类的实现方法。包规范是头文件，包含类的函数原型信息。在C++中，包规范为.H文件；包体包含类操作代码，在C++中，包体为.CPP文件。

Java中可以用包规范图标表示.JAVA文件。

还有其他组件图标，适用于运行组件。运行组件包括执行文件、DLL文件和任务。

Package Specification



Package Body



DLL文件 这个图标表示数据链接库 (DLL) 文件。

DLL



任务规范和体 (Task Specification和Body) 这些图标表示具有独立控制线程的包。执行文件通常表示为具有.EXE扩展的任务规范。

Task Specification



Task Body



Component框图

Component框图是个UML框图，显示系统中的组件及其相互依赖性。图9.1是个Component框图。

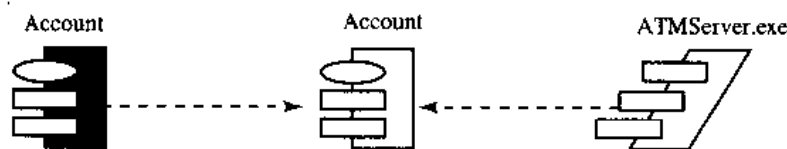


图9.1 Component框图

从Component框图可以看出系统中的源代码和运行组件。

利用这个框图，负责编译和部署系统的人员知道有哪些代码库，编译代码时生成哪些执行文件。开发人员知道有哪些代码库，相互间有什么关系。组件依赖性使负责编译的人员知道组件所需的编译顺序。

生成Component框图

在Rose中，可以在Component视图中生成Deployment框图。生成框图后，可以直接在框图中生成组件或将现有组件从浏览器中拖动到框图中。

在浏览器中，Component组件用下列图标显示：



要在Component视图中生成Component框图：

1. 在浏览器中，右单击包含Component框图的包。
2. 从弹出菜单选择New>Component Diagram。
3. 输入新Component框图名。

或

1. 选择Browse>Component Diagram打开Select Component Diagram窗口。
2. 选择所要包。
3. 从Component Diagram框中选择<New>并单击OK。
4. 输入新Component框图名并单击OK。

要删除Component框图：

1. 在浏览器中，右单击Component框图。
2. 从弹出菜单选择Delete。

或

1. 选择Browse>Component Diagram打开Select Component Diagram窗口。
2. 选择所要包。
3. 选择要删的组件。
4. 单击Delete按钮。

增加组件

生成Component框图后，下一步要增加组件。在Component框图工具栏中，列出了上述图标的按钮。在In C++、Java或Visual Basic项目中，最常用的图标是包规范、包体和执行文件图标。前面曾介绍过，包规范图标用于.H文件。包规范和组件图标可以用于.JAVA文件、Visual Basic项目和DLL文件。包体图标用于.CPP文件。

还可以将文档加进组件中。文档包括组件用途说明和组件类说明。

和类一样，组件可以包装或组织。通常，对每个逻辑视图包生成一个组件视图包。例如，如果逻辑视图包Orders包含类，则对应的组件视图包包含Order、OrderItem和OrderForm类的组件。

要增加组件：

1. 选择工具箱工具栏中的Component。

2. 单击框图中要放新组件的位置。

3. 输入新组件名。

或

1. 选择Tools►Create►Component。

2. 单击框图中要放新组件的位置。

3. 输入新组件名。

或

1. 在浏览器中右单击组件所在的包。

2. 从弹出菜单选择New►Component。

3. 输入新组件名。

要将文档加进组件中:

1. 右单击所要组件。

2. 从弹出菜单选择Open Specification打开组件规范窗口。

3. 选择General标签。

4. 在Documentation字段中输入文档。

或

1. 双击所要组件打开组件规范窗口。

2. 选择General标签。

3. 在Documentation字段中输入文档。

或

1. 选择所要组件。

2. 选择Browse►Specification打开组件规范窗口。

3. 选择General标签。

4. 在Documentation字段中输入文档。

或

1. 选择所要组件。

2. 在Documentation字段中输入文档。

要从框图中删除组件:

1. 从框图中选择组件。

2. 按Delete。

说明: 组件从框图中删除, 但在浏览器和其他Component框图仍然存在。

要从模型中删除组件:

1. 从框图中选择组件。

2. 选择Edit►Delete from Model或按Ctrl+D。

或

1. 右单击浏览器中的组件。

2. 从弹出菜单中选择Delete。

说明: 组件从浏览器和所有Component框图中删除。

增加组件细节

和其他Rose模型元素一样，每个组件可以增加许多细节规范，包括：

版型 第一个细节是组件版型。版型控制用哪个图标表示组件。

上面列出的版型是<none>（使用Component图标）、子程序规范、子程序体、主程序、包规范、包体、执行文件、DLL、任务规范和任务体。此外，Rose中还有ActiveX、Applet、Application、Generic Package和Generic Subprogram文件的版型。如果需要，还可以生成新版型，表示特定编程语言和应用程序中的新型组件。

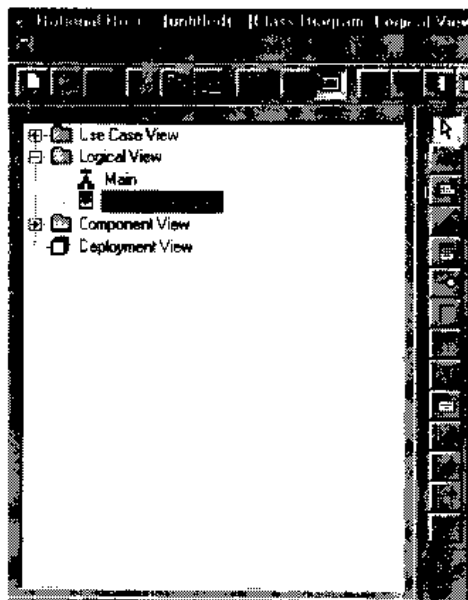
语言 在Rose中，可以对各个组件分别指定语言。因此，模型的一个部分可以用C++，一个部分可以用Java，再一个部分可以用Visual Basic等等，只要安装企业版Rose即可。这是Rose 98增加的新特性。

Rose企业版包含C++、Java、Visual Basic和Oracle8的加入文件。不同厂家提供了许多插入件，可以扩展Rose功能。市面上还能买到其他语言（PowerBuilder、Forte、Visual Age Java等等）的插入件。关于Rose链接伙伴的完整清单，见Rational Rose Web站点www.rational.com。

声明 在Rose中，可以包括每个组件生成代码期间要加进的补充声明。声明包括语言特定语句，用于声明变量、类等等。C++ #include语句也是个声明。

类 生成类的代码之前，要先将类映射组件。这个映射能使Rose知道类代码应存放在哪个实际文件中。

每个组件可以对应一个或几个类。将类映射组件后，组件名会出现在逻辑视图中类名后面的括号内。



要指定版型：

1. 打开所要组件的标准规范窗口。
2. 选择General标签，如图9.2。
3. 在Stereotype字段中输入版型。

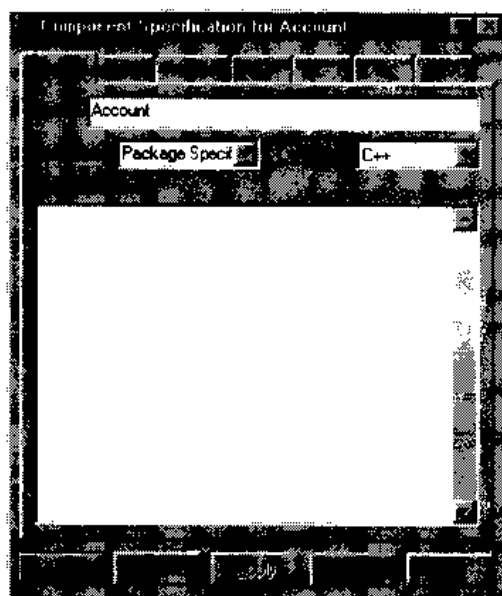
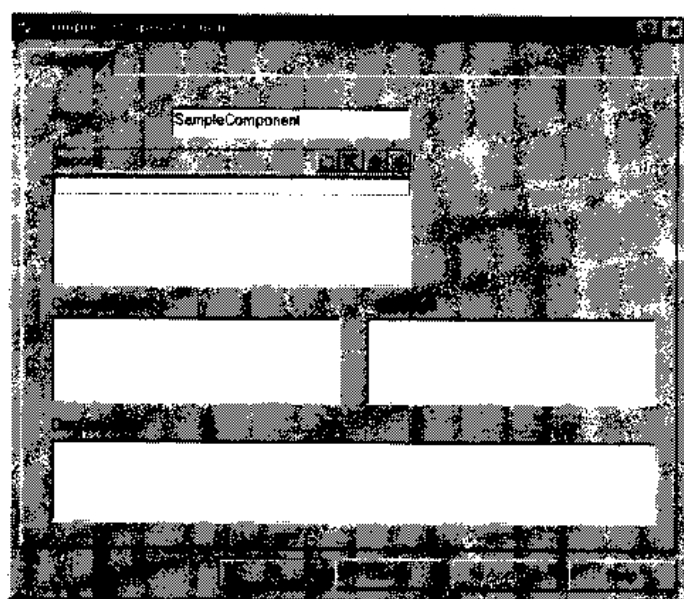


图9.2 对组件指定版型

如果使用Rose 98i, 而组件为Java或CORBA组件, 则还有另一个组件规范窗口, 如下图。



或

1. 选择所要组件。
2. 在<<>>中输入版型: <<Name >>。

要指定语言:

1. 打开所要组件的标准规范窗口。
2. 选择General标签。
3. 在Language字段中选择语言。

要增加声明:

1. 打开所要组件的标准规范窗口。
2. 选择General标签，如图9.3。
3. 在Declarations字段中输入声明。

1. 打开所要组件的标准规范窗口。
2. 选择Realizes标签，如图9.4。

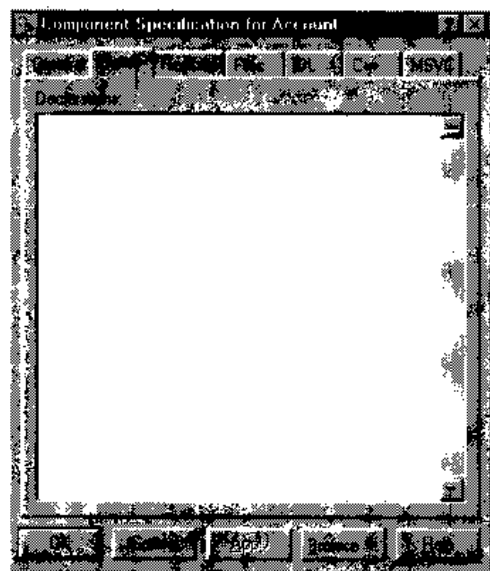


图9.3 增加组件声明

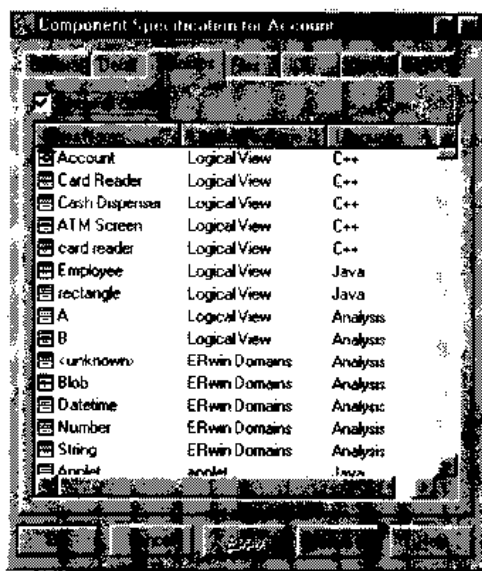


图9.4 将类映射组件

3. 右单击要映射的类。
 4. 从弹出菜单选择Assign。
- 或
1. 在浏览器中选择要映射的类。
 2. 将类拖动到浏览器或框图中所要组件上。

将文件和URL连接组件

可以直接将文件和URL连接组件。例如，可以将一个源代码文件连接模型中的组件，或者将测试组件功能的测试脚本连接组件。Rose可以通过浏览器或组件规范窗口将文件和URL连接组件。

要将文件连接组件：

1. 右单击浏览器或Component框图中的组件。
2. 从弹出菜单选择Open Standard Specification。
3. 选择Files标签。
4. 右单击Files标签内任一空白处。
5. 从弹出菜单选择Insert File。
6. 用Open对话框寻找要连接的文件。
7. 选择Open将文件连接组件。

或

1. 右单击浏览器中的组件。
2. 选择New►File。
3. 用Open对话框寻找要连接的文件。
4. 选择Open将文件连接组件。

要将URL连接组件:

1. 右单击浏览器或Component框图中的组件。
2. 从弹出菜单选择Open Standard Specification。
3. 选择Files标签。
4. 右单击Files标签内任一空白处。
5. 从弹出菜单选择Insert URL。
6. 输入要连接的URL名。

或

1. 右单击浏览器中的组件。
2. 选择New►URL。
3. 输入要连接的URL名。

要打开连接的文件:

1. 在浏览器中找到文件。
2. 双击文件名。Rose自动启动相应应用程序并装入文件。

或

1. 右单击浏览器中的文件。
2. 从弹出菜单选择Open。Rose自动启动相应应用程序并装入文件。

或

1. 右单击浏览器或Component框图中的组件。
2. 从弹出菜单选择Open Standard Specification。
3. 选择Files标签。
4. 双击文件名。Rose自动启动相应应用程序并装入文件。

或

1. 右单击浏览器或Component框图中的组件。
2. 从弹出菜单选择Open Standard Specification。
3. 选择Files标签。
4. 右单击要打开的文件。
5. 从弹出菜单选择Open File/URL。Rose自动启动相应应用程序并装入文件。

要打开连接的URL:

1. 在浏览器中找到URL。
2. 双击URL名。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 右单击浏览器中的URL。
2. 从弹出菜单选择Open。Rose自动启动Web浏览器应用程序并装入URL。

或

1. 右单击浏览器或Component框图中的组件。
2. 从弹出菜单选择Open Standard Specification。
3. 选择Files标签。
4. 双击URL名。Rose自动启动Web浏览器应用程序并装入URL。

或

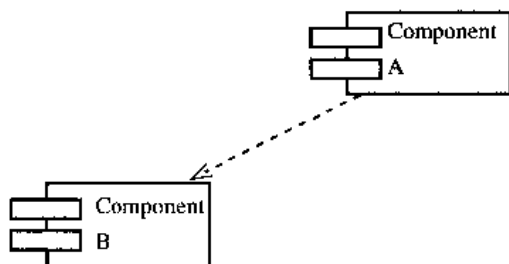
1. 右单击浏览器或Component框图中的组件。
2. 从弹出菜单选择Open Standard Specification。
3. 选择Files标签。
4. 右单击要打开的URL。
5. 从弹出菜单选择Open File/URL。Rose自动启动Web浏览器应用程序并装入URL。

要删除连接的文件或URL：

1. 右单击浏览器中的文件或URL。
2. 从弹出菜单选择Delete。

增加组件依赖性

组件之间存在的唯一关系是组件依赖性（component dependency）。组件依赖性指一个组件依赖于另一组件。组件依赖性画成组件之间的虚线箭头。



本例中，组件A依赖于组件B。换句话说，A中的有些类依赖于B中的有些类。

这种组件依赖性在编译时有意义。本例中，由于A依赖于B，因此要先编译B之后才能编译A。别人看了这个框图就知道要先编译B之后才能编译A。

和包依赖性一样，要避免循环组件依赖性。如果A依赖于B，B又依赖于A，则两者都等待对方先编译。这样，就要把两者当作一个大组件处理。必须先删除循环依赖性之后才能生成代码。

组件依赖性还有维护意义。如果A依赖于B则B中的任何改变均会影响A。维护人员可以用这个框图评估改变时的影响。一个组件依赖的组件越多，就越会受到改变的影响。

最后依赖性还可以确定复用性。本例中A很难复用，因为A依赖于B，必须复用B才能复用A。而B则很容易复用，因为它依赖于任何其他组件。一个组件依赖的组件越少，复用性越好。

要增加组件依赖性：

1. 选择工具箱中的Dependency图标。
2. 从Client组件向Supplier组件拖动一条依赖性线。

或

1. 选择Tools►Create►Dependency。
2. 从Client组件向Supplier组件拖动一条依赖性线。

要删除组件依赖性：

1. 选择所要组件依赖性。
2. 按Delete。

或

1. 选择组件依赖性。
2. 选择Edit►Delete。

练习

本练习中生成订单处理系统的Component框图。目前我们已经标识Enter New Order使用案例所需的类。建立其他使用案例时，要向框图中增加新组件。

问题

完成分析和设计之后，开发小组的成员Dan生成Component框图。这时，小组已经决定用C++，因此他准备对每个类生成相应组件。

图9.5是整个系统的主Component框图。这个框图关注要生成的组件包。

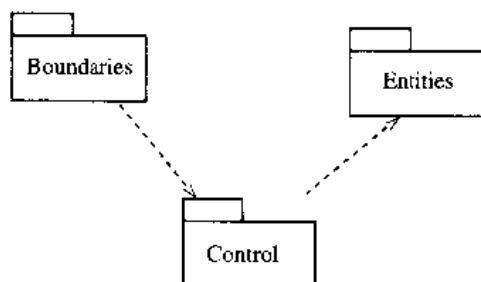


图9.5 订单处理系统的Component框图

图9.6是实体包中的所有组件。这些组件包含放在逻辑视图中实体包中的所有类。

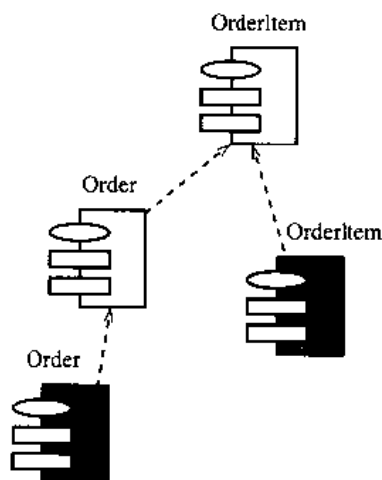


图9.6 实体包Component框图

图9.7是控制包中的组件。这些组件包含放在逻辑视图中控制包中的所有类。

图9.8是边界包中的组件。这些组件包含放在逻辑视图中边界包中的所有类。

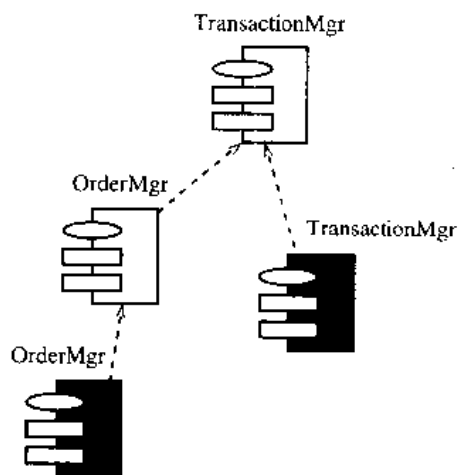


图9.7 控制包Component框图

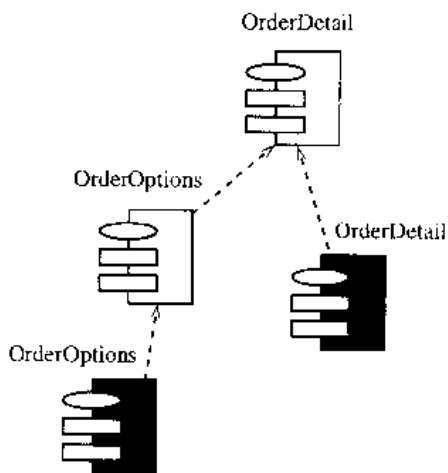


图9.8 边界包Component框图

图9.9显示系统中的所有组件。我们将这个框图称为System Component框图。有这个框图，就可以看出系统中所有组件之间的依赖性。

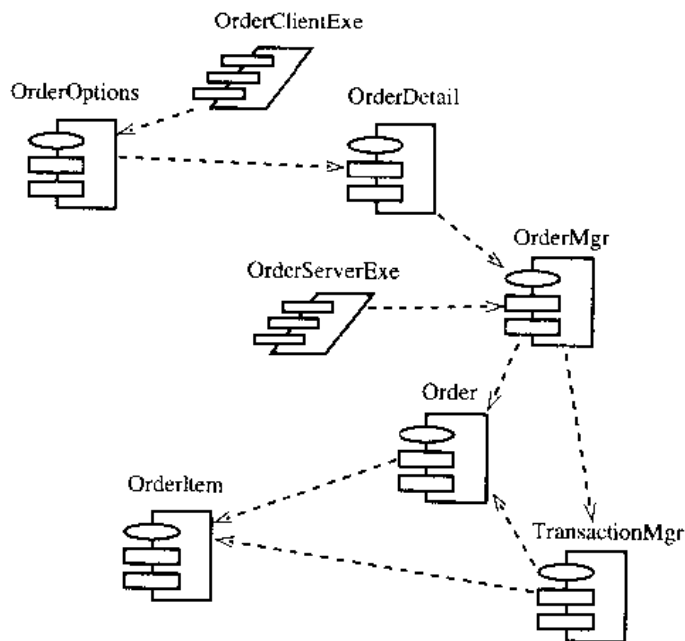


图9.9 订单处理系统的System Component框图

练习步骤

生成组件包

1. 右单击浏览器中的Component视图。
2. 选择New►Package。
3. 将新包命名为Entities。

4. 对Boundaries和Control包重复第1到第3步。

将包加进主Component框图

1. 双击打开主Component框图。

2. 将Entities、Boundary和Control包从浏览器拖动到主Component框图。

绘制包依赖性

1. 从工具箱中选择Dependency。

2. 在主Component框图中单击Boundaries包。

3. 画出向Control包的依赖性。

4. 重复1到3步增加从Control到Entities包的依赖性。

增加包的组件和画出依赖性

1. 双击主Component框图中的Entities包打开Entities包的主Component框图。

2. 从工具箱中选择Package Specification。

3. 将包规范放进框图中。

4. 输入包规范名为OrderItem。

5. 重复第2到第4步，加进Order包规范。

6. 从工具箱中选择Package Body。

7. 将包体放进框图中。

8. 输入包体名OrderItem。

9. 重复第6到第8步，加进Order包体。

10. 从工具箱中选择Dependency。

11. 单击OrderItem包体。

12. 拖动到OrderItem包规范的依赖性线。

13. 重复第10到第13步增加从Order包体到Order包规范的依赖性。

14. 重复第10到第13步增加从Order包规范OrderItem包规范的依赖性。

15. 用这个方法生成下列组件和依赖性。

对边界包：

- OrderOptions包规范
- OrderOptions包体
- OrderDetail包规范
- OrderDetail包体

边界包中的依赖性：

- OrderOptions包体到OrderOptions包规范
- OrderDetail包体到OrderDetail包规范
- OrderOptions包规范到OrderDetail包规范

对控制包：

- OrderMgr包规范
- OrderMgr包体

- TransactionMgr包规范
- TransactionMgr包体

控制包中的依赖性:

- OrderMgr包体到OrderMgr包规范
- TransactionMgr包体到TransactionMgr包规范
- OrderMgr包规范到TransactionMgr包规范

生成System Component框图

1. 右单击浏览器中的Component视图。
2. 从弹出菜单选择New►Component Diagram。
3. 将新框图命名为System。
4. 双击System Component框图。

将组件放进System Component框图

1. 如果需要, 在浏览器中展开实体组件包。
2. 单击实体组件包中的Order包规范。
3. 将Order包规范拖动到框图上。
4. 重复第2到第3步将OrderItem包规范放到框图上。
5. 用这个方法将下列组件放到框图上。

在边界组件包中:

- OrderOptions包规范
- OrderDetail包规范

在控制组件包中:

- OrderMgr包规范
- TransactionMgr包规范
- OrderClientExe任务规范
- OrderServerExe任务规范

6. 从工具箱中选择Task Specification。
7. 将Task Specification放在框图上, 并取名OrderClientExe。
8. 对OrderServerExe任务规范重复第6和第7步。

将其余依赖性加进System Component框图

加进组件后, 已经存在的依赖性自动出现在System Component框图中。然后可以将其余依赖性加进System Component框图:

1. 从工具箱中选择Dependency。
2. 单击OrderDetail包规范。
3. 向OrderMgr包规范拖动依赖性线。
4. 对下列依赖性重复第1到第3步:
 - OrderMgr包规范到Order包规范

- TransactionMgr包规范到OrderItem包规范
- TransactionMgr包规范到Order包规范
- OrderClientExe任务规范到OrderOptions包规范
- OrderServerExe任务规范到OrderMgr包规范

将类映射组件:

1. 在浏览器逻辑视图中, 找到实体包中的Order类。
2. 将Order类拖动到Component视图中的Order组件包规范中, 从而将Order类映射Order组件包规范。
3. 将Order类拖动到Component视图中的Order组件包体中, 从而将Order类映射Order组件包体
4. 重复第1到第3步将下列类映射组件:
 - OrderItem类映射OrderItem包规范
 - OrderItem类映射OrderItem包体
 - OrderOptions类映射OrderOptions包规范
 - OrderOptions类映射OrderOptions包体
 - OrderDetails类映射OrderDetails包规范
 - OrderDetails类映射OrderDetails包体
 - OrderMgr类映射OrderMgr包规范
 - OrderMgr类映射OrderMgr包体
 - TransactionMgr类映射TransactionMgr包规范
 - TransactionMgr类映射TransactionMgr包体

小结

本章介绍Rose的Component视图。Component视图主要关心系统的实际结构。组件就是与系统相关联的文件, 可以是源代码文件、执行文件或DLL文件。在Rose中, 可以用各种图标区分不同类型的组件。

类映射特定语言时, 首先映射组件。每个组件指定特定语言。在Rose企业版中, 可以用不同语言生成不同部件代码。

组件依赖性提供编译依赖性信息, 这种关系可以表示各个组件所需的编译顺序。

下一章介绍组件如何在网络上部署。

第10章 Deployment视图

- 生成和使用Deployment视图
- 增加处理器
- 增加设备
- 增加连接
- 增加处理器

本章介绍Rose的最后一个视图——Deployment视图。Deployment视图考虑应用程序的实际部署，包括网络布局 and 组件在网络上的位置。我们还要介绍部署问题，如具有多少网络带宽？希望出现多少并发用户？服务器关闭时怎么办？等等。

Deployment视图包含处理器、设备、进程和处理器与设备之间的连接。这一切都画在Deployment视图上。每个系统只有一个Deployment视图，因此每个Rose模型只有一个Deployment视图。

Deployment视图

Deployment视图显示网络上的所有节点、节点间的连接和每个节点上运行的进程。图10.1是个Deployment视图例子。

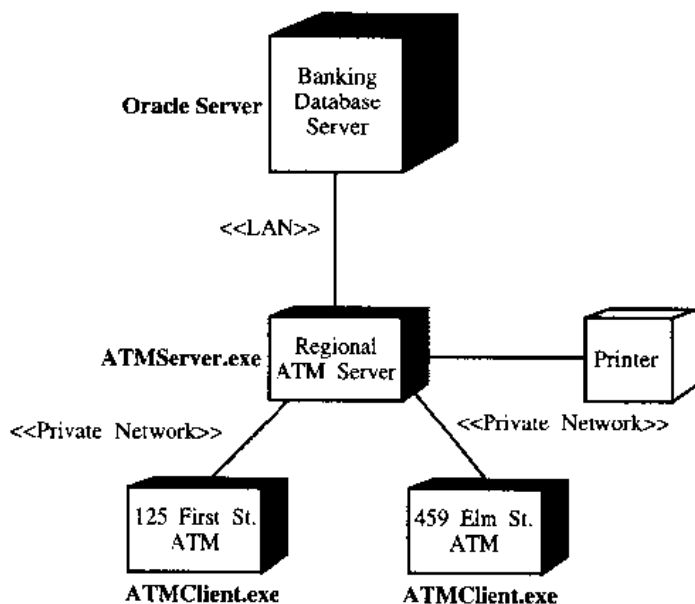


图10.1 ATM系统的Deployment视图

下面几节介绍Deployment视图的每个部分。

打开Deployment视图

Rose中Deployment视图是在Deployment视图中生成的。由于只有一个Deployment视图，因此不在浏览器中显示。要访问Deployment视图，可以双击浏览器中的Deployment view字样。

要打开Deployment视图：

1. 双击浏览器中的Deployment view字样。
2. Rose打开模型的Deployment视图

增加处理器

处理器是任何具有处理功能的机器。服务器、工作站和其他具有处理功能的机器都是处理器。

在UML中，处理器用下列符号显示：



要增加处理器：

1. 选择工具箱中的Processor。
2. 单击Deployment视图放上处理器。
3. 输入处理器名。

或

1. 选择Tools►视图Create►Processor。
2. 单击Deployment视图放上处理器。
3. 输入处理器名。

或

1. 右单击浏览器中的Deployment视图。
2. 从弹出菜单选择New►Processor。
3. 输入处理器名。

要将文档加进处理器中：

1. 右单击所要处理器。
2. 从弹出菜单选择Open Specification，打开处理器规范窗口。
3. 选择General标签。
4. 在Documentation字段中输入文档。

或

1. 双击所要处理器打开处理器规范窗口。
2. 选择General标签。
3. 在Documentation字段中输入文档。

或

1. 选择所要处理器。
2. 选择Browse►Specification。打开处理器规范窗口。

或

1. 选择所要处理器。
 2. 在文档窗口输入文档。
- 要删除框图中的处理器：

1. 选择框图中的处理器。
2. 按Delete。

或

1. 选择框图中的处理器。
2. 选择Edit►Delete。

说明：从框图中删除处理器后，它在浏览器中仍然存在。

要从模型中删除处理器：

1. 选择Deployment框图中的处理器。
2. 选择Edit►Delete from Model或按Ctrl-D。

或

1. 右单击浏览器中的处理器。
2. 从弹出菜单选择Delete。

说明：Rose从Deployment框图中从浏览器删除处理器。

增加处理器细节

在处理器规范中，可以增加处理器版型、特征和计划的信息。

和其他模型元素中一样，版型将处理器进行分类。例如，可能有Unix机和PC机，这两种机型可以分别定义不同版型。

处理器特征是处理器的物理描述。例如，可以包括处理器速度和内存量。

计划字段记录处理器使用的进程计划。选项包括：

Preemptive 表示高优先级进程可以抢先于低优先级进程。

Non Preemptive 表示进程没有优先级，当前进程或执行完毕之后再执行下一进程。

Cyclic 表示进程间的控制循环。每个进程有一定的执行时间，然后将控制传递给下一个进程。

Executive 表示用某种计算算法控制计划。

Manual 表示进程由用户进行计划。

要指定版型：

1. 打开所要处理器的规范窗口。
2. 选择General标签，如图10.2。
3. 在Stereotype字段中输入版型。

或

1. 选择所要处理器。
2. 在<<>>中输入版型：<<Name>>。

要将特性加进处理器中:

1. 打开所要处理器的规范窗口。
2. 选择Detail标签, 如图10.3。
3. 在Characteristics字段中输入特性。

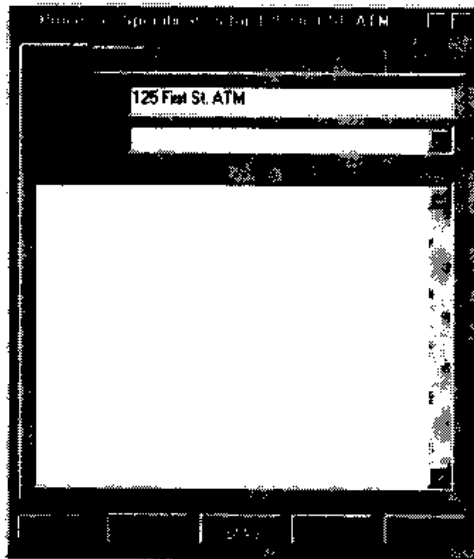


图10.2 输入处理器版型

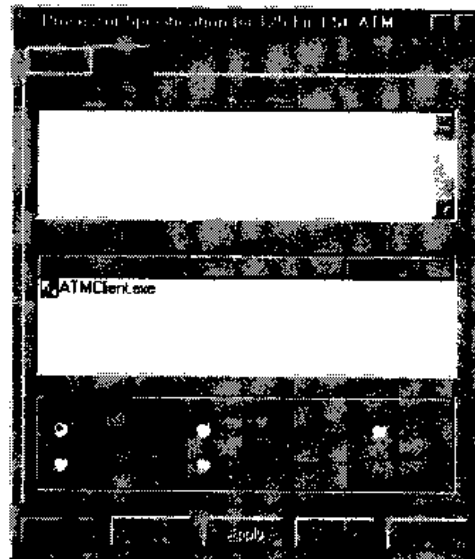


图10.3 输入处理器特性

要设置处理器计划:

1. 打开所要处理器的规范窗口。
2. 选择Detail标签。
3. 选择下列计划之一: preemptive、non-preemptive、cyclic、executive或manual。

要在框图上显示计划:

1. 右单击所要处理器。
2. 从弹出菜单选择Show Scheduling。

要在框图上显示处理进程:

1. 右单击所要处理器。
2. 从弹出菜单选择Show Processes。

增加设备

设备是没有处理功能的机器或硬件, 包括哑终端、打印机、扫描仪等。在UML中, 设备用下列符号表示:



处理器和设备都是网络上的节点。

要增加设备:

1. 选择工具箱中的Device。

2. 单击Deployment视图放上设备。

3. 输入设备名。

或

1. 选择Tools►Create►Device。

2. 单击Deployment视图放上设备。

3. 输入设备名。

或

1. 右单击浏览器中的Deployment视图。

2. 从弹出菜单选择New►Device。

3. 输入设备名。

要将文档加进设备中:

1. 右单击所要设备。

2. 从弹出菜单选择Open Specification, 打开设备规范窗口。

3. 选择General标签。

4. 在Documentation字段中输入文档。

或

1. 双击所要设备打开设备规范窗口。

2. 选择General标签。

3. 在Documentation字段中输入文档。

或

1. 选择所要设备。

2. 选择Browse►Specification。打开设备规范窗口。

或

1. 选择所要设备。

2. 在文档窗口输入文档。

要删除框图中的设备:

1. 选择框图中的设备。

2. 按Delete。

或

1. 选择框图中的设备。

2. 选择Edit►Delete。

说明: 从框图中删除设备后, 它在浏览器中仍然存在。

要从模型中删除设备:

1. 选择Deployment框图中的设备。

2. 选择Edit►Delete from Model或按Ctrl+D。

或

1. 右单击浏览器中的设备。

2. 从弹出菜单选择Delete。

说明: Rose从Deployment框图中和浏览器删除设备。

增加设备细节

和处理器一样,设备可以加进各种细节。第一是版型,用于将设备分类。第二是Characteristics字段。和处理器一样,设备的特征是设备的物理描述。

要指定版型:

1. 打开所要设备的规范窗口。
2. 选择General标签,如图10.4。
3. 在Stereotype字段中输入版型。

或

1. 选择所要设备。
2. 在<<>>中输入版型: <<Name>>。

要将特性加进设备中:

1. 打开所要设备的规范窗口。
2. 选择Detail标签,如图10.5。
3. 在Characteristics字段中输入特性。

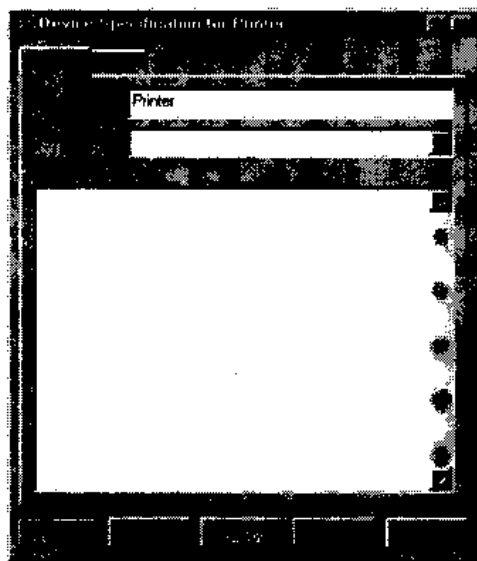


图10.4 输入设备版型

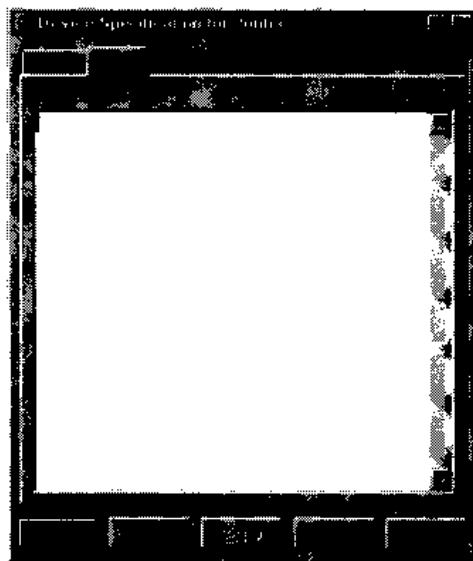


图10.5 输入设备特性

增加连接

连接是两个处理器、两个设备或处理器与设备之间的实际链接。通常,链接表示网络节点之间的物理网络连接。连接也可以表示两个节点间的Internet链接。

要增加连接:

1. 选择工具箱中的Connection。
2. 单击要连接的节点。
3. 向另一节点拖动连接线。

或

1. 选择**Tools>Create>Connection**。
2. 单击要连接的节点。
3. 向另一节点拖动连接线。

要删除连接：

1. 选择框图中的连接。
2. 按**Delete**。

或

1. 选择框图中的连接。
2. 选择**Edit>Delete**。

增加连接细节

连接可以指定版型。连接还可以有特性，提供物理连接的细节。例如，连接可能是T1线。这种说明加进**Characteristics**字段中。

要指定版型：

1. 打开所要连接的规范窗口。
2. 选择**General**标签，如图10.6。
3. 输入版型到**Stereotype**字段中。

或

1. 选择所要连接。
2. 输入版型到<<>>中：<<Name>>。

要将特性加进连接中：

1. 打开所要连接的规范窗口。
2. 选择**Detail**标签，如图10.7。
3. 在**Characteristics**字段中输入特性。

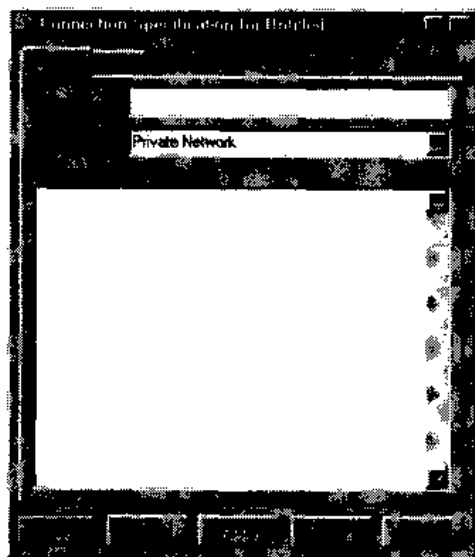


图10.6 输入连接版型

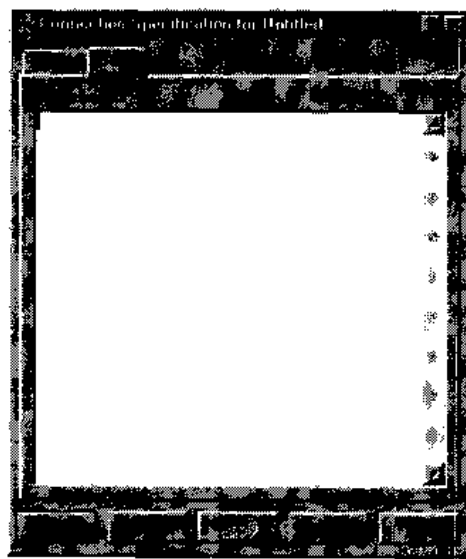


图10.7 将特性加进连接中

增加进程

增加进程是一个处理器上运行的单线程执行过程。例如执行文件是个进程。在框图中增加进程时，只考虑与所建系统有关的进程。

进程可以在**Deployment**框图中显示或隐藏。如果显示，则直接列在所运行的处理器下面。

进程可以指定优先级，如果所运行的处理器使用占先式计划，则进程优先级确定何时能运行。

要增加进程：

1. 右单击浏览器中所要的处理器。
2. 从弹出菜单中选择**New►Process**。
3. 输入新进程名。

或

1. 打开所要进程的规范窗口。
2. 选择**Detail**标签。
3. 右单击**Processes**框。
4. 从弹出菜单选择**Insert**。
5. 输入新进程名。

要将文档加进进程：

1. 打开所要进程的规范窗口。
2. 选择**Detail**标签。
3. 在**Documentation**字段中输入文档。

或

1. 双击浏览器中所要的处理器。
2. 选择**Detail**标签。
3. 在**Documentation**字段中输入文档。

或

1. 右单击浏览器中所要的处理器。
2. 从弹出菜单中选择**Open Specification**。
3. 选择**Detail**标签。
4. 在**Documentation**字段中输入文档。

要将优先级加进进程：

1. 打开所要进程的规范窗口。
2. 选择**General**标签，如图10.8。
3. 在**priority**字段中输入优先级。

要删除进程：

1. 右单击浏览器中所要的处理器。
2. 从弹出菜单中选择**Delete**。

或

1. 打开所要进程的规范窗口。
2. 选择**Detail**标签。

3. 右单击所要进程。
4. 从弹出菜单选择Delete。

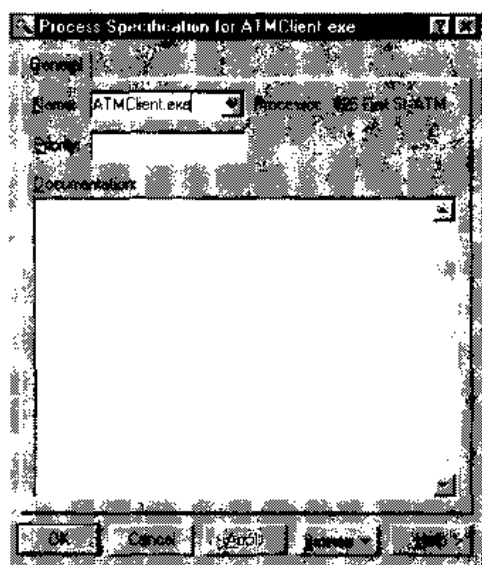


图10.8 输入进程信息

练习

本练习对订单处理系统生成Deployment框图。

问题

项目小组已经完成了大量分析和设计工作，使用案例、对象交互和组件已经定义完毕，但网络管理部门还要了解组件在哪个机器上。因此，小组要建立订单处理系统的Deployment框图。

生成Deployment框图

建立订单处理系统的Deployment框图。完成的框图如图10.9所示。

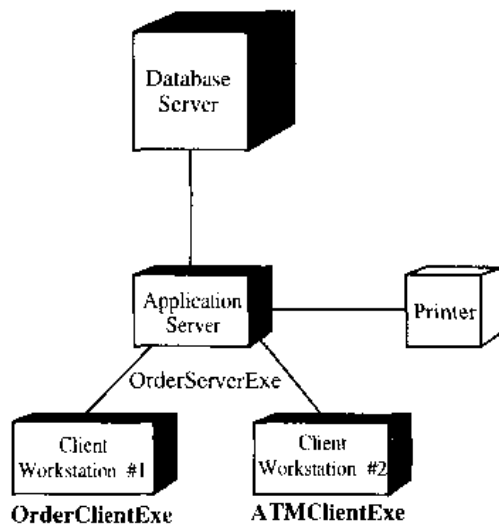


图10.9 订单处理系统Deployment框图

练习步骤

将节点加进Deployment框图

1. 双击浏览器中的Deployment视图打开Deployment框图。
2. 从工具箱中选择Processor。
3. 单击框图放上处理器。
4. 输入处理器名Database Server。
5. 重复第2到第4步加入下列处理器：
 - Application Server
 - Client Workstation #1
 - Client Workstation #2
6. 从工具箱中选择Device。
7. 单击框图放上设备。
8. 输入设备名Printer。

增加连接

1. 从工具箱中选择Connection。
2. 单击Database Server处理器。
3. 拖动连接线到Application Server处理器。
4. 重复第1到第3步加进下列连接：
 - Application Server处理器到Client Workstation #1处理器
 - Application Server处理器到Client Workstation #2处理器
 - Application Server处理器到Printer设备。

增加进程

1. 右单击浏览器中的Application Server处理器。
2. 从菜单选择New►Process
3. 输入进程名OrderServerExe。
4. 重复第1到第3步增加下列进程：
 - 对Client Workstation #1处理器：OrderClientExe
 - 对Client Workstation #2处理器：OrderClientExe

显示进程

1. 右单击浏览器中的Application Server处理器。
 2. 从菜单选择Show Processes。
 3. 重复第1和第2步显示下列处理器的进程：
 - Client Workstation #1处理器
 - Client Workstation #2处理器
-

小结

本章介绍Rose的Deployment视图。在Deployment框图中，项目小组描述网络结构和各个进程在哪里运行。这样就有了完成下列任务所需的信息：

- 用使用案例和角色定义系统范围，并在Use Case框图上画出使用案例和角色。
- 用使用案例和Use Case文档分析问题。
- 在Sequence或Collaboration框图中描述系统中的文档和系统流程。
- 生成实现事件流中的功能所需类，并画出类。
- 定义和画出类的属性、关系和操作。
- 生成State Transition框图，检查类的动态行为。
- 进行结构评估，方法是将类组成包，并检查类和包之间的关系。
- 在Processor框图中定义和浏览系统实际结构。
- 在Deployment框图中浏览网络结构和部署信息。

下一部分介绍Rose的代码生成特性。下面几章介绍如何从Rose模型生成C++、Java和PowerBuilder代码。我们还要介绍如何用Rose定义Oracle8结构，如何生成Data Definition Language (DDL) 脚本和Interface Definition Language (IDL) 文件。

第11章 用Rational Rose生成代码简介

- 用Rational Rose生成代码
- 检查Rose模型
- 设置代码生成属性
- 选择类、组件和包

Rational Rose最强大的特性之一是能够生成表示模型的代码。本章介绍从Rose模型生成代码之前要完成的基本步骤。下面几章介绍生成C++、Java、PowerBuilder和Visual Basic代码的具体步骤。我们还介绍从Rose模型生成IDL、DDL和Oracle8结构。

代码生成选项随Rose版本不同而不同，Rose 98与98i之间也有所有不同：

- Rose Modeler可以生成系统的模型，但不支持代码生成和逆向转出工程代码。
- Rose Professional可以用一种语言生成代码。
- Rose Enterprise可以用C++、Java、Visual Basic、Oracle8结构和其他语言生成代码。

许多Rose伙伴公司已经开发出支持其他语言代码生成和逆向转出工程代码的插件件，详见Rational的Web站点www.rational.com。

准备生成代码

生成代码的基本步骤有六步：

1. 检查模型
2. 生成组件
3. 将类映射组件
4. 设置代码生成属性
5. 选择类、组件和包
6. 生成代码

并不是每种语言都需要这些步骤。例如，生成C++代码时不需要先生成组件。不运行Check Model步骤可以在任何语言中生成代码，但代码生成期间可能出错。下面几章介绍各种语言中代码生成的细节。

尽管并非总是需要这些步骤，但建议先完成前五步再生成代码。模型检查有助于发现模型中的问题和不一致性，以免影响代码。组件步骤可以将逻辑系统设计映射实际实现方法，并提供大量有用信息。如果跳过这些步骤，则Rose用逻辑视图中的包结构生成组件。

第一步：检查模型

Rose包括独立于语言的模型检查特性，可以在生成代码之前保证模型一致性。最好先进行这个检查再生成代码。

要检查Rose模型:

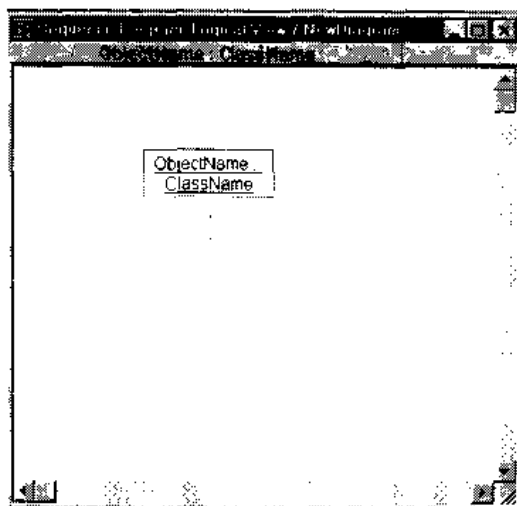
1. 从菜单中选择**Tools►Check Model**。
2. 发现的错误写入日志窗口。

常见错误包括Sequence或Collaboration框图中的消息不映射操作, 或Sequence或Collaboration框图中的对象不映射类等等。下面是一些常见错误及解决方案。

下列消息表示Sequence或Collaboration框图中的对象不映射类:

```
Unresolved reference from use case "<Use case name>" to ClassItem with
name (Unspecified) by object <Object name>
```

看看Sequence或Collaboration框图中的对象。每个框应包含对象名加冒号加类名如下:



寻找不带类名的对象。右击对象并选择弹出菜单中的**Open Specification**。在对象规范窗口中, 用**Class**下拉列表框选择对象的类。

下列消息表示Sequence或Collaboration框图中的消息不映射操作:

```
Unresolved reference to Operation with name <Message name> in message:
<Message name> between <Class name> and <Class name> in Sequence dia-
gram <Use case name>/<Sequence diagram name>
```

右击框图中的相应消息(日志窗口显示错误消息名及其**Open Specification**框图), 并将消息映射操作。如果需要, 对消息生成新操作。

访问问题

Check Model菜单项目可以找出模型中的大多数问题和不一致性。**Access Violations**菜单项目寻找不同包中的两个类之间存在关系时发生的问题。例如, 如果实体包中的**Order**类与控制包中的**Access Violations**类有关系, 则实体包和控制包之间必须有关系, 否则Rose会发现访问问题。

要发现访问问题:

1. 从菜单中选择**Report►Show Access Violations**。
2. Rose在窗口中显示访问问题, 如图11.1。

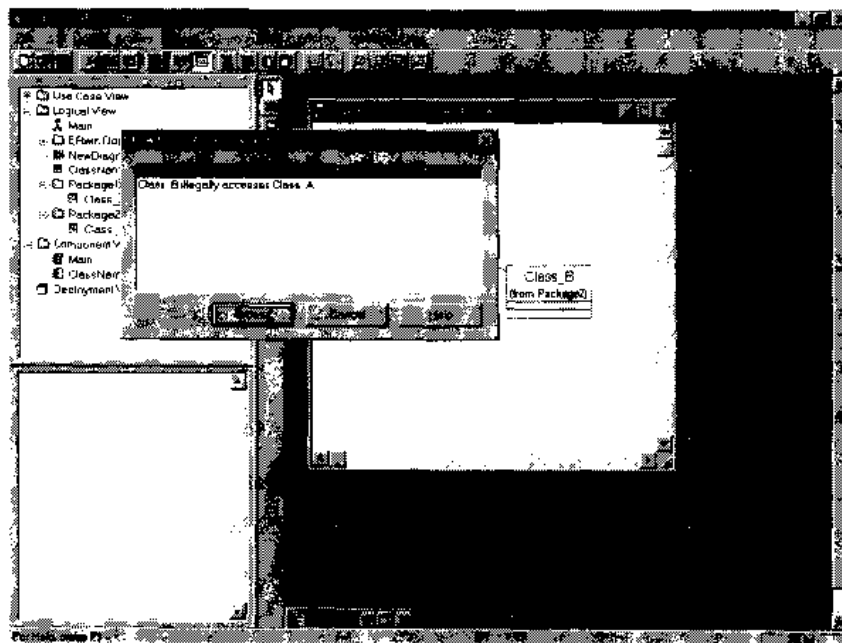


图11.1 发现访问问题

语言特定检查

要进行语言特定检查，选择Tools►Check Model。如果使用Java，则可以选择Tools►Java►Syntax Check进行Java语法检查。这个检查可以找出“一个编译单元中的多个公开类”之类的错误。

第二步：生成组件

代码生成过程的第二步是生成保存类的组件。组件的种类很多：源代码文件、执行文件、运行库、ActiveX组件和小程序都是组件。生成代码之前，可以先将类映射源代码组件。

生成组件后，可以在组件框图上加进组件之间的依赖性。组件之间的依赖性系统的编译依赖性（组件和组件之间的依赖性见第9章）。

如果生成C++、Java或Visual Basic，则不需要完成这个步骤。在Java或Visual Basic中，Rose自动生成每个类的相应组件，但不生成组件之间的依赖性。

要生成组件：

1. 打开Component框图。
2. 用框图工具栏中的Component图标将新组件加进框图中。

第三步：将类映射组件

每个源代码组件表示一个或几个类的源代码文件。例如，在C++中，每个类映射两个源代码组件，一个表示头文件，一个表示体文件。在PowerBuilder中，许多类映射单个组件。PowerBuilder源代码组件是PowerBuilder库（.PBL）文件，Java中每个源代码组件表示一个JAVA文件。

代码生成过程的第三步是将类映射组件。对PowerBuilder，要先将类映射组件之后才能生成代码。但C++、Java或Visual Basic中则可选可不选。Rose不用这个步骤也能生成代码。

如果生成Java或Visual Basic代码, 则Rose自动生成相应组件并将类映射组件。但C++中则并不自动生成组件。任何语言均不生成组件依赖性。因此, 最好不管用什么语言都先完成这个步骤。

要将类映射组件:

1. 右击Component框图或浏览器中的组件。
2. 从弹出菜单选择Open Specification。
3. 选择Realizes标签。
4. 在Realizes标签中右单击相应类, 并从弹出菜单选择Assign。
5. 浏览器在Logical视图的类名后面括号中显示组件名。

或者

1. 在浏览器Logical视图中找到所要类。
2. 将类拖动到在Component视图中的相应组件。
3. 组件名在Logical视图的类名后面括号中显示。

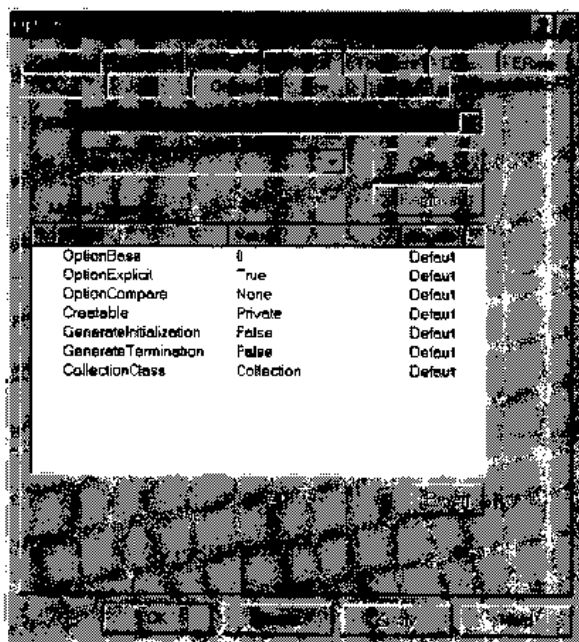
第四步: 设置代码生成属性

类、属性、组件和其他模型元素可以设置多个代码生成属性。这些属性控制代码如何生成。Rose提供常用缺省设置。

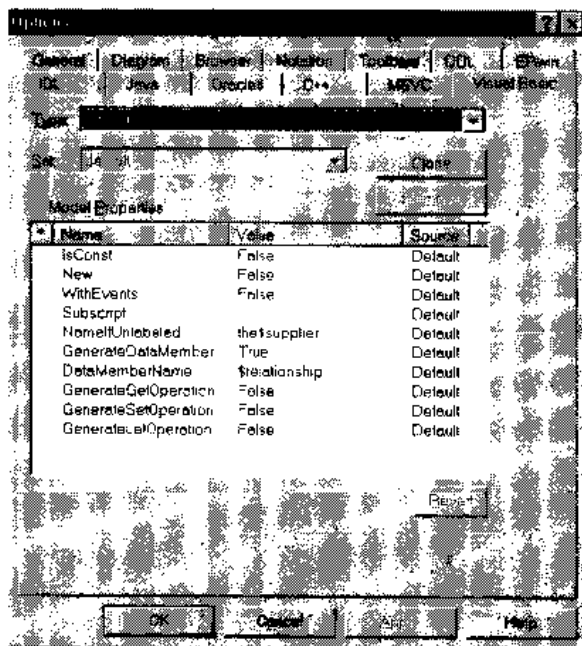
例如, C++属性的一个代码生成属性是GenerateGetOperation, 控制该属性是否生成Get()操作。Java类的一个代码生成属性是GenerateDefaultConstructor, 控制类是否自动生成构造器。Visual Basic中关系的一个属性是GenerateDataMember, 控制是否自动生成支持该关系的属性。

Rose中的每种语言都有几个代码生成属性。下面几章将介绍每种语言的代码生成属性。生成代码之前, 最好先检查代码生成属性并作出必要的改变。

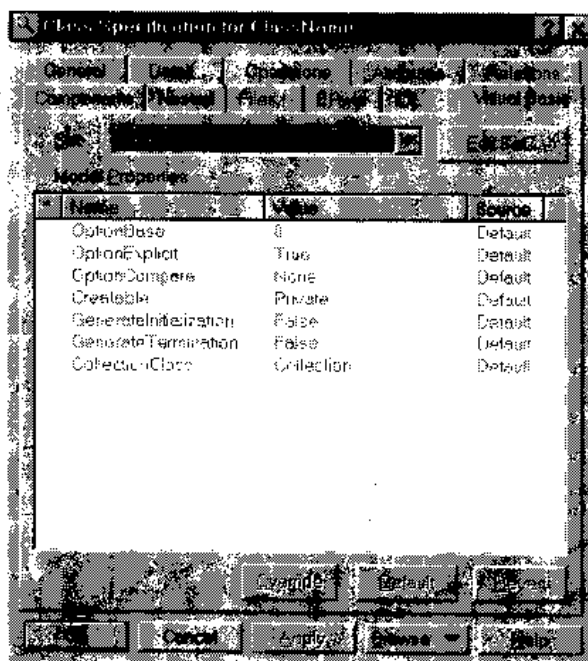
要浏览代码生成属性, 选择Tools►Options, 然后选择相应语言标签, 例如, 下面是Visual Basic属性的标签。



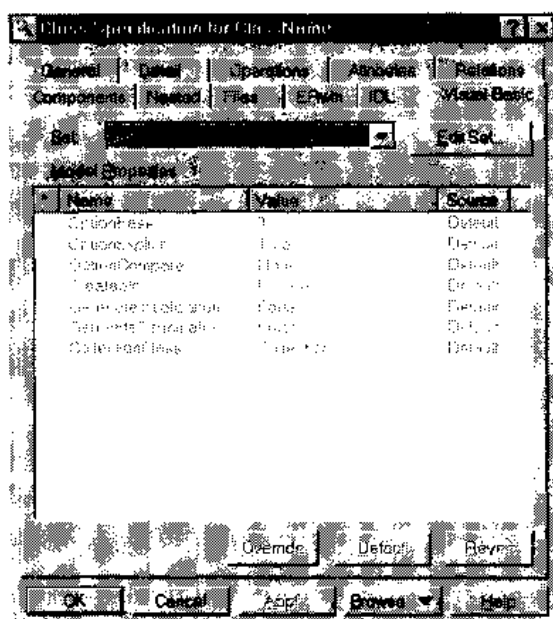
从下拉列表框中可以选择Class、Attribute、Operation或其他类型的模型元素。每种语言的下拉列表框有不同的模型元素。选择不同数值时，出现不同的属性设置。前面介绍了Visual Basic中的Class属性，下面是Attribute属性：



Tools►Options窗口中对属性设置的改变会影响使用该设置的所有模型。例如，如果改变Java标签中的Generate DefaultConstructor类属性，则会影响模型中用Java实现的所有类。

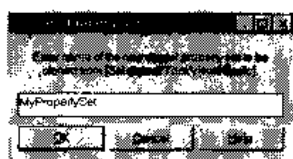


有时要改变单个类、属性、操作或模型元素的代码生成属性。为此，打开该模型元素的规范窗口，选择语言标签（C++、Java、Visual Basic或PowerBuilder）并在此改变属性。模型元素的规范窗口所作的任何改变都只影响这个模型元素。



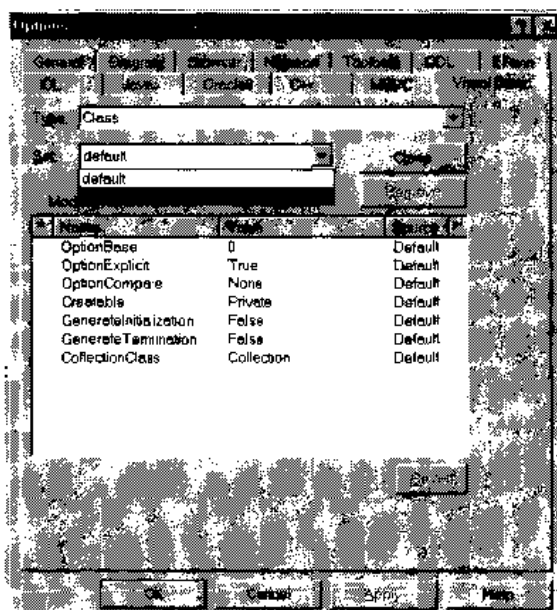
克隆属性设置

与其直接改变缺省设置值，不如先克隆属性设置，再对备份进行更改。要克隆属性设置，按Clone the Property Set窗口的Clone按钮。Rose提示输入新属性设置名。



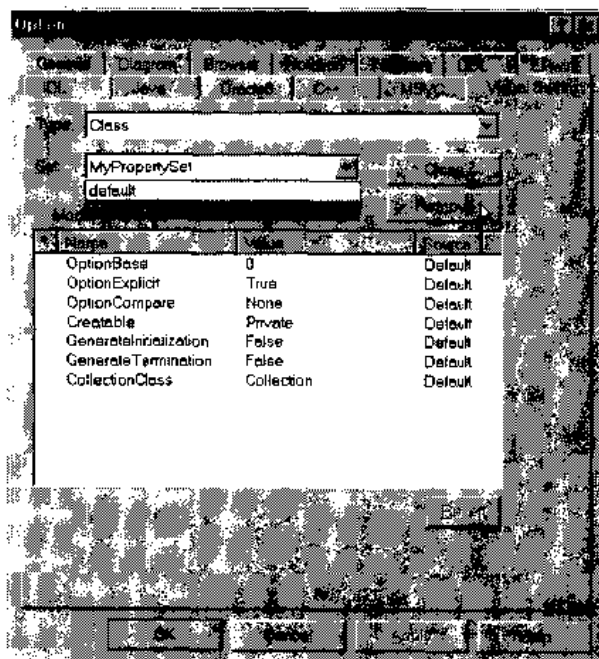
克隆属性设置后，可以打开Clone the Property Set窗口的Set下拉列表中框显示。

这个克隆属性设置可以进行任何改变，不会影响原有缺省设置。建议保持原有缺省设置，只改变克隆属性设置。



删除属性设置

如果一个克隆属性设置不再需要，可以用**Tools**►**Options**窗口从模型中删除。选择相应语言标签，然后从**Set**下拉列表框中选择克隆属性设置。



选择相应设置后，按**Remove**按钮。Rose从模型中删除这个设置。注意，原有缺省设置无法删除。

第五步：选择类、组件或包

生成代码时，可以一次生成一个类、一个组件或一个包。代码可以从框图或浏览器中删除。如果从包生成代码，则可以选择**Class**框图中的**Logical**视图包或**Component**框图中的**Component**视图包。如果选择**Logical**视图，则生成这个包中的所有类。如果选择**Component**视图，则生成这个包中的所有类。

也可以一次对多个类、组件或包生成代码。在框图中按**Ctrl**键选择要生成代码的类、组件或包，然后从菜单中选择相应的代码生成命令。

第六步：生成代码

如果安装了Rose Professional或Enterprise，则**Tools**菜单中有一些语言特定菜单选项，如图11.2。

要显示或隐藏这些代码生成菜单选项，选择**Add-Ins**►**Add-In Manager**。在**Add-In Manager**对话框中（如图11.3）用复选框显示或隐藏各个代码生成菜单选项。

框图中选择类或组件后，从菜单中选择相应代码生成选项。如果代码生成过程中遇到错误，则这些错误在日志窗口中标出。

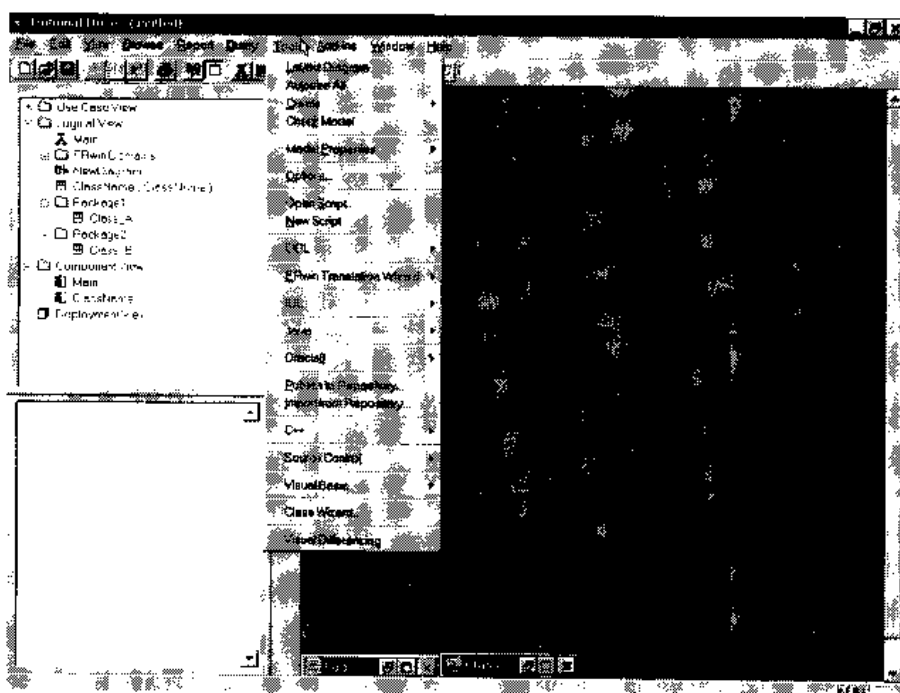


图11.2 代码生成器单选项

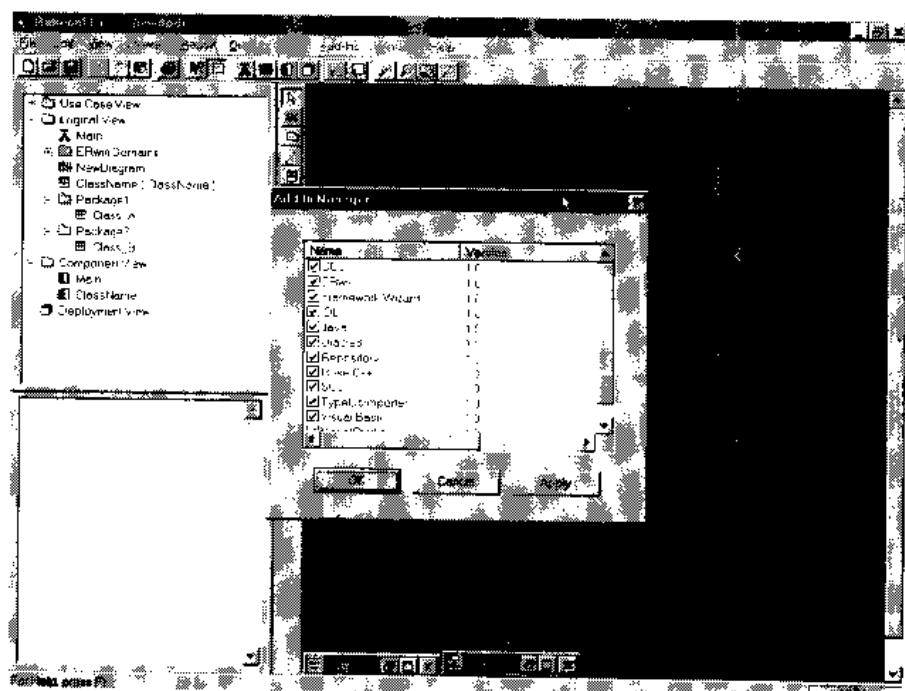


Figure 11.3 Add-In Manager

生成什么

生成代码时，Rose从Logical视图和Component视图中收集信息。尽管任何造型工具都无法生成完整应用程序，但Rose能生成大量框架代码，如下：

类 (Classes) 代码中生成模型中的所有类。

属性 (Attributes) 代码包括每个类的属性, 包括可见性、数据类型和缺省值。

操作签名 (Operation signatures) 代码中声明操作及其参数, 参数数据类型和返回值。

关系 (Relationships) 模型中的有些关系会使属性在代码生成时产生。

组件 (Components) 每个组件由相应源代码文件实现。

生成文件后, 还有两个步骤。第一, 开发人员利用这个文件编码类的每个操作。第二, 设计图形用户界面。

Rose不是GUI设计工具, 你可以用编程语言的环境设计屏幕和窗体。这个方法可以保证所建系统的设计良好。开发小组可以通过检查Rose模型而确定最佳结构和最佳设计, 然后从这个模型生成代码。这样就不是20个编程人员往20个不同方向设计, 而是都按同一个蓝图工作。

生成代码时, Rose用Component视图中建立的包结构生成相应目录。缺省情况下, 生成代码的根目录为Rose应用程序文件所在的目录。可以通过特定语言的代码生成选项改变目录。

如果没有建立组件, 则Rose用Logical视图中建立的包结构生成目录。同样, 缺省目录为Rose目录, 但可以通过特定语言的代码生成选项改变目录。

小结

本章介绍了Rose代码生成功能的概况。下面几章介绍从C++、Java、Visual Basic和PowerBuilder生成的特定代码, 还要介绍Rose模型可以生成的Interface Definition Language (IDL) 和Data Definition Language (DDL), 并介绍如何生成Oracle8结构。

复习一下, 代码生成步骤如下:

1. 检查模型。
2. 生成组件。
3. 将类映射组件。
4. 设置代码生成属性。
5. 选择类、组件和包。
6. 生成代码。

第12章 C++与Visual C++代码生成

- 设置C++代码生成属性
- 从Rose模型生成C++代码
- 将Rose元素映射到C++结构

C++是业界最广泛使用的面向对象语言之一。Rational Rose支持通过代码生成和逆向功能集成C++。本章介绍如何从Rational Rose模型生成C++代码。

Rose 98集成C++，Rose 98i提供标准C++代码生成，并与Microsoft的Visual C++ 6紧密集成。本章介绍C++和VC++中的代码生成过程。

在标准C++中生成代码的步骤如下：

1. 生成组件（可选，见第10章）。
2. 将类赋予组件（可选，见第10章）。
3. 选择代码生成属性（可选）。
4. 选择Class或Component框图中要生成的类和组件。
5. 选择Tools>C++>Code Generation。
6. 选择Tools>C++>Browse Header或Browse Body浏览生成的代码。

生成代码的第一步是生成组件的类。这些组件是代码的.CPP和.H文件。在C++中，这个步骤不是必需的。如果没有定义组件，则Rose对每个类生成.CPP和.H文件。但建议生成组件以控制类与组件间的映射和建模型件依赖性。

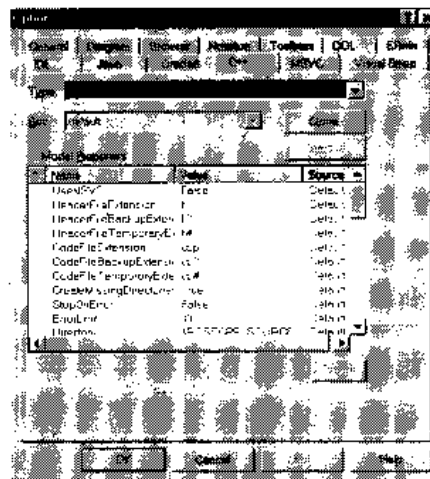
生成组件和映射类之后，下一步要设置类、组件、操作和其他模型元素的代码生成属性。代码生成属性控制所生成代码的某些方面。本章介绍可以设置的代码生成属性，并介绍每个Rose模型元素如何在代码中实现。

如果在Rose 98i中用Visual C++生成代码，则使用向导。要启动向导，选择Tools>Visual C++>Update Code启动Visual C++ Code Update工具，出现欢迎屏幕。单击Next按钮继续。Rose显示Select Components和Classes窗口。在用Visual C++生成代码之前，必须将类赋予组件。如果没有将类赋予组件，选择向导窗口中的Create a VC++ Component and Assign New Classes to It (Ctrl+R) 选项。利用这个选项，可以先生成多个需要的组件，然后再生成代码。然后选择模型中要生成代码的组件。

要改变Visual C++组件和类的代码生成属性，右单击这个屏幕上的VC++文件夹。然后可以编辑各个代码生成属性如容器类，支持关系倍增性，是否自动生成Get与Set操作或其他成员函数。代码生成属性将在“C++代码生成属性”中详细介绍。

将所有类赋予组件后，选择要生成的所有类和组件，设置所有代码生成属性，然后单击Next按钮继续，小结页面会显示生成哪些类和组件，代码生成过程中遇到哪些错误。

Rose从模型中利用大量信息生成代码。例如，它要寻找每个关系的倍增性、作用名、



本节的选项包括生成代码时使用的缺省目录、使用的文件扩展、代码生成期间可以出现的最大错误数等。表12.1列出了最常用的项目属性及其用途和缺省值。

本节和下面各节用表格列出最常用的C++属性。关于所有C++代码生成属性的完整清单，见Rose联机帮助。

表12.1 项目代码生成属性

属性	用途	缺省值
UseMSVC	控制是否生成MSVC源代码	False
HeaderFileExtension	控制生成头文件的文件扩展名	h
HeaderFileBackupExtension	控制生成文件覆盖现有文件时备份头文件的文件扩展名	h~
HeaderFileTemporaryExtension	设置代码生成期间临时使用的文件扩展名	h#
CodeFileBackupExtension	控制生成文件覆盖现有文件时备份实现文件的文件扩展名	cp~
CodeFileExtension	控制生成实现文件的文件扩展名	cpp
CodeFileTemporaryExtension	设置代码生成期间临时使用的文件扩展名	cp#
CreateMissingDirectories	控制Rose生成代码时是生成目录还是映射到包	True
StopOnError	控制Rose遇到错误时是否停止代码生成	False
ErrorLimit	设置Rose停止代码生成之前可以出现的最大错误数	30
Directory	设置代码生成的根目录，所有目录和C++文件均在其中生成	缺省情况下，Rose用路径映象中带符号\$ROS-ECPP_SOURCE的目录
PathSeparator	设置#include语句中使用的路径分隔符（如斜杠或反斜杠）	Blank（空白）
FileNameFormat	控制Rose自动生成文件名的格式	128vx_b（最多128个字符，包括元音、保留大小写、保留下划线）
BooleanType	设置模型中布尔数据类型所用的数据类型	Int
AllowTemplates	控制Rose是否对模型中的参数化类生成模板	True
AllowProtectedInheritance	控制代码中是否生成保护继承物	False

(续表)

属性	用途	缺省值
CommentWidth	标量一行说明语句中的最大字符量	60
AllowExplicitInstantiations	控制是否生成参数化类显式实例	False
AlwaysKeepOrphanedCode	控制保护区域中是否放上孤代码	False

此外,有些属性涉及关系中的倍增性。生成代码时,Rose自动生成某些关系类型的属性。例如,如果类Client和Supplier之间具有单向关联,而关系的倍增性为一对一,则Rose在Client内生成Supplier类型的属性。

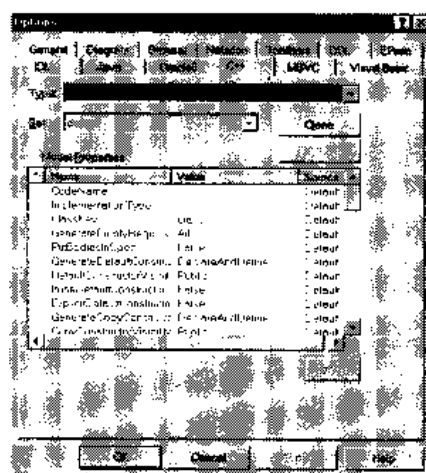
如果倍增性大于一,则Rose在生成属性时使用清单之类的容器类。不同关系和不同倍增性使用不同容器类。缺省容器类在项目的C++代码生成属性中设置。要改变缺省,可以改变这些属性中指定的容器类。

下列属性提供缺省值,它们只用于没有设置关系的ContainerClass代码生成属性时。关系的ContainerClass属性指定生成特定关系的属性时使用的容器类。

OneByValueContainer
 OneByReferenceContainer
 OptionalByValueContainer
 OptionalByReferenceContainer
 FixedByValueContainer
 UnorderedFixedByValueContainer
 FixedByReferenceContainer
 UnorderedFixedByReferenceContainer
 BoundedByValueContainer
 UnorderedBoundedByValueContainer
 BoundedByReferenceContainer
 UnorderedBoundedByReferenceContainer
 UnboundedByValueContainer
 UnorderedUnboundedByValueContainer
 UnboundedByReferenceContainer
 UnorderedUnboundedByReferenceContainer
 QualifiedByValueContainer
 UnorderedQualifiedByValueContainer
 QualifiedByReferenceContainer
 UnorderedQualifiedByReferenceContainer

类属性

类属性是适用于类的C++代码生成属性。这些属性可以改变类名、确定是否生成类的构造器与删除器并设置其他类特定属性。



这些属性可以在两个地方设置。要设置所有类的属性，选择Tools►Options，然后选择C++标签，并从下拉列表框中选择Class。要设置一个类的属性，在类规范窗口中选择C++标签并在此编辑属性。

表12.2列出了许多C++类属性、用途和缺省值。

表12.2 类代码生成属性

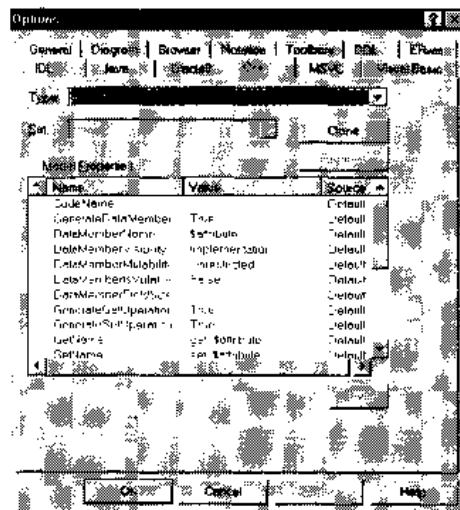
属性	用途	缺省值
CodeName	生成代码中的类名	缺省时，Rose在模型中使用类名
ImplementationType	控制类生成时用类定义还是用基本数据类型	Blank
ClassKey	控制类定义中是否使用关键字类、结构或联合	Class
GenerateEmptyRegions	控制是否生成空区域，是否在逆向转出工程代码中保留	All
PutBodiesInSpec	控制类实现细节是否在头文件中生成	False
GenerateDefaultConstructor	控制是否自动生成类的构造器	True
DefaultConstructorVisibility	设置构造器的可见性（public、private、protected）	Public
InlineDefaultConstructor	控制缺省构造器是否内联	False
ExplicitDefaultConstructor	控制缺省构造器是否使用显式关键字	False
GenerateCopyConstructor	控制是否自动生成类的备份构造器	True
CopyConstructorVisibility	设置备份构造器的可见性（public、private、protected）	Public
InlineCopyConstructor	控制备份构造器是否内联	False
ExplicitCopyConstructor	控制备份构造器是否使用显式关键字	False
GenerateDestructor	控制是否自动生成类的删除器	True
DestructorVisibility	设置删除器的可见性（public、private、protected）	Public
DestructorKind	控制删除器中是否使用虚拟关键字	Common (False)
InlineDestructor	控制删除器是否内联	False
GenerateAssignmentOperation	控制是否生成赋值操作（=）和生成框架定义	DeclareAndDefine （生成操作和框架）
AssignmentVisibility	设置操作的可见性（public、private、protected）	Public
AssignmentKind	控制是否使用虚拟关键字	Common (False)
InlineAssignmentOperation	控制赋值操作是否内联	False

(续表)

属性	用途	缺省值
GenerateEqualityOperations	控制是否生成等于操作(==和!=)和生成框架定义	DeclareAndDefine (生成操作和框架)
EqualityVisibility	设置操作的可见性(public, private, protected)	Public
EqualityKind	控制是否使用虚拟关键字	Common (False)
InlineEqualityOperation	控制等于操作是否内联	False
GenerateRelationalOperations	控制是否生成关系操作(<, >, <=和>=)和生成框架定义	False
RelationalVisibility	设置操作的可见性(public, private, protected)	Public
RelatioalKind	控制是否使用虚拟关键字	Common (False)
InlineRelationalOperations	控制关系操作是否内联	False
GenerateStorageMgmtOperations	控制是否生成new和delete操作	False
StorageMgmtVisibility	设置操作的可见性(public, private, protected)	Public
InlineStorageMgmtoperations	控制关系操作是否内联	False
GenerateSubscriptOperation	控制是否生成下标函数([])	False
SubscriptVisibility	设置操作的可见性(public, private, protected)	Public
SubscriptKind	控制操作中是否使用虚拟关键字	Common (False)
SubscriptResultType	设置函数返回类型	Blank (Void)
InlineSubscriptOperation	确定操作是否内联	False
GenerateDereferenceOperation	控制是否生成删除引用函数(*)	False
DereferenceVisibility	设置操作的可见性(public, private, protected)	Public
DereferenceKind	控制操作中是否使用虚拟关键字	Common (False)
DereferenceResultType	设置函数返回类型	Blank (Void)
InlineDereferenceOperation	确定操作是否内联	False
GenerateIndirectionOperation	控制是否生成间接函数(→)	False
IndirectionVisibility	设置操作的可见性(public, private, protected)	Public
IndirectionKind	控制操作中是否使用虚拟关键字	Common (False)
IndirectionResultType	设置函数返回类型	Blank (Void)
InlineIndirectionOperatin	确定操作是否内联	False
GenerateStreamOperations	控制是否生成流操作(<《和》>)	False
StreamVisibility	设置操作的可见性(public, private, protected)	Public
InlineStreamOperations	控制操作是否内联	False

属性的属性

属性的属性是与属性相关的C++特定属性。利用这些属性可以确定代码中是否生成属性、代码中用什么属性名、属性是否生成Get和Set操作等。



设置这些属性的地方有两个要对所有属性进行设置，选择Tools►Options，然后选择C++标签，并从下拉列表框选择Attribute。要对一个属性进行设置，选择属性规范窗口的C++标签，并在此进行编辑。

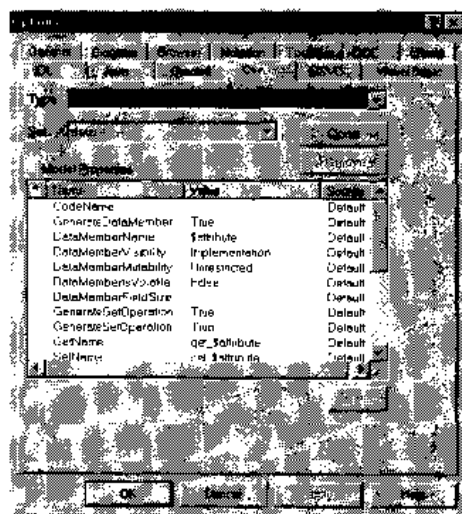
表12.3列出了属性的属性、用途和缺省值。

表12.3 属性代码生成属性

属性	用途	缺省值
CodeName	设置生成代码中的属性名	Blank (用模型中的属性名)
GenerateDataMember	控制是否生成属性的成员变量	True
DataMemberName	设置生成属性的成员变量名	缺省用模型中的属性名
DataMemberVisibility	设置生成成员变量的可见性	Private
DataMemberMutability	控制属性用mutable关键字、const关键字或别的方法生成	Neither (不限制)
DataMemberIsVolatile	控制属性是否用V关键字生成	False
DataMemberFieldSize	指定生成属性的位数	Blank
GenerateGetOperation	控制是否生成属性的Get操作，帮助其他类取得属性值	True
GenerateSetOperation	控制是否生成属性的Set操作，帮助其他类设置属性值	True
GetName	设置生成的Get操作名	get_<attribute name>
SetName	设置生成的Set操作名	set_<attribute name>
GetSetKinds	控制Get和Set操作为虚拟、静态、抽象、朋友或其他	None (Common)
GetIsConst	控制Get操作是否用const关键字	True
GetResultsIsConst	控制Get操作是否返回const值	GetIsConst (Same As Function中设置的值)
GetSetByReference	控制Get和Set操作按引用或按值传递变元和返回值	By value (按值)
InlineGet	确定Get操作是否内联	True
SetReturnsValue	指定Set操作是否返回非空	False
InlineSet	确定Set操作是否内联	True

操作属性

操作属性是针对操作的C++代码生成属性。这些属性可以设置操作名、控制操作是否虚拟和设置操作的其他代码生成属性。



设置这些属性的地方有两处。要设置所有操作的操作属性，选择Tools►Options，然后选择C++标签，并从下拉列表框中选择Operation。要设置某个操作的操作属性，在规范窗口中选择C++标签，并编辑属性。

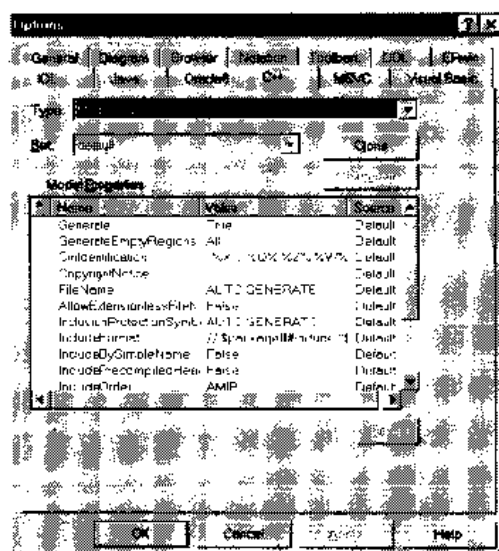
表12.4列出了许多操作属性及其用途和缺省值。

表12.4 操作代码生成属性

属性	用途	缺省值
CodeName	设置生成的操作名	模型中的操作名
OperationKind	控制操作为虚拟、静态、抽象、朋友或其他	None (Common)
OperationIsConst	控制操作是否用Const关键字生成	False
OperationIsExplicit	控制生成操作时是否使用关键字explicit	False
Inline	控制是否内联操作	False
EntryCode	用于输入操作的代码或说明语句（这个代码不放在保护区域中，否则会在逆向转出工程代码中被保护）	Empty
ExitCode	用于输入操作的代码或说明语句（这个代码不放在保护区域中，否则会在逆向转出工程代码中被保护）	Empty
GenerateEmptyRegions	控制操作是否生成保护区	总是生成（ALL）
BodyAnnotations	控制实现文件中是否生成操作文档、开发性、空间、时间、限定、异常、前提条件、事后条件和词法的图注	全部生成

模块规范（头文件）属性

模块规范属性是与Rose所生成头文件（H）相关的属性。这些属性可以确定是否生成头文件，在文件中包括版权说明，改变头文件名或控制头文件的其他规范。



这些属性可以在两个地方设置。要设置所有头文件的属性，选择Tools►Options，然后选择C++标签，并从下拉列表框选择Module Specification。要设置一个文件的属性，在组件规范窗口中选择C++标签并编辑属性。

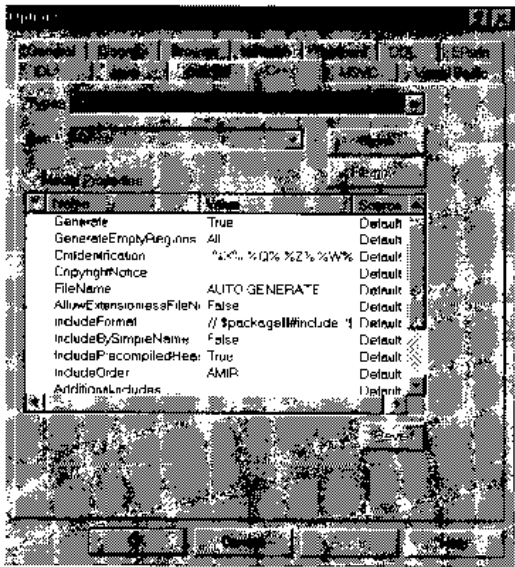
表12.5列出了头文件的许多属性及其用途和缺省值。

表12.5 头文件代码生成属性

属性	用途	缺省值
Generate	控制是否生成头文件	True
GenerateEmptyRegions	控制是否生成保护区域	All
CmlIdentification	用于输入配置管理软件可用的代码	%X%%Q%%Z%%W%
CopyRightNotice	用于在文件中包括版权说明	Blank
FileName	设置头文件名	组件名
AllowExtensionlessFileName	允许产生没有扩展名的头文件	False
InclusionProtectionSymbol	设置预处理器指令中防止头文件重复出现的符号	AUTO GENERATE (同组件名)
IncludeFormat	设置include语句使用的语法	//\$package #include "\$file"
IncludeBySimpleName	确定include语句是包含完整路径名还是只包含文件名	False (完整路径)
IncludePrecompiledHeader	确定是否在include语句开头放上预编译头	False
IncludeOrder	用include语句设置代码区域顺序	AMIR
AdditionalIncludes	输入代码中要包括的其他include语句	Blank
InliningStyle	控制是否生成内联函数定义 (在类定义后面或类声明中)	FollowingClassDeclaration
TypesDefined	指定模块中定义的类型	Blank
IncludeClosure	指定模块包括的头文件	Blank

模块体（实现文件）属性

模块体属性是与Rose生成的实现文件（.CPP）相关的属性。这些属性可以确定是否生成文件，在文件中包括版型声明，改变文件名或控制实现文件的其他规范。



这些属性可以在两个地方设置。要设置所有体文件的属性，选择Tools►Options，然后选择C++标签，并从下拉列表框选择Module Body。要设置一个文件的属性，在组件规范窗口中选择C++标签并编辑属性。

表12.6列出了体文件的许多属性及其用途和缺省值

表12.6 体文件代码生成属性

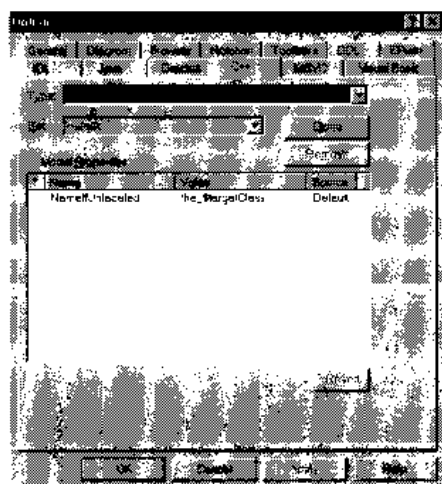
属性	用途	缺省值
Generate	控制是否生成体文件	True
GenerateEmptyRegions	控制是否生成保护区域	All
CmlIdentification	用于输入配置管理软件可用的代码用	%X% %Q% %Z% %W%
CopyrightNotice	用于在文件中包括版权说明	Blank
FileName	设置体文件名	组件名
AllowExtensionlessFileName	允许产生没有扩展名的体文件	False
IncludeFormat	设置include语句使用的语法	//\$package #include "\$file"
IncludeBySimpleName	确定include语句是包含完整路径名还是只包含文件名	False（完整路径）
IncludePrecompiledHeader	确定是否在include语句开头放上预编译头	False
IncludeOrder	用include语句设置代码区域顺序	AMIR
AdditionalIncludes	输入代码中要包括的其他include语句	Blank
InliningStyle	控制是否生成内联函数定义（在类定义后面或类声明中）	FollowingClassDeclaration
TypesDefined	指定模块中定义的类型	Blank
IncludeClosure	指定模块包括的体文件	Blank

关联属性

关联属性是涉及关联关系的C++代码生成属性。这些属性可以控制关联代码生成。在C++中，具有一个关联属性。

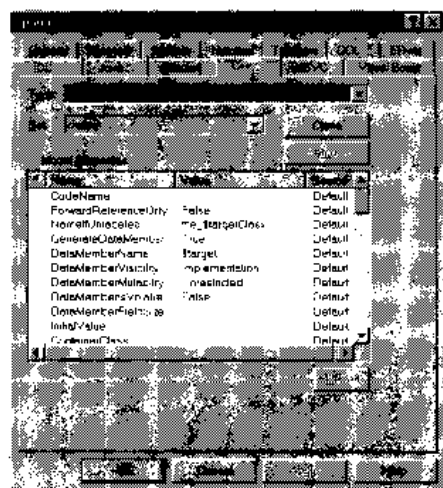
要设置这个属性，选择Tools►Options，然后选择C++标签，并从下拉列表框选择Association。

关联只有一个代码生成属性。NameIfUnlabeled属性设置没有作用名时该关系生成的属性名。缺省用数值_\$targetClass。



作用属性

作用属性是影响关系代码生成的C++代码生成属性。利用作用属性可以设置生成的属性名，设置属性使用的容器类和改变生成代码的其他部分。



和大多数其他属性设置一样，设置这些属性的地方有两个。要对所有作用进行设置，选择Tools►Options，然后选择C++标签，并从下拉列表框选择Role。要对一个作用进行设置，选择作用规范窗口的C++标签，并在此进行编辑。

表12.7列出了作用的属性、用途和缺省值。

表12.7 作用代码生成属性

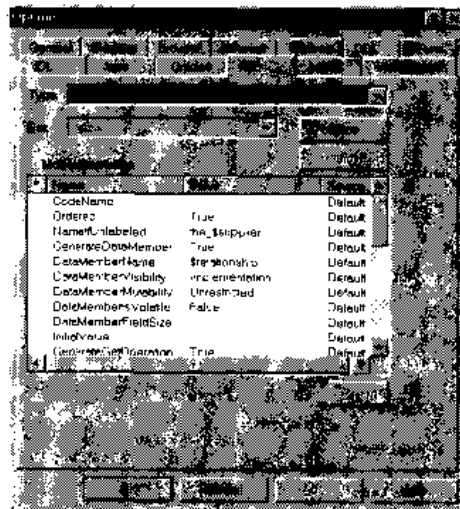
属性	用途	缺省值
CodeName	设置生成的关系名	Blank
ForwardReferenceOnly	控制Rose是否在模块中类定义之前放上关系供应者类的声明	False
NameIfUnlabeled	设置没有作用名且DataMemberName为blank或\$relationship时生成的属性名	the_\$targetClass
GenerateDataMember	控制是否对关系生成属性	True
DataMemberName	设置生成的关系名	\$target
DataMemberVisibility	设置生成属性的可见性	Implementation
DataMemberMutability	控制该属性生成关键字mutable、const或neither都不用	都不用（无限制）
DataMemberIsVolatile	控制代码中是否使用V关键字	False
DataMemberFieldSize	指定生成属性中的位数	Blank
Initial Value	设置生成属性的初始值	Blank
ContainerClass	设置对关系所生成属性的数据类型	Blank
ContainerGet	如果关系有限定符，这个属性输入从容器中取特定元素的原型C++表达式	\$data.get (keys)
ContainerSet	如果关系有限定符，这个属性输入设置容器中特定元素的原型C++表达式	\$data.set (keys, \$value)
QualifiedContainer	如果关系有限定符，这个属性输入存放供应者类引用和限定符键值的容器类	Blank
AssocClassContainer	生成支持与关联类关系的属性时指定所用数据类型	\$supplier*
AssocClassInitial Value	生成支持与关联类关系的属性时指定其初始值	Blank
GetSetKinds	控制Get和Set操作为虚拟、静态、抽象、朋友或其他	None (Common)
GetSetByReference	控制Get和Set操作按引用或按值传递变元和返回值	按值
GenerateGetOperation	控制属性是否生成Get操作	True
GetName	设置所生成Get操作的名称	get_\$target
GetIsConst	控制属性是用const关键字	True
GetResultIsConst	控制Get操作是否返回const值	GetIsConst (Same_As_Function) 中设置的值
InlineGet	控制Rose是否内联Get操作	True
GenerateSetOperation	控制属性是否生成Set操作	True
SetName	设置所生成Set操作的名称	set_\$target
SetReturns Value	控制Get操作是否返回非空值	False
InlineSet	控制Rose是否内联Set操作	True
QualifiedGetSetBy-Reference	指定限定Get和Set操作按引用或按值传递变元和返回值	Same_As_GetSetByReference
GenerateQualifiedGet-Operation	控制Rose是否生成一个Get操作，从支持关系的容器中取得项目	True
QualifiedGetName	设置限定Get操作名	get_\$target

(续表)

属性	用途	缺省值
QualifiedGetIsConst	控制限定Get操作是否用const关键字生成	True
QualifiedGetResultIs-Const	控制限定Get操作是否返回const值	QualifiedGetIsConst (Same_As_Function) 中设置的值
InlineQualifiedGet	控制Rose是否内联限定Get操作	True
GenerateQualifiedSet-Operation	控制Rose是否生成一个Set操作, 从支持关系的容器中设置项目	True
QualifiedSetName	设置限定Get操作名	get_\$target
QualifiedSetReturns-Value	控制限定Get操作是否非空值	False
InlineQualifiedSet	控制Rose是否内联限定Set操作	True
GenerateAssocClassData-Member	控制Rose是否生成支持关联类关系的数据成员	True
AssocClassDataMember-Name	设置支持关联类关系的数据成员	\$target
AssocClassDataMember-Visibility	设置支持关联类关系的数据成员可见性	Implementation
AssocClassDataMember-Mutability	控制是否用mutable、const或不用这些关键字生成数据成员	不用这些关键字(不限制)
AssocClassDataMember-IsVolatile	控制代码中是否使用V关键字	False
AssocClassGetSetKinds	控制Get和Set操作为虚拟、静态、抽象、朋友或其他	None (Common)
GenerateAssocClassGet-Operation	控制Rose是否对支持关联类关系的数据成员生成Get函数	True
AssocClassGetName	设置对支持关联类关系的数据成员生成Get函数名称	get_\$target
AssocClassGetIsConst	控制是否用const关键字产生关联类Get操作	True
AssocClassGetResultIs-Const	控制关联类Get操作是否返回const值	QualifiedGetIsConst (Same_As_Function) 中设置的值
InlineAssocClassGet	控制Rose是否内联关联类Get操作	True
GenerateAssocClassSet-Operation	控制Rose是否对支持关联类关系的数据成员生成Set函数	True
AssocClassSetName	设置对支持关联类关系的数据成员生成Set函数名称	set_\$target
AssocClassSetReturns-Value	控制关联类Set操作是否返回非空值	False
InlineAssocClassSet	控制Rose是否内联关联类Set操作	True
AssocClassForwardReferenceOnly	控制关联类定义之前是否放上关联类提前声明	True
GenerateForwardReference (98)	控制是否用#include和正引用放上引用接口	False (#include)
IsReadOnly (98)	控制数据成员是否只读	False
BoundedRoleType (98)	控制约束关系中是否使用顺序或数组	Sequence

累积 (Has关系) 属性

Has属性是涉及累积的C++属性。利用这些属性可以控制累积生成的代码。



要设置这些属性，选择Tools>Options，然后选择C++标签，并从下拉列表框中选择Has。表12.8列出了累积的代码生成属性及其用途和缺省值。

表12.8 累积代码生成属性

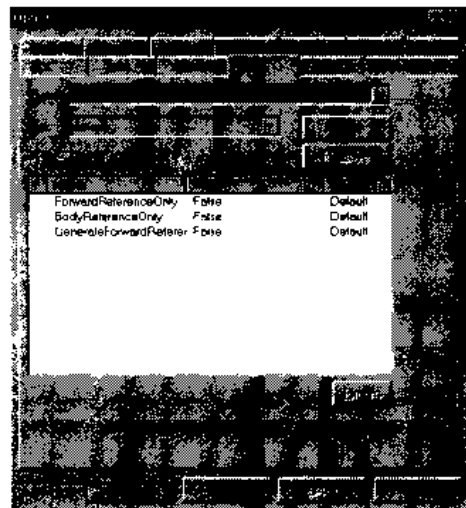
属性	用途	缺省
CodeName	设置累积代码中的关系名	Blank
Ordered	控制关系供应者方的对象是否排序	True
NameIfUnlabeled	设置无作用名而DataMemberName为blank或\$relationship时所产生的属性名	the_\$supplier
GenerateDataMember	控制是否对关系生成属性	True
DataMemberName	设置产生的属性名	\$relationship
DataMemberVisibility	设置产生的属性可见性	implementation
DataMemberMutability	控制该属性生成关键字mutable、const或neither都不用	都不用（无限制）
DataMemberIsVolatile	控制代码中是否使用V关键字	False
DataMemberFieldSize	指定生成属性中的位数	Blank
InitialValue	设置生成属性的初始值	Blank
GenerateGetOperation	控制是否对数据成员生成Get操作	True
GenerateSetOperation	控制是否对数据成员生成Set操作	True
GetName	设置生成的Get操作名	get_<attribute name>
SetName	设置生成的Set操作名	set_<attribute name>
GetSetKinds	控制Get和Set操作作为虚拟、静态、抽象、朋友或其他	None (Common)
ContainerClass	设置倍数大于一时生成属性所用的数据类型	Blank
SelectorName	控制是否对数据成员生成直接Set或Get函数	Blank (False)
SelectorType	指定SelectorName属性中指定的变元类型	Blank
GetIsConst	控制是否用const关键字生成Get操作	True
GetResultsIsConst	控制Get操作是否返回const值	GetIsConst (Same_As_Function) 设置的值

(续表)

属性	用途	缺省
GetSetByReference	控制Set或Get操作按引用或按数值传递变元和返回值	按值
InlineGet	控制Rose是否内联Get操作	True
SetReturnsValue	控制Set操作是否返回非空值	False
InlineSet	控制Rose是否内联Set操作	True
ForwardReferenceOnly	控制Rose是否在模块的类定义前面放上关系供应者类的前声明	False
GenerateForwardReference (98)	控制引用接口是否包括#include或forward引用	False (#include)
IsReadOnly (98)	控制数据成员是否只读	False
BoundedHasRelType (98)	控制约束关系中是否使用顺序和数组	Sequence

依赖性属性

依赖性属性控制C++中如何实现依赖性关系。



这些属性可以在两个地方设置。要设置所有依赖性的属性，选择Tools►Options，然后选择C++标签，并从下拉列表框选择Dependency。要设置一个文件的属性，在组件规范窗口中选择C++标签并编辑属性。

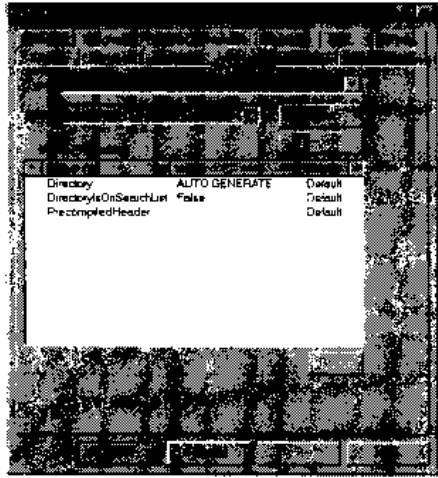
表12.9列出依赖性代码生成属性

表12.9 依赖性代码生成属性

属性	用途	缺省值
ForwardReferenceOnly	控制Rose是否在模块的类定义前面放上关系供应者类的前说明	False
BodyReferenceOnly	控制是否只在模块体中而不在模块规范中生成前声明或#include语句	False
GenerateForwardReference (98)	控制引用接口是否包括#include或forward引用	#include (False)

子系统属性

子系统属性是Rose模型Component view包中适用的属性。



要设置子系统属性，可以选择Tools►Options、选择C++标签，并从下拉列表框选择Sub-system。要设置一个子系统的子系统属性，在包规范窗口中选择C++标签并在此编辑属性。

表12.10列出了子系统代码生成属性。

表12.10 子系统代码生成属性

属性	用途	缺省值
Directory	设置对这个包生成的目录名	包名 (AUTO GENERATE)
DirectoryIsOnSearchList	控制#include语句中的路径包含绝对或相对路径	绝对路径 (False)
PrecompiledHeader	指定生成代码中应包括的预编译头	Blank

类类别属性

子系统属性是Rose模型Component view包中适用的属性，而类类别属性适用于模型中的Logical view包。

表12.11列出类类别属性。

表12.11 类代码生成属性

属性	用途	缺省值
IsNameSpace	指定这个包是否名称空间	False
Indent	包是名称空间时设置缩排空格数	2
CodeName	设置名称空间名	Blank
GenerateEmptyRegions	设置生成的保护区域类型	All

一般化属性

和其他关系一样，可以在C++中设置一般化（继承）关系的代码生成属性，一般化（继承）关系只有一个属性InstanceArguments，可以设置从参数化类继承时使用的实际变元。这个属性缺省为空白。

生成代码

下面几节介绍一个类、属性、操作及类间关系生成的C++代码。这些节中均会用一些样本代码显示Rose模型中生成的内容。

Rose生成代码时使用模型元素规范中提供的信息。例如，它在生成类代码时要查阅类的不同细节（可见性、属性、操作等）。

说明：下面几节的代码样本是用Rose 98i生成的。Rose 98生成类似代码，但稍有不同。

下面先介绍典型类的代码生成过程。

类

对象模型中的类在生成代码时变成C++类。每个类生成如下代码：

```
Class TheClass
{
public:
    TheClass();
    ~TheClass();
};
```

但是，代码中还生成大量其他信息。稍后将介绍完整的头和实现文件。类的所有属性、操作和关系都会反映在生成的代码中。每个类生成的主要元素包括：

- 类名
- 类可见性
- 类的构造器
- 类的删除器
- 每个属性的Get()和Set()操作
- 类文档
- 属性
- 操作
- 关系
- 文档

模型中的每个类生成两个C++文件，一个是头文件，一个是实现文件。每个文件都用类名命名。例如，Employee类产生Employee.h和Employee.cpp文件。

生成代码时，Rose用模型的Component视图中建立的包结构生成相应目录。模型中的每个包生成一个目录。在Rose生成的目录中，由该包中类的.cpp和.h文件。如果Component视图中没有生成组件和包，则Rose用Logical视图中的包结构生成相应目录。

Rose模型中的大多数信息都直接用于生成代码。例如，每个类的属性、操作、关系和类名直接影响生成的代码。其他模型字段（如输入类文档）不直接影响代码，这些字段值成为生成代码的说明语句。

表12.12列出了类规范窗口中的字段及哪些字段直接影响生成的代码。

表12.12 类规范窗口中的字段对生成代码的影响

字段	对代码的影响
Name	模型中的名称变成类名
Type	直接影响生成的类类型
Stereotype	说明语句
Export Control	直接影响生成的类的可见性
Documentation	说明语句
Cardinality	说明语句
Space	说明语句
Persistence	生成代码中的说明语句，但影响类是否生成DDL
Concurrency	说明语句
Abstract	生成抽象类
Formal Arguments	参数化类的代码中包括正式变元
Operations	代码中生成
Attributes	代码中生成
Relationships	代码中生成

下面看看下列类生成的代码：



生成头文件

下列代码是这个类生成的头文件。我们一次一段地介绍头文件，看看Rose生成了什么项目。但首先要介绍Rose插入源代码中的注释。Rose加进这些注释，从而在修改代码和重新生成（逆向转出工程代码）时不需要改写这些改变。典型的代码段如下：

```

#### begin name%123456789000.codesection preserve=yes
#### end name%123456789000.codesection

```

12位数字与字母字符串（这里为%123456789000）是Rose赋予各个代码段的唯一标识符。**preserve=**（no/yes）告诉Rose在重新生成源代码时重新生成这段或让它保持不变。这些标识符保证模型和代码在逆向转出工程代码时不受影响。例如，生成代码后，在#### begin和#### end语句之间插入实现代码。如果逆向转出工程代码代码到Rose模型，对模型进行改变，并重新生成代码，则Rose不会改写或删除刚刚完成的程序。

上述类的头文件如下:

```
//## begin module%37267F000276.cm preserve=no
// %X% %Q% %Z% %W%
//## end module%37267F000276.cm

//## begin module%37267F000276.cp preserve=no
//## end module%37267F000276.cp

//## Module: Sampleclass%37267F000276; Package specification
//## Subsystem: <Top Level>
//## Source file: C:\Program Files\Rational\Rose 98i\C++\source\
Sampleclass.h

#ifndef Sampleclass_h
#define Sampleclass_h 1

//## begin module%37267F000276.additionalincludes preserve=no
//## end module%37267F000276.additionalincludes

//## begin module%37267F000276.includes preserve=yes
//## end module%37267F000276.includes

//## begin module%37267F000276.declarations preserve=no
//## end module%37267F000276.declarations

//## begin module%37267F000276.additionalDeclarations preserve=yes
//## end module%37267F000276.additionalDeclarations

//## begin SampleClass%37267EF301B8.preface preserve=yes
//## end SampleClass%37267EF301B8.preface

//## Class: SampleClass%37267EF301B8
//## Category: <Top Level>
//## Persistence: Transient
//## Cardinality/Multiplicity: n

class SampleClass
{
    //## begin SampleClass%37267EF301B8.initialDeclarations preserve=yes
    //## end SampleClass%37267EF301B8.initialDeclarations

public:
    //## Constructors (generated)
    SampleClass();

    SampleClass(const SampleClass &right);

    //## Destructor (generated)
    ~SampleClass();

    //## Assignment Operation (generated)
    const SampleClass & operator=(const SampleClass &right);
```

```

    //## Equality Operations (generated)
    int operator==(const SampleClass &right) const;

    int operator!=(const SampleClass &right) const;

    // Additional Public Declarations
    //## begin SampleClass%37267EF301B8.public preserve=yes
    //## end SampleClass%37267EF301B8.public
protected:
    // Additional Protected Declarations
    //## begin SampleClass%37267EF301B8.protected preserve=yes
    //## end SampleClass%37267EF301B8.protected

private:
    // Additional Private Declarations
    //## begin SampleClass%37267EF301B8.private preserve=yes
    //## end SampleClass%37267EF301B8.private

private: //## implementation
    // Additional Implementation Declarations
    //## begin SampleClass%37267EF301B8.implementation preserve=yes
    //## end SampleClass%37267EF301B8.implementation
};

//## begin SampleClass%37267EF301B8.postscript preserve=yes
//## end SampleClass%37267EF301B8.postscript

// Class SampleClass

//## begin module%37267F000276.epilog preserve=yes
//## end module%37267F000276.epilog

#endif

```

可以看出，Rose生成与模型中的类同名的类。下面一次一段地介绍这个文件。

配置管理段

配置管理支持与配置管理软件的集成，包括下列语句：

```

//## begin module%37267F000275.cm preserve=no
// %X% %Q% %Z% %W%
//## end module%37267F000276.cm

```

配置管理段包括配置管理设置的信息。要改变配置管理设置，选择Tools►Options，在C++标签中，从Type下拉列表框选择Module Specification打开模块规范代码生成属性。可以将CmIdentification属性值变为配置管理软件能识别的字符串。

这个设置中可以采用的样本值如下：

- \$date插入生成代码的日期
- \$time插入生成代码的时间

- \$module插入组件名
- \$file插入组件文件

版权声明段

版权声明段可以在代码中加入版权声明，包括下列两行：

```
///

```
begin module%37267F000276.cp preserve=no
///

```
## end module%37267F000276.cp
```


```


```

缺省情况下，代码不包括版权声明段。但如果要增加版权声明，可以改变代码生成属性。选择**Tools**►**Options**，在**C++**标签中，从**Type**下拉列表框选择**Module Specification**打开模块规范代码生成属性。将**CopyrightNotice**字段变为包括任何版权信息。如果输入这个信息，则它会和上述两行一起出现在头文件中。

这个设置中可以采用的样本值如下：

- \$date插入生成代码的日期
- \$time插入生成代码的时间
- \$module插入组件名
- \$file插入组件文件

版权声明放在

```
///

```
begin
```

和

```
///

```
## end
```

语句之间，以便在逆向转出工程代码期间得到保护。
```


```

模块段

模块段包含所生成组件的基本信息，包括下列语句：

```
///

```
Module: Sampleclass%37267F000276; Package specification
///

```
## Subsystem: <Top Level>
///

```
Source file: C:\Program Files\Rational\Rose 98i\C++\source\
Sampleclass.h
```


```


```


```

这个段包括描述类所在模块（即组件）的说明语句。本例中，从第一行可以看出，**SampleClass**类在包规范**SampleClass**中。

第二行说明类放在哪个**Logical view**包中。这里类不在某个**Logical**视图包中，因此其子系统为**<Top Level>**。

第三行列出头文件本身的位置，其格式取决于模块规范的属性设置。

预处理器指令段

预处理器指令段包括下列语句：

```
#ifndef Sampleclass_h
#define Sampleclass_h 1
```

这些语句放在代码中可以防止头文件重复出现，其格式取决于模块规范的属性设置。

包括段

代码中生成两个包括段。第一个用于模型中用代码生成属性指定的任何包括语句。第二个让编程人员手工将包括语句加进源代码中。`#include`语句插入`///
end`之间的保护代码区。Rose保护Sampleclass之间的`#include`语句。

```
///  
end module%37267F000276.additionalIncludes
```

在这个段中，Rose插入要加进代码的其他`#include`语句。要加进新`#include`语句，选择Tools►Options，在C++标签中，从Type下拉列表框选择Module Specification打开模块规范代码生成属性。然后在AdditionalIncludes字段中输入其他`#include`语句。这个字段中输入的名称应为一个清单，每个名称用格式“name”或<name>。

```
///  
end module%37267F000276.includes
```

本节让编程人员手工将包括语句加进源代码中。

声明段

声明段可以将声明插入代码中。生成代码后，可以在`///
end`语句之间插入声明，保证该声明在逆向转出工程代码时不受影响。

```
///  
end module%37267F000276.declarations  
  
///  
end module%37267F000276.additionalDeclarations
```

类定义段

类定义段包含类本身的信息，包括类名、Logical视图和Component视图包、持续性和基数。类定义段第一行包括类名，第二行Category列出类所在的Logical视图包。如果类不在Logical视图包中，则显示<Top Level>。用Rose 98而不是98i生成代码时，要加进Subsystem行，Subsystem行列出类所在的Component视图包。如果类不在Component视图包中，则显示Cardinality/Multiplicity行显示类在指定时间在内存中可以有几个实例。

在类定义中，包括缺省构造器、复制构造器和删除器。如果不要产生这些函数或要改变生成选项，可以改变这个类的C++代码生成选项。打开类规范窗口，然后选择C++标签。

可以改变GenerateDefaultConstructor、DefaultConstructorVisibility、InlineDefaultConstructor和ExplicitDefaultConstructor属性，控制是否生成缺省构造器、构造器可见性、是否内联生成构造器、构造器是否显式构造器。

改变GenerateCopyConstructor、CopyConstructorVisibility、InlineCopyConstructor和ExplicitCopyConstructor属性可以控制是否生成复制构造器、构造器可见性、是否内联生成构造器、构造器是否显式构造器。

改变GenerateDestructor、DestructorVisibility、DestructorKind和InlineDestructor属性可以控制是否生成删除器、删除器可见性、删除器是否虚拟、删除器是否显式构造器。

缺省情况下, Rose生成赋值操作和等值操作。如果不要产生这些函数或要改变生成选项, 可以改变这个类的GenerateAssignmentOperation, AssignmentVisibility, AssignmentKind, InlineAssignmentOperation, GenerateEqualityOperations, Equality-Visibility, EqualityKind和InlineEqualityOperations代码生成属性。

如果类有公开属性或操作, 则这些声明也放在这个文件的公开段中。

在保护、专用和专用实现段中, Rose分别包括保护、专用和专用实现属性与操作的声明。

```
class SampleClass
{
    ///# begin SampleClass%37267EF301B8.initialDeclarations preserve=yes
    ///# end SampleClass%37267EF301B8.initialDeclarations

    public:
        ///# Constructors (generated)
        SampleClass();

        SampleClass(const SampleClass &right);

        ///# Destructor (generated)
        ~SampleClass();

        ///# Assignment Operation (generated)
        const SampleClass & operator=(const SampleClass &right);

        ///# Equality Operations (generated)
        int operator==(const SampleClass &right) const;

        int operator!=(const SampleClass &right) const;

        // Additional Public Declarations
        ///# begin SampleClass%37267EF301B8.public preserve=yes
        ///# end SampleClass%37267EF301B8.public

    protected:
        // Additional Protected Declarations
        ///# begin SampleClass%37267EF301B8.protected preserve=yes
        ///# end SampleClass%37267EF301B8.protected

    private:
        // Additional Private Declarations
        ///# begin SampleClass%37267EF301B8.private preserve=yes
        ///# end SampleClass%37267EF301B8.private

    private: ///# implementation
        // Additional Implementation Declarations
        ///# begin SampleClass%37267EF301B8.implementation preserve=yes
        ///# end SampleClass%37267EF301B8.implementation

};

///# begin SampleClass%37267EF301B8.postscript preserve=yes
///# end SampleClass%37267EF301B8.postscript
```

```
// Class SampleClass

///< begin module%37267F000276.epilog preserve=yes
///< end module%37267F000276.epilog

#endif
```

如果用文档窗口或类规范窗口文档区输入类文档，则这个文档在代码中成为说明语句。

生成的实现文件

Rose生成的另一文件是实现文件，缺省扩展名为.CPP。下面是上述类的实现文件。

```
///< begin module%37267F0103AC.cm preserve=no
// %X% %Q% %Z% %W%
///< end module%37267F0103AC.cm

///< begin module%37267F0103AC.cp preserve=no
///< end module%37267F0103AC.cp

///< Module: Sampleclass%37267F0103AC; Package body
///< Subsystem: <Top Level>
///< Source file: C:\Program Files\Rational\Rose 98i\C++\source\
Sampleclass.cpp

///< begin module%37267F0103AC.additionalIncludes preserve=no
///< end module%37267F0103AC.additionalIncludes

///< begin module%37267F0103AC.includes preserve=yes
///< end module%37267F0103AC.includes

// Sampleclass
#include "Sampleclass.h"
///< begin module%37267F0103AC.declarations preserve=no
///< end module%37267F0103AC.declarations

///< begin module%37267F0103AC.additionalDeclarations preserve=yes
///< end module%37267F0103AC.additionalDeclarations

// Class SampleClass

SampleClass::SampleClass()
    ///< begin SampleClass::SampleClass%.hasinit preserve=no
    ///< end SampleClass::SampleClass%.hasinit
    ///< begin SampleClass::SampleClass%.initialization preserve=yes
    ///< end SampleClass::SampleClass%.initialization
{
    ///< begin SampleClass::SampleClass%.body preserve=yes
    ///< end SampleClass::SampleClass%.body
}
```

```

SampleClass::SampleClass(const SampleClass &right)
    ///# begin SampleClass::SampleClass%copy.hasinit preserve=no
    ///# end SampleClass::SampleClass%copy.hasinit
    ///# begin SampleClass::SampleClass%copy.initialization preserve=yes
    ///# end SampleClass::SampleClass%copy.initialization
{
    ///# begin SampleClass::SampleClass%copy.body preserve=yes
    ///# end SampleClass::SampleClass%copy.body
}

SampleClass::~SampleClass()
{
    ///# begin SampleClass::~SampleClass%.body preserve=yes
    ///# end SampleClass::~SampleClass%.body
}

const SampleClass & SampleClass::operator=(const SampleClass &right)
{
    ///# begin SampleClass::operator=%%.body preserve=yes
    ///# end SampleClass::operator=%%.body
}

int SampleClass::operator==(const SampleClass &right) const
{
    ///# begin SampleClass::operator==%.body preserve=yes
    ///# end SampleClass::operator==%.body
}

int SampleClass::operator!=(const SampleClass &right) const
{
    ///# begin SampleClass::operator!=%.body preserve=yes
    ///# end SampleClass::operator!=%.body
}

// Additional Declarations
    ///# begin SampleClass%37267EF301B8.declarations preserve=yes
    ///# end SampleClass%37267EF301B8.declarations
    ///# begin module%37267F0103AC.epilog preserve=yes
    ///# end module%37267F0103AC.epilog

```

和头文件一样，下面逐段地介绍实现文件。

配置管理段

和头文件中一样，实现文件中的配置管理段支持与配置管理软件的集成，包括下列语句：

```
///  
begin module%37267F0103AC.cm preserve=no  
// %X% %Q% %Z% %W%  
///  
end module%37267F0103AC.cm
```

配置管理段包括匹配管理设置的信息。第二行的属性是缺省配置管理设置。

和头文件中一样，实现文件中也可以改变配置管理设置。要改变配置管理设置，选择Tools►Options，在C++标签中，从Type下拉列表框选择 **Module Body** 打开模块规范代码生成属性。可以用CmIdentification属性改变第二行的缺省值。这个属性可以将配置管理设置变为配置管理软件认识的字符串。这个设置中可以采用的样本值如下：

- \$date插入生成代码的日期
- \$time插入生成代码的时间
- \$module插入组件名
- \$file插入组件文件

版权声明段

实现文件中可以插入版权声明段。和头文件中一样，实现文件中版权声明段可以在代码中加入版权声明，包括下列两行：

```
///  
begin module%37267F0103AC.cp preserve=no  
///  
end module%37267F0103AC.cp
```

要增加版权声明，选择Tools►Options，在C++标签中，从Type下拉列表框选择**Module Body** 打开模块规范代码生成属性。可以将CopyrightNotice字段变为包括版权信息。如果输入这个信息，则它会和上述两行一起出现在头文件中。

这个设置中可以采用的样本值如下：

- \$date插入生成代码的日期
- \$time插入生成代码的时间
- \$module插入组件名
- \$file插入组件文件

版权声明放在///
begin和///
end语句之间，以便在逆向转出工程代码期间得到保护。

模块段

模块段包括所产生实现文件的基本信息，包括下列语句：

```
///  
Module: Sampleclass%37267F0103AC; Package body  
///  
Subsystem: <Top Level>  
///  
Source file: C:\Program Files\Rational\Rose 98i\C++\source\Sampleclass.cpp
```

这个段包括描述类所在模块（即组件）的说明语句。本例中，从第一行可以看出，SampleClass类在包规范SampleClass中。

第二行说明类放在哪个Logical视图包中。这里类不在某个Logical视图包中，因此其子系统为<Top Level>。

第三行列出实现文件本身的位置。

包括段

和头文件中一样，实现文件中也包含两个包括段。第一个用于模型中用代码生成属性指定的任何包括语句。第二个让编程人员手工将包括语句加进源代码中。**#include**语句插入**/// **begin****和**/// **end****之间的保护代码区。Rose保护Sampleclass之间的**#include**语句。

```
/// begin module%37267F0103AC.additionalIncludes preserve=no
/// end module%37267F0103AC.additionalIncludes
```

在这个段中，Rose插入要加进代码的其他**#include**语句。要加进新**#include**语句，选择**Tools>Options**，在C++标签中，从Type下拉列表框选择Module Body打开模块规范代码生成属性。然后在AdditionalIncludes字段中输入其他**#include**语句。这个字段中输入的名称应为一个清单，每个名称用格式“name”或<name>。

```
/// begin module%37267F0103AC.includes preserve=yes
/// end module%37267F0103AC.includes
```

本节让编程人员手工将包括语句加进源代码中。在包括段末尾，Rose插入类的头文件**#include**语句。本例中，生成下列两行：

```
// Sampleclass
#include "Sampleclass.h"
```

声明段

声明段可以将声明插入代码中。生成代码后，可以在**/// **begin****和**/// **end****语句之间插入声明，保证该声明在逆向转出工程代码时不受影响。

```
/// begin module%37267F0103AC.declarations preserve=no
/// end module%37267F0103AC.declarations

/// begin module%37267F0103AC.additionalDeclarations preserve=yes
/// end module%37267F0103AC.additionalDeclarations
```

类定义段

类定义段包含类的实现代码。Rose包括缺省构造器、备份构造器、删除器的代码。和头文件中一样，可以改变是否生成缺省构造器、备份构造器、删除器和设置生成属性。为此，在组件规范窗口的C++标签中改变代码生成属性。表12.13显示了影响缺省构造器、备份构造器、删除器的属性。

表12.13 影响缺省构造器、备份构造器、删除器的属性

属性	用途	缺省
GenerateDefaultConstructor	控制是否生成缺省构造器	True (DeclareAndDefine)
DefaultConstructorVisibility	控制缺省构造器的可见性	Public
InlineDefaultConstructor	确定缺省构造器是否内联	False
ExplicitDefaultConstructor	确定缺省构造器是否显示	False
GenerateCopyConstructor	控制是否生成备份构造器	True (DeclareAndDefine)
CopyConstructorVisibility	控制备份构造器的可见性	Public
InlineCopyConstructor	确定备份构造器是否内联	False
ExplicitCopyConstructor	确定备份构造器是否显式	False
GenerateDestructor	控制是否生成删除器	True
DestructorVisibility	控制删除器的可见性	Public
DestructorKind	确定删除器是否虚拟	Not virtual (Common)
InlineDestructor	确定删除器是否内联	False

下面列出类定义段的生成代码:

```
// Class SampleClass

SampleClass::SampleClass()
    /// begin SampleClass::SampleClass%.hasinit preserve=no
    /// end SampleClass::SampleClass%.hasinit
    /// begin SampleClass::SampleClass%.initialization preserve=yes
    /// end SampleClass::SampleClass%.initialization
{
    /// begin SampleClass::SampleClass%.body preserve=yes
    /// end SampleClass::SampleClass%.body
}

SampleClass::SampleClass(const SampleClass &right)
    /// begin SampleClass::SampleClass%copy.hasinit preserve=no
    /// end SampleClass::SampleClass%copy.hasinit
    /// begin SampleClass::SampleClass%copy.initialization preserve=yes
    /// end SampleClass::SampleClass%copy.initialization
{
    /// begin SampleClass::SampleClass%copy.body preserve=yes
    /// end SampleClass::SampleClass%copy.body
}

SampleClass::~~SampleClass()
{
    /// begin SampleClass::~~SampleClass%.body preserve=yes
    /// end SampleClass::~~SampleClass%.body
}

const SampleClass & SampleClass::operator=(const SampleClass &right)
{
    /// begin SampleClass::operator=%.body preserve=yes
```



```

    ///# end SampleClass::operator=%s.body
}

int SampleClass::operator==(const SampleClass &right) const
{
    ///# begin SampleClass::operator==%s.body preserve=yes
    ///# end SampleClass::operator==%s.body
}

int SampleClass::operator!=(const SampleClass &right) const
{
    ///# begin SampleClass::operator!=%s.body preserve=yes
    ///# end SampleClass::operator!=%s.body
}

// Additional Declarations
    ///# begin SampleClass%37267EF301B8.declarations preserve=yes
    ///# end SampleClass%37267EF301B8.declarations
    ///# begin module%37267F0103AC.epilog preserve=yes
    ///# end module%37267F0103AC.epilog

```

属性

除类本身以外，Rose还产生类属性。对每个属性，Rose包括：

- Visibility
- Data type
- Default value
- Get operation
- Set operation

对某个属性，Rose产生如下代码：

```

Class TheClass
{
public:
    int PublicAttribute;
    int GetPublicAttribute();
    int GetProtectedAttribute();
    int GetPrivateAttribute();
    void set_PublicAttribute (int value);
    void set_ProtectedAttribute (int value);
    void set_PrivateAttribute (int value);

protected:
    int ProtectedAttribute;

private:
    int PrivateAttribute;
};

```

但完整的头和实现文件中会生成更多细节，包括说明语句和Rose标识符。下面详细介绍下列类生成的代码：



上述类的属性在头文件中生成三段代码。第一段如下：

```
private:
    ///## Get and Set Operations for Class Attributes (generated)

    ///## Attribute: ID%372684250050
    const int get_ID () const;
    void set_ID (int value);
```

这段包括属性自动生成的Get和Set操作声明。可以通过改变属性的代码生成属性确定是否生成属性的Get和Set操作。表12.14列出了影响Get和Set操作的代码生成属性。

表12.14 影响Get和Set操作的属性

属性	用途	缺省
GenerateGetOperation	控制是否生成Get操作	True
GenerateSetOperation	控制是否生成Set操作	True
GetName	设置Get操作名	get_<attribute name>
SetName	设置Set操作名	set_<attribute name>
GetSetKinds	控制Get和Set操作为虚拟、静态、抽象、版权或其他	None (Common)
GetIsConst	控制是否用const关键字生成Get操作	True
GetResultsIsConst	控制Get操作是否返回const值	GetIsConst (Same_As_Function) 设置的值
GetSetByReference	控制Get和Set操作按引用或按值传递变元和返回值	按值传递 (False)
InlineGet	控制Rose是否内联Get操作	True
SetReturnsValue	控制Set操作是否返回非空值	False
InlineSet	控制Rose是否内联Set操作	True

属性头文件中生成的第二段代码如下：

```
private: ///## implementation
    // Data Members for Class Attributes
    ///## begin SampleClass::ID%372684250050.attr preserve=no
```

```
private: int {0}
int ID;
///< end SampleClass::ID%372684250050.attr
```

这段声明每个属性。这里专用属性ID是个整数。因此属性在代码的“Private:”段中列出。代码放在///< begin和///< end语句之间,以便在逆向转出工程代码时得到保护。

属性头文件中生成的第二段代码包括Get和Set操作的代码。Get操作返回属性值,Get操作设置属性值。

```
///< Get and Set Operations for Class Attributes (inline)

inline const int SampleClass::get_ID () const
{
    ///< begin SampleClass::get_ID%372684250050.get preserve=no
    return ID;
    ///< end SampleClass::get_ID%372684250050.get
}

inline void SampleClass::set_ID (int value)
{
    ///< begin SampleClass::set_ID%372684250050.set preserve=no
    ID = value;
    ///< end SampleClass::set_ID%372684250050.set
}
```

实现文件中有两个代码段包括属性信息:构造器和备份构造器。在这些构造器中,Rose包括设置属性缺省值的逻辑(如果指定了缺省值)。本例中,ID属性的缺省值为0,因此生成下列语句:

```
SampleClass::SampleClass()
    ///< begin SampleClass::SampleClass%.hasinit preserve=no
→    : ID(0)
    ///< end SampleClass::SampleClass%.hasinit
    ///< begin SampleClass::SampleClass%.initialization preserve=yes
    ///< end SampleClass::SampleClass%.initialization
{
    ///< begin SampleClass::SampleClass%.body preserve=yes
    ///< end SampleClass::SampleClass%.body
}

SampleClass::SampleClass(const SampleClass &right)
    ///< begin SampleClass::SampleClass%copy.hasinit preserve=no
→    : ID(0)
    ///< end SampleClass::SampleClass%copy.hasinit
    ///< begin SampleClass::SampleClass%copy.initialization preserve=yes
    ///< end SampleClass::SampleClass%copy.initialization
{
```

```

    ///## begin SampleClass::SampleClass%copy.body preserve=yes
    ///## end SampleClass::SampleClass%copy.body
};

```

操作

Rose对类中每个操作生成代码。对于每个操作，生成的代码包括操作名、参数、参数数据类型和返回值。每个操作生成下列代码：

```

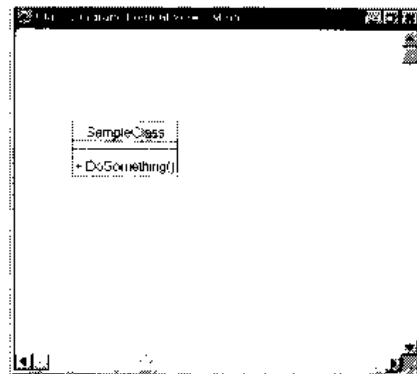
Class TheClass
{
public:
void PublicOperation();

protected:
void ProtectedOperation();

private:
void PrivateOperation();
};

```

同样，其中还有更多细节。这里介绍下列类的生成代码：



头文件中，Rose生成操作签名：

```

class SampleClass
{
    ///## begin SampleClass%3726867600AA.initialDeclarations preserve=yes
    ///## end SampleClass%3726867600AA.initialDeclarations

public:
    ///## Constructors (generated)
    SampleClass();

    SampleClass(const SampleClass &right);

    ///## Destructor (generated)
    ~SampleClass();

    ///## Assignment Operation (generated)
    const SampleClass & operator=(const SampleClass &right);

```

```

    ///# Equality Operations (generated)
    int operator==(const SampleClass &right) const;

    int operator!=(const SampleClass &right) const;

    ///# Other Operations (specified)
→    ///# Operation: DoSomething%3726869500FA

    "    // This is documentation entered into the documentation
    "    // window for the DoSomething operation.
    "
    int DoSomething (int Parameter1);

```

可以看出, 代码中生成完整的操作签名。输入的任何文档也出现在代码的说明语句中。如果输入操作协议、限定、异常、时间、空间、前提条件、词法和事后条件等信息, 则这个信息也会出现在头文件中, 见下面的代码。

```

    ///# Other Operations (specified)
    "    ///# Operation: DoSomething%3726869500FA; This is the operation
    protocol, used to describe in what order operations can run.
    // This is documentation entered into the documentation
    // window for the DoSomething operation.
    "    ///# Qualification: This is the operation qualification, which
    includes language-dependent qualifications.
    "    ///# Exceptions: Exceptions
    "    ///# Time Complexity: This is the operation time, which speci-
    fies how long the operation is expected to take.
    "    ///# Space Complexity: This is the operation size, which quan-
    tifies how much memory the operation is expected to take.
    "    ///# Preconditions:
    // These are the preconditions, which include conditions
    // that must be true before the operation can run.
    "    ///# Semantics:
    // These are the operation semantics, which describe, in
    // pseudocode, what the operation will do.
    "    ///# Postconditions:
    // These are the postconditions, which include conditions
    // that must be true after the operation runs.
    int DoSomething (int Parameter1) throw (Exceptions);

```

Rose在实现文件中也生成操作的代码。我们在前面介绍了**SampleClass**类的头文件, 下面介绍这个类的实现文件。

在实现文件中, **Rose**对定义在每个操作插入下列行:

```

    ///# Other Operations (implementation)
    int SampleClass::DoSomething (int Parameter1)
    {
        ///# begin SampleClass::DoSomething%3726869500FA.body preserve=yes
        ///# end SampleClass::DoSomething%3726869500FA.body
    }

```

生成代码后，在实现文件的`///
end`之间插入每个操作的实现代码。通过把代码放进保护代码区，可以在逆向转出工程代码期间保护这段代码。

如果用操作规范窗口Detail标签的Exceptions字段输入操作的异常信息，则这些异常出现在代码中，见下页代码：

```
///  
Other Operations {implementation}  
int SampleClass::DoSomething (int Parameter1) throw (Exceptions)  
{  
    ///  
    begin SampleClass::DoSomething%3726869500FA.body preserve=yes  
    ///  
    end SampleClass::DoSomething%3726869500FA.body  
}
```

和其他模型元素一样，可以通过修改代码生成属性控制操作的生成代码。例如，可以修改OperationKind属性以生成虚拟函数。前面列出了这个操作的代码生成属性，表12.15再次列出。

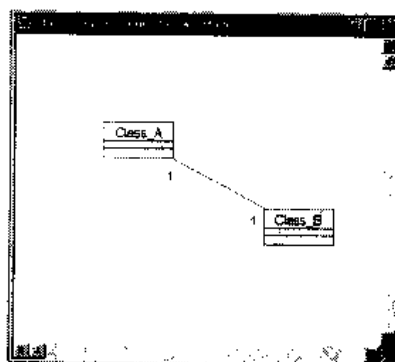
表12.15 操作代码生成属性

属性	用途	缺省值
CodeName	设置生成的操作名	模型中的操作名
OperationKind	控制操作为虚拟、静态、抽象、朋友或其他	None (Common)
OperationIsConst	控制操作是否用Const关键字生成	False
OperationIsExplicit	控制生成操作时是否使用关键字explicit	False
Inline	控制是否内联操作	False
EntryCode	用于输入操作的代码或说明语句（这个代码不放在保护区域中，否则会在逆向转出工程代码中被保护）	Empty
ExitCode	用于输入操作的代码或说明语句（这个代码不放在保护区域中，否则会在逆向转出工程代码中被保护）	Empty
GenerateEmptyRegions	控制操作是否生成保护区域	总是生成 (ALL)
BodyAnnotations	控制实现文件中是否生成操作文档、并发性、空间、时间、限定、异常、前提条件、事后条件和词法的图注	全部生成

双向关联

要支持双向关联，Rose在代码中生成属性。关系中的每个类包含一个支持关联的属性。本节介绍的代码生成属性包括：

- DataMemberVisibility作用属性
- GenerateGetOperation作用属性
- GenerateSetOperation作用属性



对上述两个类生成的代码如下:

```

Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    Class_B *the_Class_B;
};

and

Class Class_B
{
public:
    Class_B();
    ~Class_B();
private:
    Class_A *the_Class_A;
};

```

可以看出, Rose在双向关联关系的两端自动生成属性。利用_Class_B属性, Class_A可以方便地访问Class_B。利用_Class_A属性, Class_B可以方便地访问Class_A。

下面详细介绍对上述两个类生成的代码。关联在Class_A和Class_B的头文件中均有体现。Class_A的头文件如下, 我们要逐行介绍:

```

//## begin module%3726883A0212.cm preserve=no
// %X% %Q% %Z% %W%
//## end module%3726883A0212.cm

//## begin module%3726883A0212.cp preserve=no
//## end module%3726883A0212.cp

//## Module: Class_A%3726883A0212; Pseudo Package specification
//## Source file: C:\Program Files\Rational\Rose 98i\C++\source\
Class_A.h

#ifndef Class_A_h
#define Class_A_h 1

//## begin module%3726883A0212.additionalIncludes preserve=no
//## end module%3726883A0212.additionalIncludes

//## begin module%3726883A0212.includes preserve=yes
//## end module%3726883A0212.includes

→ // Class_B
→ #include "Class_B.h"

//## begin module%3726883A0212.additionalDeclarations preserve=yes
//## end module%3726883A0212.additionalDeclarations

```

```

    ///## begin Class_A%3726883A0212.preface preserve=yes
    ///## end Class_A%3726883A0212.preface

    ///## Class: Class_A%3726883A0212
    ///## Category: <Top Level>
    ///## Persistence: Transient
    ///## Cardinality/Multiplicity: n

class Class_A
{
    ///## begin Class_A%3726883A0212.initialDeclarations preserve=yes
    ///## end Class_A%3726883A0212.initialDeclarations

public:
    ///## Constructors (generated)
    Class_A();

    Class_A(const Class_A &right);

    ///## Destructor (generated)
    ~Class_A();

    ///## Assignment Operation (generated)
    const Class_A & operator=(const Class_A &right);

    ///## Equality Operations (generated)
    int operator==(const Class_A &right) const;

    int operator!=(const Class_A &right) const;

    ///## Get and Set Operations for Associations (generated)
    →    ///## Association: <unnamed>%3726887C02D0
    →    ///## Role: Class_A::<the_Class_B>%3726887D0032
    →    const Class_B * get_the_Class_B () const;
    →    void set_the_Class_B (Class_B * value);

    // Additional Public Declarations
    ///## begin Class_A%3726883A0212.public preserve=yes
    ///## end Class_A%3726883A0212.public

protected:
    // Additional Protected Declarations
    ///## begin Class_A%3726883A0212.protected preserve=yes
    ///## end Class_A%3726883A0212.protected

private:
    // Additional Private Declarations
    ///## begin Class_A%3726883A0212.private preserve=yes
    ///## end Class_A%3726883A0212.private

private: ///## implementation
    // Data Members for Associations

```



```

→      ///Association: <unnamed>%3726887C02D0
→      ///begin Class_A::<the_Class_B>%3726887D0032.role preserve=no
      public: Class_B {1 -> 1RHN}
→      Class_B *the_Class_B;
→      ///end Class_A::<the_Class_B>%3726887D0032.role

      // Additional Implementation Declarations
      ///begin Class_A%3726883A0212.implementation preserve=yes
      ///end Class_A%3726883A0212.implementation
};

///begin Class_A%3726883A0212.postscript preserve=yes
///end Class_A%3726883A0212.postscript

// Class Class_A

→      ///Get and Set Operations for Associations (inline)
→
→      inline const Class_B * Class_A::get_the_Class_E () const
→      {
→      ///begin Class_A::get_the_Class_B%3726887D0032.get preserve=no
→      return the_Class_B;
→      ///end Class_A::get_the_Class_B%3726887D0032.get
→      }
→
→      inline void Class_A::set_the_Class_B (Class_B * value)
→      {
→      ///begin Class_A::set_the_Class_B%3726887D0032.set preserve=no
→      the_Class_B = value;
→      ///end Class_A::set_the_Class_B%3726887D0032.set
→      }

///begin module%3726883A0212.epilog preserve=yes
///end module%3726883A0212.epilog

#endif

```

前两个标号行为:

```

// Class_B
#include "Class_B.h"

```

保证包括Class_B的头文件, 从而建立Class_A与Class_B之间的关系, 使Class_A知道Class_B存在。尽管这里不介绍Class_B的整个头文件, 但它包括类似语句:

```

// Class_A
#include "Class_A.h"

```

第二组标号行:

```

///Association: <unnamed>%3726887C02D0
///Role: Class_A::<the_Class_B>%3726887D0032

```

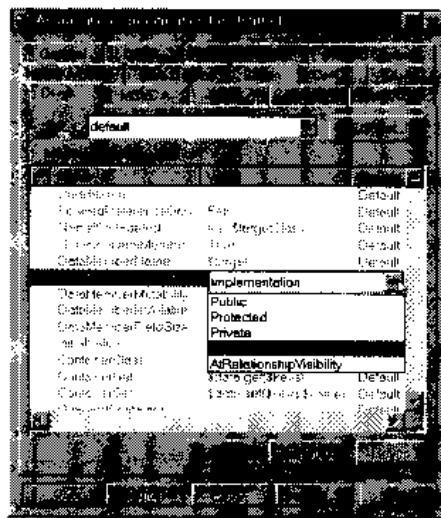
```
const Class_B * get_the_Class_B () const;
void set_the_Class_B (Class_B * value);
```

包括Class_B的Get和Set操作声明。生成代码时，Rose在Class_A中放上属性Class_B。由于每个属性产生Get和Set操作，而关联生成属性，因此关联属性生成Get和Set操作。

第三组标号行：

```
/**# Association: <unnamed>%3726887C02D0
    /**# begin Class_A::<the_Class_B>%3726887D0032.role preserve=nc
public: Class_B {1 -> 1RHN}
    Class_B *the_Class_B;
    /**# end Class_A::<the_Class_B>%3726887D0032.role
```

生成属性_Class_B，已在前面介绍。对每个关联，Rose生成相应属性。缺省情况下，属性名为_<class name>。要改变属性名，在关联上设置作用名。Rose用作用名生成属性名。每个生成属性缺省可见性为专用。要改变可见性，打开关联规范窗口，选择C++A或C++B标签，并改变DataMemberVisibility属性。



最后，第四组标号行包含_Class_B属性的Get和Set操作代码：

```
/**# Get and Set Operations for Associations (inline)
inline const Class_B * Class_A::get_the_Class_B () const
{
    /**# begin Class_A::get_the_Class_B%3726887D0032.get preserve=nc
    return the_Class_B;
    /**# end Class_A::get_the_Class_B%3726887D0032.get
}

inline void Class_A::set_the_Class_B (Class_B * value)
{
    /**# begin Class_A::set_the_Class_B%3726887D0032.set preserve=nc
    the_Class_B = value;
    /**# end Class_A::set_the_Class_B%3726887D0032.set
}
```

改变GenerateGetOperation和GenerateSetOperation代码生成属性可以控制这些操作是否生成打开关联规范窗口，选择C++A或C++B标签，浏览关系两端采用的代码生成属性。改变GenerateGetOperation和GenerateSetOperation属性可以控制这些操作是否生成。

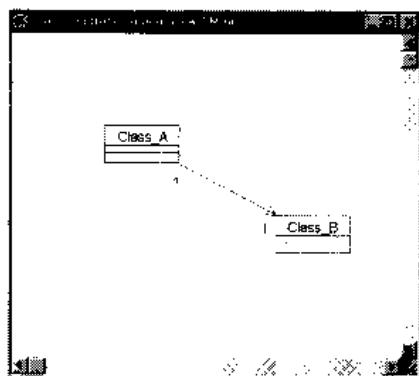
注意，这个关联加进一对一的倍增性。下面将介绍倍增性设置对代码生成的影响。

单向关联

和双向关联一样，Rose也生成支持单向关联的属性。但对于单向关联，只在关系一端生成属性。

本节介绍下列代码生成属性：

- DataMemberVisibility作用属性
- GenerateGetOperation作用属性
- GenerateSetOperation作用属性



对上述Class_A和Class_B类，生成如下代码：

```
Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    Class_B *the_Class_B;
};
```

和

```
Class Class_B
{
public:
    Class_B();
    ~Class_B();
};
```

可以看出，Rose只在关系一端生成专用属性。具体地说，是在Client类中生成属性，而不在Supplier类中生成。

在**Supplier**类中生成的代码包括上面双向关联中介绍的所有头文件行和实现文件行。对于双向关联，每个类提供一个新属性，两个类中都包括上节介绍的代码。而对于单向关联，则只在**Supplier**类中生成代码。

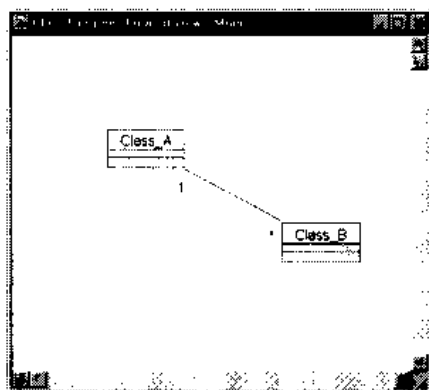
注意，这个关联加进一对一的倍增性。下面将介绍倍增性设置对代码生成影响。

倍增性为一对多的关联

在一对一关系中，**Rose**只是生成支持关联的相应属性。而对于一对多关系，一个类要包含其他类的设置。本节介绍下列代码生成属性：

- **UnorderedUnboundedByReferenceContainer**项目属性
- **ContainerClass**作用属性

下面举一个例子。



本例中，我们具有一对多关系。如图所示，**Class_B**只能生成指向**Class_A**的属性，而仅有**Class_A**类中的简单指针属性是不够的，**Class_A**类中生成的属性要用某种容器类作为数据类型。这个容器类可以是清单、集合或其他容器类。

Rose生成如下代码：

```

Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    UnboundedSetByReference<Class_B> the_Class_B;
};
  
```

和

```

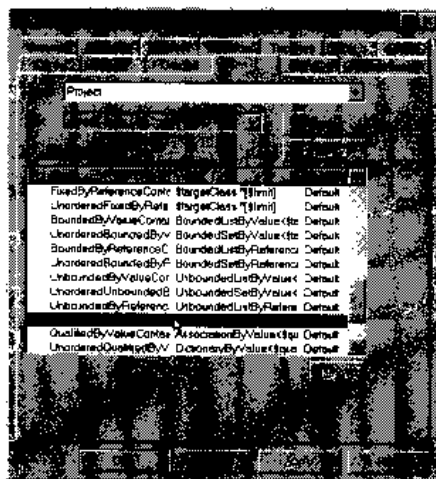
Class Class_B
{
public:
    Class_B();
    ~Class_B();
private:
    Class_A *the_Class_A;
};
  
```

可以看出, **Class_B**包含指向**Class_A**的简单指针, 但**Class_A**中生成**Class_B**属性时使用容器类。

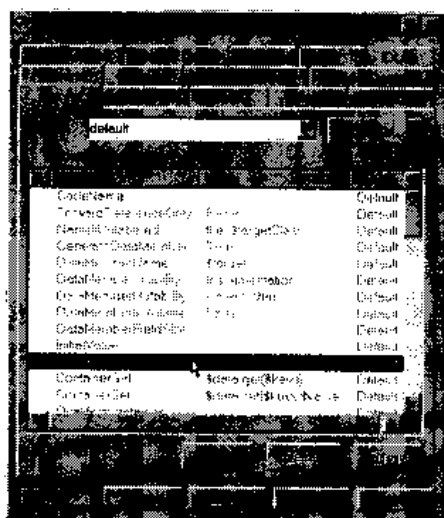
Rose缺省并不提供实现层容器类, 而是生成如上代码, 用**UnboundedSetByReference**作为容器类, 你要在C++代码生成属性中提供有效容器类。

要改变容器类, 可以设置二个属性, 两个是全局缺省, 一个是针对关系的。

要设置所有一对多关系的容器类, 从菜单中选择**Tools►Options**。在C++标签中选择下拉列表框内的**Project**。将**UnorderedUnboundedByReferenceContainer**属性值变为容器类设置。

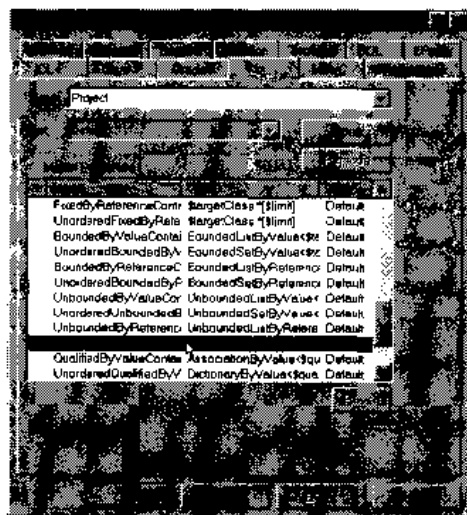


要改变一个关系的容器类, 打开关系规范窗口并选择C++A或C++B标签。改变**ContainerClass**属性, 反映要用的容器的类。



生成代码时, Rose首先检查该关系的**ContainerClass**属性。如果这个属性有内容, 则将其用作容器类。如果**ContainerClass**属性是空值, 则Rose检查**UnorderedUnboundedByReferenceContainer**属性。

第三个可以设置的属性是**Tools►Options**屏幕中的**ContainerClass**属性。在这个屏幕的C++标签中, 从下拉列表框选择**Role**。在此改变**ContainerClass**属性等价于在所有关系的关联规范窗口中改变**ContainerClass**属性。



下面是Class_A与Class_B具有双向、一对多关系时在Class_A类头文件中生成的部分代码。头文件中插入了四项内容：Class_B的#include语句，_Class_B属性，这个属性的Get和Set操作。

```
// Class_B
#include "Class_B.h"

///< Get and Set Operations for Associations (generated)

///< Association: <unnamed>%3726887C02D0
///< Role: Class_A::<the_Class_B>%3726887D0032
const UnboundedSetByReference<Class_B> get_the_Class_B () const;
void set_the_Class_B (UnboundedSetByReference<Class_B> value);

private: ///< implementation
    // Data Members for Associations
    ///< Association: <unnamed>%3726887C02D0
    ///< begin Class_A::<the_Class_B>%3726887D0032.role preserve=no
public: Class_B (1 -> nRHn)
    UnboundedSetByReference<Class_B> the_Class_B;
    ///< end Class_A::<the_Class_B>%3726887D0032.role

///< Get and Set Operations for Associations (inline)
inline const UnboundedSetByReference<Class_B>
    Class_A::get_the_Class_B () const
{
    ///< begin Class_A::get_the_Class_B%3726887D0032.get preserve=no
    return the_Class_B;
    ///< end Class_A::get_the_Class_B%3726887D0032.get
}

inline void Class_A::set_the_Class_B
    (UnboundedSetByReference<Class_B> value)
{
```

```

    /// begin Class_A::set_the_Class_B%3726887D0032.set preserve=no
    the_Class_B = value;
    /// end Class_A::set_the_Class_B%3726887D0032.set
}

```

在Class_B类的头文件中, 可以看出Class_A的指针作为属性, 并生成这个属性的Get和Set操作。由于Class_B中的属性不需要容器类, Class_B生成代码与上节介绍单向关联和双向关联时完全相同。Class_B头文件中与Class_A有关的部分如下:

```

// Class_A
#include "Class_A.h"

///## Get and Set Operations for Associations (generated)

    ///## Association: <unnamed>%3726887C02D0
    ///## Role: Class_B::<the_Class_A>%3726887D0064
    const Class_A * get_the_Class_A () const;
    void set_the_Class_A (Class_A * value);
private: ///## implementation
    // Data Members for Associations

    ///## Association: <unnamed>%3726887C02D0
    ///## begin Class_B::<the_Class_A>%3726887D0064.role preserve=no
    public: Class_A {n -> 1RHN}
    Class_A *the_Class_A;
    ///## end Class_B::<the_Class_A>%3726887D0064.role

    ///## Get and Set Operations for Associations (inline)

    inline const Class_A * Class_B::get_the_Class_A () const
    {
        ///## begin Class_B::get_the_Class_A%3726887D0064.get preserve=no
        return the_Class_A;
        ///## end Class_B::get_the_Class_A%3726887D0064.get
    }

    inline void Class_B::set_the_Class_A (Class_A * value)
    {
        ///## begin Class_B::set_the_Class_A%3726887D0064.set preserve=no
        the_Class_A = value;
        ///## end Class_B::set_the_Class_A%3726887D0064.set
    }

```

Class_B包含Class_A的简单指针, 而不是使用容器类。

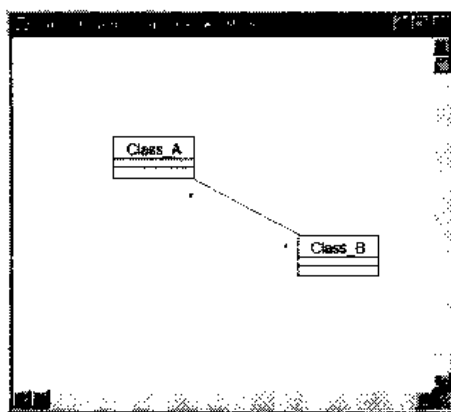
增殖性为多对多的关联

这里生成的代码与一对多关系相似, 但Rose在关系两端生成容器类。本节介绍的代码生成属性如下:

- UnorderedUnboundedByReferenceContainer项目属性

- ContainerClass角色属性

下面介绍下图关系的生成代码。



这里Rose在关系两端生成容器类，生成的代码如下：

```

Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    UnboundedSetByReference<Class_B> the_Class_B;
};
  
```

和

```

Class Class_B
{
public:
    Class_B();
    ~Class_B();
private:
    UnboundedSetByReference<Class_A> the_Class_A;
};
  
```

当然，这只是生成代码的简化形状。Rose插入说明语句并影响头文件中的几段。Class_A的代码与上节“倍增性为一对多的关联”中相同。这里的差别之外是Class_B中生成属性时也用容器类。

Class_B头文件中插入下列四段以支持这个关系：

```

// Class_A
#include "Class_A.h"

//## Get and Set Operations for Associations (generated)

//## Association: <unnamed>%3726887C02D0
//## Role: Class_B:.<the_Class_A>%3726887D0064
  
```



```
const UnboundedSetByReference<Class_A> get_the_Class_A () const;
void set_the_Class_A (UnboundedSetByReference<Class_A> value);

private: ///# implementation
    // Data Members for Associations

    ///# Association: <unnamed>%3726887C02D0
    ///# begin Class_B::<the_Class_A>%3726887D0064.role preserve=no
    public: Class_A {n -> nRHN}
    UnboundedSetByReference<Class_A> the_Class_A;
    ///# end Class_B::<the_Class_A>%3726887D0064.role

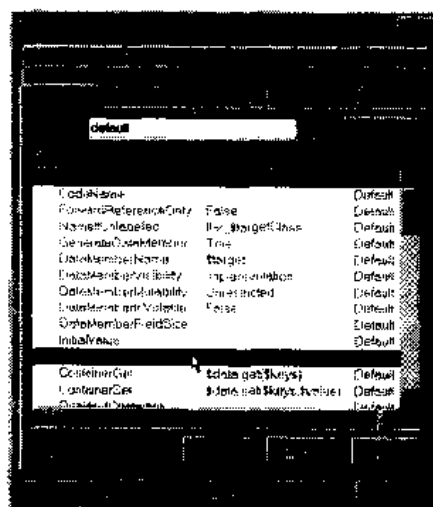
    ///# Get and Set Operations for Associations (inline)

    inline const UnboundedSetByReference<Class_A>
    Class_B::get_the_Class_A () const
    {
        ///# begin Class_B::get_the_Class_A%3726887D0064.get preserve=no
        return the_Class_A;
        ///# end Class_B::get_the_Class_A%3726887D0064.get
    }

    inline void Class_B::set_the_Class_A
    (UnboundedSetByReference<Class_A> value)
    {
        ///# begin Class_B::set_the_Class_A%3726887D0064.set preserve=no
        the_Class_A = value;
        ///# end Class_B::set_the_Class_A%3726887D0064.set
    }
}
```

和一对多关联一样，可以通过修改UnorderedUnboundedByReferenceContainer或ContainerClass属性改变多对多关系使用的容器类。

要显示ContainerClass属性，打开关联规范窗口并选择C++A或C++B标签（要修改的关系端）。在这个标签上，可以改变ContainerClass属性，包括要用的特定容器类。改变此处的属性只影响当前关系。



要改变所有关系的容器类，改变UnorderedUnboundedByReferenceContainer项目属性，或从选项屏幕中改变ContainerClass属性。选择Tools►Options，然后选择C++标签。从下拉列表框中选择Project并改变UnorderedUnboundedByReferenceContainer属性，或从下拉列表框中选择Role并改变ContainerClass属性。



如果ContainerClass属性有内容，则Rose使用这个值生成代码。如果ContainerClass属性为空值，则Rose寻找UnorderedUnboundedByReferenceContainer项目属性。

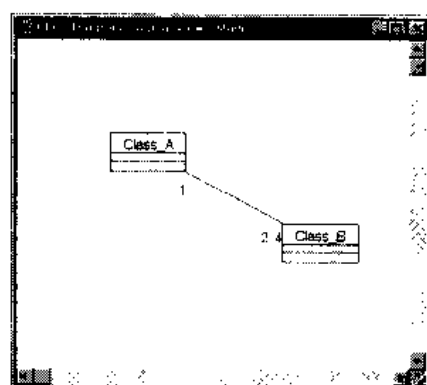
带约束倍增性的关联

前面介绍了一对一关系、一对多关系和多对多关系。“多”端的关系称为非约束关系。下面介绍约束关系。

这里介绍的代码生成属性包括：

- ContainerClass作用属性
- BoundedByReferenceContainer项目属性
- FixedByReferenceContainer项目属性

下图是个约束关系的例子：



本例中，Class_A的每个实例与2到4个Class_B实例相关。

这里要介绍两种约束关系：约束关联和固定关联。约束关联是具有倍增性范围为关联，如上例中的2..4，固定关联则是具有单个倍数的关联，如一对四关系。

这些关系和“多”关系一样，Rose用容器类生成属性。在Class_B中，可以生成Class_A的简单指针，但在Class_A中则使用某种容器类。

下面先介绍约束关联。上述Class_A和Class_B的简化代码如下：

```
Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    BoundedSetByReference<Class_B> the_Class_B;
};
```

和

```
Class Class_B
{
public:
    Class_B();
    ~Class_B();
private:
    Class_A *the_Class_A;
};
```

下面介绍这个关系实际生成的详细代码。前面曾介绍过，在Class_B中，可以生成Class_A的简单指针。这里生成的代码与单向和双向关联中相同。头文件包括下列段：

```
// Class_A
#include "Class_A.h"

///## Get and Set Operations for Associations (generated)

    ///## Association: <unnamed>%3726887C02D0
    ///## Role: Class_B::<the_Class_A>%3726887D0064
    const Class_A * get_the_Class_A () const;
    void set_the_Class_A (Class_A * value);

private: ///## implementation
    // Data Members for Associations

    ///## Association: <unnamed>%3726887C02D0
    ///## begin Class_B::<the_Class_A>%3726887D0064.role preserve=no
    public: Class_A {2..4 -> 1RHN}
    Class_A *the_Class_A;
    ///## end Class_B::<the_Class_A>%3726887D0064.role

    ///## Get and Set Operations for Associations (inline)

    inline const Class_A * Class_B::get_the_Class_A () const
    {
```

```

    /// begin Class_B::get_the_Class_A%3726887D0064.get preserve=no
    return the_Class_A;
    /// end Class_B::get_the_Class_A%3726887D0064.get
}

inline void Class_B::set_the_Class_A (Class_A * value)
{
    /// begin Class_B::set_the_Class_A%3726887D0064.set preserve=no
    the_Class_A = value;
    /// end Class_B::set_the_Class_A%3726887D0064.set
}

```

Class_A中对Class_B属性使用容器类。Class_A中引用Class_B的头文件代码段如下:

```

// Class_B
#include "Class_B.h"

///## Get and Set Operations for Associations (generated)
    ///## Association: <unnamed>%3726887C02D0
    ///## Role: Class_A::<the_Class_B>%3726887D0032
    const BoundedSetByReference<Class_B,4> get_the_Class_B () const;
    void set_the_Class_B (BoundedSetByReference<Class_B,4> value);

private: ///## implementation
    // Data Members for Associations
    ///## Association: <unnamed>%3726887C02D0
    ///## begin Class_A::<the_Class_B>%3726887D0032.role preserve=no
    public: Class_B (1 -> 2..4RHN)
    BoundedSetByReference<Class_B,4> the_Class_B;
    ///## end Class_A::<the_Class_B>%3726887D0032.role

    ///## Get and Set Operations for Associations (inline)

    inline const BoundedSetByReference<Class_B,4>
        Class_A::get_the_Class_B () const
    {
        ///## begin Class_A::get_the_Class_B%3726887D0032.get preserve=no
        return the_Class_B;
        ///## end Class_A::get_the_Class_B%3726887D0032.get
    }

    inline void Class_A::set_the_Class_B
        (BoundedSetByReference<Class_B,4> value)
    {
        ///## begin Class_A::set_the_Class_B%3726887D0032.set preserve=no
        the_Class_B = value;
        ///## end Class_A::set_the_Class_B%3726887D0032.set
    }

```

注意, 缺省情况下, Rose用BoundedSetByReference容器类。和前面一样, 要输入这些关系中使用的有效容器类。容器类可以在三个地方设置。

首先, 可以设置特定关系的容器类。为此, 打开关联规范窗口, 选择C++A或C++B标签并改变ContainerClass属性。

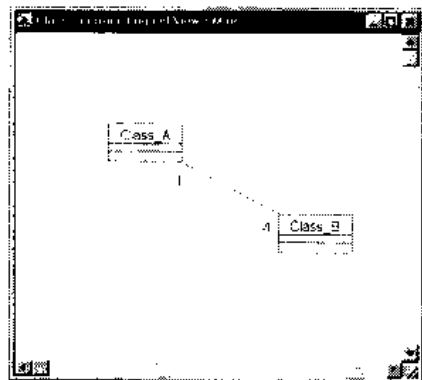
要设置所有关联关系的容器类, 选择Tools>Options, 然后选择C++标签, 从下拉列表框中选择Role并改变ContainerClass属性。

要设置所有约束关联关系的容器类, 选择Tools>Options, 然后选择C++标签, 从下拉列表框中选择Project并改变BoundedByReferenceContainer属性值。



生成代码时, Rose首先检查关系中的ContainerClass属性。如果这个属性是空的, 则使用Role属性的ContainerClass属性设置。如果这个属性也是空的, 则使用BoundedByReferenceContainer属性。

下面介绍固定关系。假设Class_A和Class_B之间的关系为一对四, 如下图。



Class_A头文件中的代码仍然对Class_B属性使用容器类。这里Rose缺省使用数组。

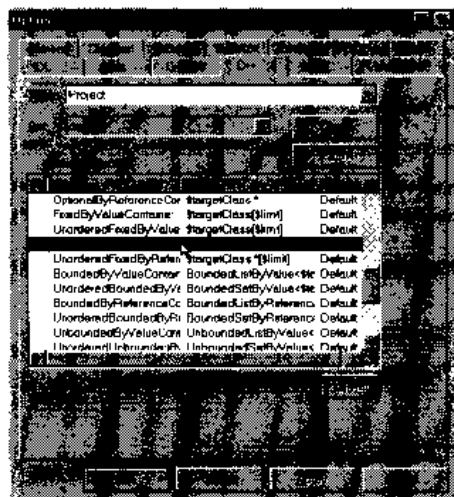
```
private: ///< implementation
    // Data Members for Associations

    ///< Association: <unnamed>%3726887C02D0
    ///< begin Class_A::<the_Class_B>%3726887D0032.role preserve=no
public: Class_B {1 -> 4RHN}
    Class_B *the_Class_B[4];
    ///< end Class_A::<the_Class_B>%3726887D0032.role
```

和前面一样,缺省值可以用几种方法改变。第一种只影响当前关系,是改变关系规范窗口C++标签的ContainerClass属性。

第二种方法是通过Tools>Options菜单改变ContainerClass属性。选择C++标签,然后选择Role,改变这个属性影响所有关联关系。

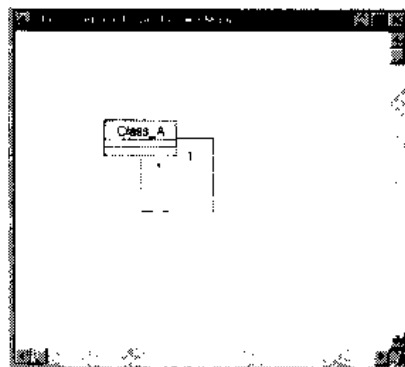
最后,可以改变FixedByReferenceContainer项目属性。选择Tools>Options,然后选择C++标签,并从下拉列表框选择Project,并改变FixedByReferenceContainer项目属性值。



反身关联

反身关联的处理方法与两个类之间的关联差不多。

对下列情形:



生成的代码如下:

```
class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    UnboundedSetByReference<Class_A> Class_A;
};
```

和普通关联一样，类中生成属性以支持关系。如果倍增性为一，则生成简单属性，如果倍增性为多，则使用容器类。

下面是上述关系所生成头文件的一部分：

```
//## Get and Set Operations for Associations (generated)

//## Association: <unnamed>%37268D26001E
//## Role: Class_A::<the_Class_A>%37268D2701CD
const Class_A * get_the_Class_A () const;
void set_the_Class_A (Class_A * value);

//## Association: <unnamed>%37268D26001E
//## Role: Class_A::<the_Class_A>%37268D2701CC
const UnboundedSetByReference<Class_A> get_the_Class_A () const;
void set_the_Class_A (UnboundedSetByReference<Class_A> value);

private: //## implementation
// Data Members for Associations

//## Association: <unnamed>%37268D26001E
//## begin Class_A::<the_Class_A>%37268D2701CD.role preserve=no
public: Class_A {n -> 1RHN}
Class_A *the_Class_A;
//## end Class_A::<the_Class_A>%37268D2701CD.role

//## Association: <unnamed>%37268D26001E
//## begin Class_A::<the_Class_A>%37268D2701CC.role preserve=no
public: Class_A {1 -> nRHN}
UnboundedSetByReference<Class_A> the_Class_A;
//## end Class_A::<the_Class_A>%37268D2701CC.role

//## Get and Set Operations for Associations (inline)

inline const Class_A * Class_A::get_the_Class_A () const
{
//## begin Class_A::get_the_Class_A%37268D2701CD.get preserve=no
return the_Class_A;
//## end Class_A::get_the_Class_A%37268D2701CD.get
}

inline void Class_A::set_the_Class_A (Class_A * value)
{
//## begin Class_A::set_the_Class_A%37268D2701CD.set preserve=no
the_Class_A = value;
//## end Class_A::set_the_Class_A%37268D2701CD.set
}

inline const UnboundedSetByReference<Class_A> Class_A::get_the_Class_A
() const
{
//## begin Class_A::get_the_Class_A%37268D2701CC.get preserve=no
```

```

return the_Class_A;
//## end Class_A::get_the_Class_A%37268D2701CC.get
}

inline void Class_A::set_the_Class_A
(UnboundedSetByReference<Class_A> value)
{
//## begin Class_A::set_the_Class_A%37268D2701CC.set preserve=nc
the_Class_A = value;
//## end Class_A::set_the_Class_A%37268D2701CC.set
}

```

可以看出，生成的代码与典型一对多关系中相似。这里Class_A包含Class_A类型的属性。

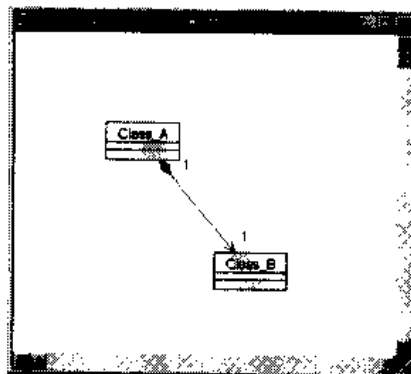
按值累积（复合关系）

累积关系有两种：按值和按引用。在按值关系中，一个类包含另一个类。在按引用关系中，一个类包含另一个类的引用。

下面介绍如下代码生成属性：

- ContainerClass作用属性
- FixedByValueContainer项目属性
- UnorderedBoundedByValueContainer项目属性

下面先介绍按值累积（复合关系）。在UML中，复合用下列符号表示：



和关联关系一样，Rose在生成累积代码时生成属性。本例中，Class_B是Class_A中的属性。生成的代码可以简化如下：

```

Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    Class_B the_Class_B;
};

```


和

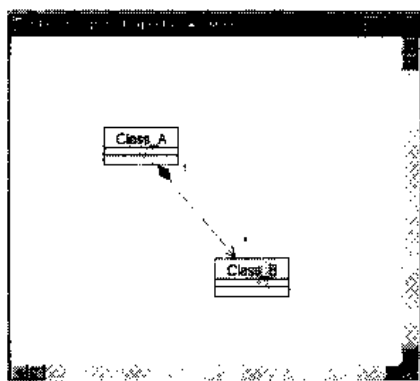
```
Class Class_B
{
public:
    Class_B();
    ~Class_B();
};
```

让我们看看这个关系生成的实际代码。在Class_A中，生成Class_B的属性：

```
private: ///# implementation
    // Data Members for Associations

    ///# Association: <unnamed>%37268F970050
    ///# begin Class_A::<the_Class_B>%37268F970316.role preserve=no
public: Class_B {1 -> 1VHN}
    Class_B the_Class_B;
    ///# end Class_A::<the_Class_B>%37268F970316.role
```

如果关系的倍数大于一，则Rose在生成关系时使用容器类。下面看这个关系：



下面是Class_A头文件中的代码：

```
private: ///# implementation
    // Data Members for Associations

    ///# Association: <unnamed>%37268F970050
    ///# begin Class_A::<the_Class_B>%37268F970316.role preserve=no
public: Class_B {1 -> nVHN}
    UnboundedSetByValue<Class_B> the_Class_B;
    ///# end Class_A::<the_Class_B>%37268F970316.role
```

这里Rose用UnboundedSetByValue作为缺省容器类。固定倍数后（如一对四关系），Rose可以用数组。而对于约束倍增性（如1对2.4关系），则Rose要用BoundedSetByValue容器。要改变使用的容器类，可以改变关系的ContainerClass属性。

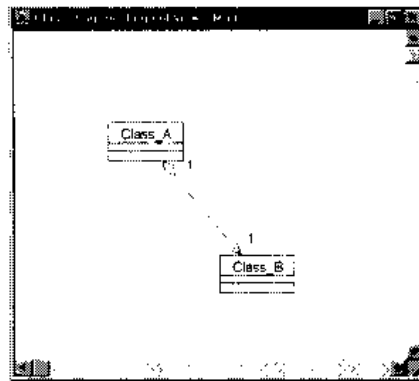
要改变所有固定累积的容器类，改变FixedByValueContainer项目属性。要改变所有约束累积的容器类，改变UnorderedBoundedByValueContainer项目属性。

按引用累积

按引用累积生成的代码与按值累积相似。但按引用累积中生成的属性是指针。
本节介绍下列代码生成属性：

- **ContainerClass**作用属性
- **FixedByReferenceContainer**项目属性
- **UnorderedBoundedByReferenceContainer**项目属性

例如，下面是前述关系的按引用累积版本：



这里仍是一对一关系，但采用按引用累积。下面是生成代码的简化版本：

```

Class Class_A
{
public:
    Class_A();
    ~Class_A();
private:
    Class_B *the_Class_B;
};
as is

Class Class_B
{
public:
    Class_B();
    ~Class_B();
};
  
```

Class_A头文件中生成的代码包括下列内容：

```

private: ///# implementation
    // Data Members for Associations

    ///# Association: <unnamed>%37268F970050
    ///# begin Class_A::<the_Class_B>%37268F970316.role preserve=no
public: Class_B {1 -> 1RHN}
→ Class_B *the_Class_B;
    ///# end Class_A::<the_Class_B>%37268F970316.role
  
```

这里在Class_A中生成的一个属性指针。和其他关系中一样，如果倍数大于一，则用容器类。如果倍数固定（例如一对四关系），则用数组：

```
Class_B *the_Class_B[4];
```

如果倍数约束（例如1对2..4关系），则Rose用“BoundedSetByReference”容器类：

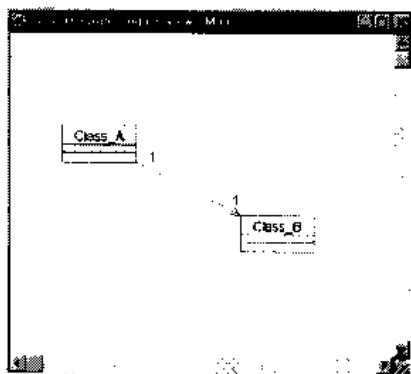
```
BoundedSetByReference<Class_B,4> the_Class_B;
```

要改变关系的容器类，可以在关系规范窗口中改变C++标签上的ContainerClass属性。

要改变所有固定按引用累积的容器类，改变FixedByReference属性。要改变所有固定按引用累积的容器类，改变UnorderedBoundedByReference项目属性。

依赖性关系

依赖性关系不生成属性。如果Class_A和Class_B之间有依赖性关系：



则Class_A和Class_B中不生成属性。生成的代码如下：

```
Class Class_A
{
public:
    Class_A();
    ~Class_A();
};
```

和

```
Class Class_B
{
public:
    Class_B();
    ~Class_B();
};
```

Rose只在Class_A中放Class_B的两个引用，一个引用Class_B.h，一个是说明，表示依赖性关系存在。Class_A头文件的这两个部分如下：

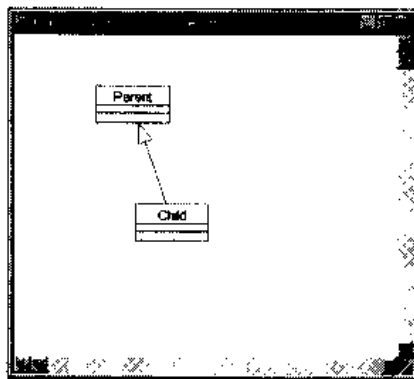
```
// Class_B
#include "Class_B.h"

and

///< Uses: <unnamed>%372690CD02E4;Class_B {1 -> 1}
```

一般化关系

UML一般化关系在C++中变成继承关系。在Rose模型中，继承关系显示如下：



对于一般化关系，Rose产生如下内容：

```
Class Parent
{
public:
    Parent();
    ~Parent();
};
```

和

```
Class Child : public Parent
{
public:
    Child();
    ~Child();
};
```

下面看看实际生成的代码。在父类代码中，没有子类的说明语句，使父类更一般化，可以派生许多子类而不影响父类代码中。

在子类中，生成支持从父类继承的代码。类声明如下：

```
class Child : public Parent ///< Inherits: <unnamed>%3729546E0028
{
    ///< begin Child%37295469024E.initialDeclarations preserve=yes
    ///< end Child%37295469024E.initialDeclarations
```

```

public:
    /// Constructors (generated)
    Child();

    Child(const Child &right);

    /// Destructor (generated)
    ~Child();

    /// Assignment Operation (generated)
    const Child & operator=(const Child &right);

    /// Equality Operations (generated)
    int operator==(const Child &right) const;

    int operator!=(const Child &right) const;

    // Additional Public Declarations
    /// begin Child%37295469024E.public preserve=yes
    /// end Child%37295469024E.public

protected:
    // Additional Protected Declarations
    /// begin Child%37295469024E.protected preserve=yes
    /// end Child%37295469024E.protected

private:
    // Additional Private Declarations
    /// begin Child%37295469024E.private preserve=yes
    /// end Child%37295469024E.private

private: /// implementation
    // Additional Implementation Declarations
    /// begin Child%37295469024E.implementation preserve=yes
    /// end Child%37295469024E.implementation

};

```

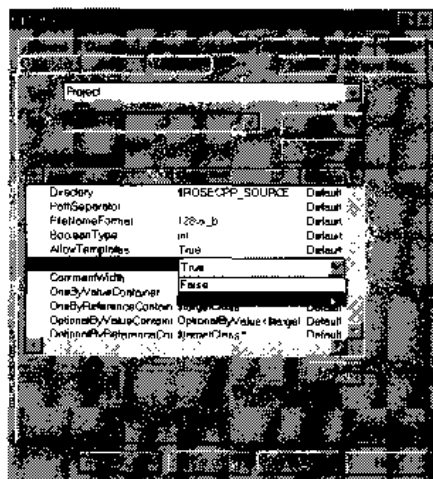
Rose在子类中生成#include行, 包括父类的头文件。本例中, 继承关系用公开继承实现。如果要使用保护继承, 则要打开关系规范窗口, 在General标签中设置要保护的关系的可见性。Rose生成如下子类:

```

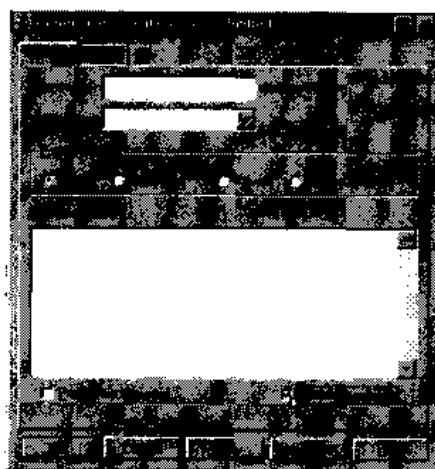
class Child : protected Parent    /// Inherits:
<unnamed>%3729546E0028

```

但这要求项目的AllowProtectedInheritance C++属性为真。选择Tools>Options, 再选C++标签, 从下拉列表框选择Project, 并保证项目的AllowProtectedInheritance C++属性为真。



要使用虚拟继承，则要打开关系规范窗口，在General标签中设置Virtual inheritance。

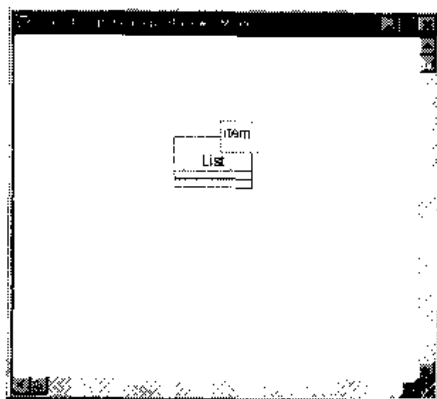


Rose生成虚拟继承的代码时，子类如下：

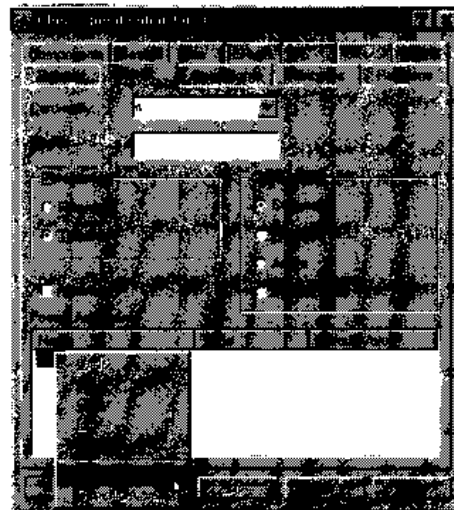
```
class Child : virtual public Parent //## Inherits:
```

参数化类

生成参数化类时，Rose在C++中生成模板类。例如对于下列类：



Rose生成模板类List。这个类的头文件如下。头文件中还包括对这个类生成的正式变元。要增加正式变元，打开类规范窗口并选择Detail标签。在标签底部的Formal Arguments区插入变元。



```

///<# begin module%372955190050.cm preserve=no
// %X% %Q% %Z% %W%
///<# end module%372955190050.cm

///<# begin module%372955190050.cp preserve=no
///<# end module%372955190050.cp

///<# Module: List%372955190050; Pseudo Package specification
///<# Source file: C:\Program Files\Rational\Rose 98i\C++\source\List.h

#ifndef List_h
#define List_h 1

///<# begin module%372955190050.additionalIncludes preserve=no
///<# end module%372955190050.additionalIncludes

///<# begin module%372955190050.includes preserve=yes
///<# end module%372955190050.includes

///<# begin module%372955190050.additionalDeclarations preserve=yes
///<# end module%372955190050.additionalDeclarations

///<# begin List%372955190050.preface preserve=yes
///<# end List%372955190050.preface

///<# Class: List%372955190050; Parameterized Class Utility
///<# Category: <Top Level>
///<# Persistence: Transient
///<# Cardinality/Multiplicity: n

```

```

template <int Item>
class List
{
    /// begin List%372955190050.initialDeclarations preserve=yes
    /// end List%372955190050.initialDeclarations

    public:
        // Additional Public Declarations
        /// begin List%372955190050.public preserve=yes
        /// end List%372955190050.public

    protected:
        // Additional Protected Declarations
        /// begin List%372955190050.protected preserve=yes
        /// end List%372955190050.protected

    private:
        // Additional Private Declarations
        /// begin List%372955190050.private preserve=yes
        /// end List%372955190050.private

    private: /// implementation
        // Additional Implementation Declarations
        /// begin List%372955190050.implementation preserve=yes
        /// end List%372955190050.implementation
};

/// begin List%372955190050.postscript preserve=yes
/// end List%372955190050.postscript

// Parameterized Class Utility List

/// begin module%372955190050.epilog preserve=yes
/// end module%372955190050.epilog

#endif

```

生成C++举例

本书多次提到ATM例子应用程序。本节生成ATM例子的框架C++代码，其组件模型见图12.1和图12.2。

对于ATM Client Component，将所有组件的语言设置为C++。然后选择Main Component框图中的ATMServer和ATMClient包，再选择Tools►C++►Code Generation。本书选配光盘中包括了每个组件生成的代码。

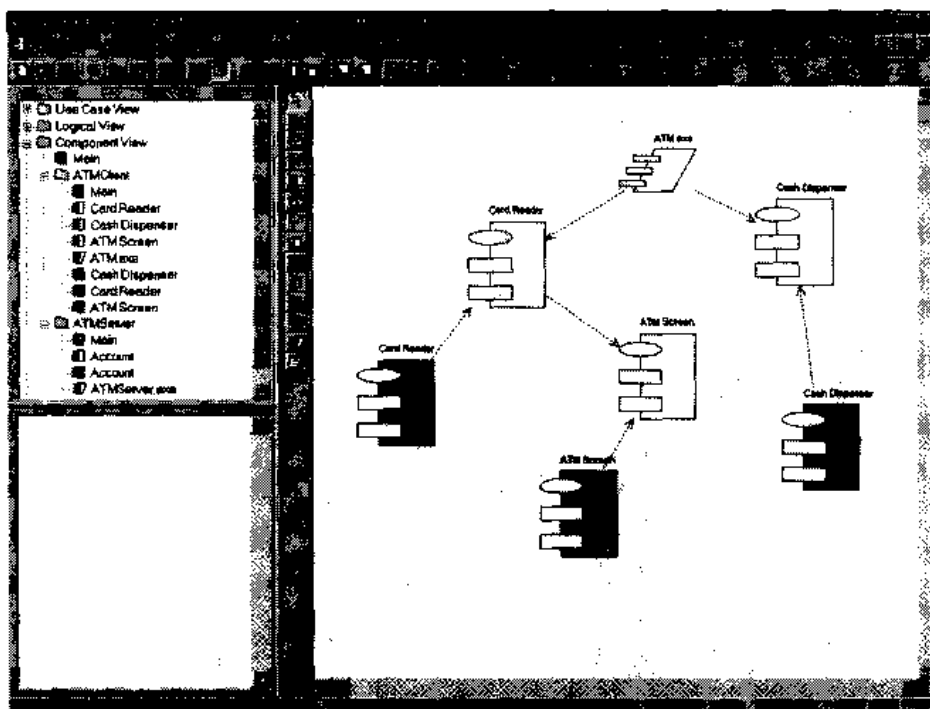


图12.1 ATM Client Component模型

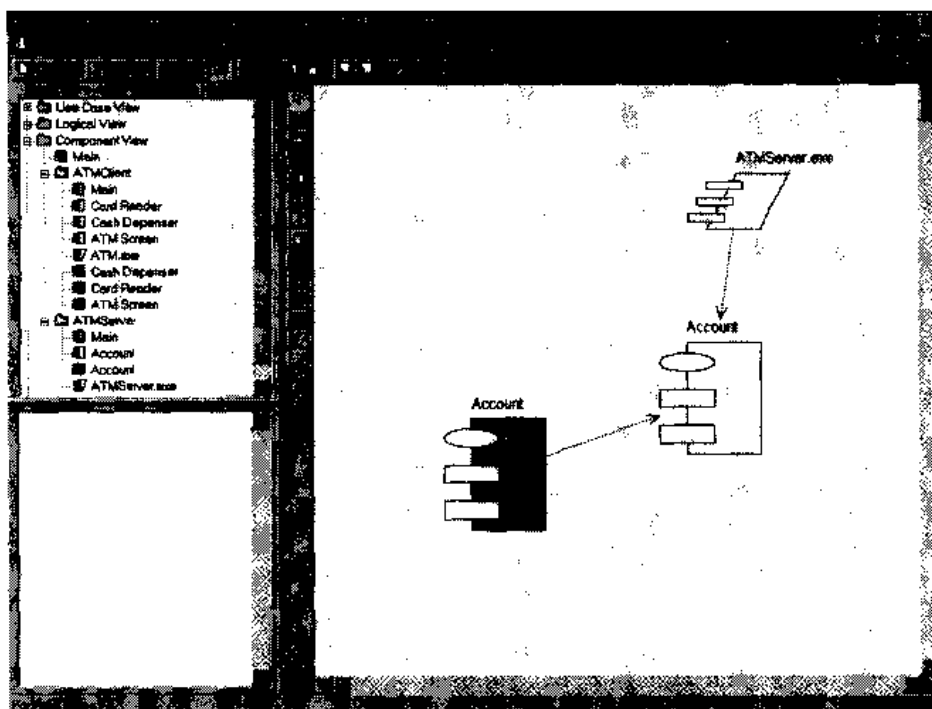


图12.2 ATM Server Component模型

练习

第3到第10章完成了订单输入系统的模型。现在要生成订单输入系统的代码。我们用图12.3所示的System Component框图。要生成代码，步骤如下，本书选配光盘中包括了本练习的代码。

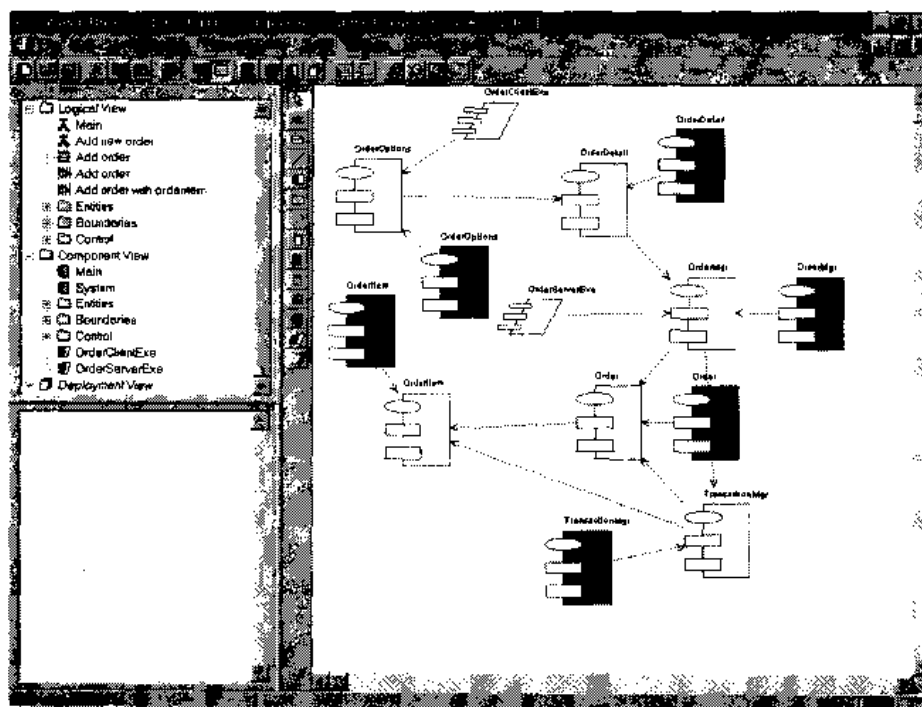


图12.3 订单输入系统的System Component框图

练习步骤

将包体加进System Component框图

1. 打开System Component框图。
2. 选择浏览器中的Entities: Order Package body。
3. 将订单包体拖动到System Component框图中。
4. 对下列组件重复第2和第3步:
 - Entities: OrderItem Package Body
 - Boundaries: OrderOptions Package Body
 - Boundaries: OrderDetail Package Body
 - Control: TransactionMgr Package Body
 - Control: OrderMgr Package Body

将语言设置为C++

1. 打开实体组件包中Order组件的组件规范窗口（包规范）。
2. 选择语言为C++。
3. 对下列组件重复第1到第2步:
 - Entities: Order Package Body
 - Entities: OrderItem Package Specification
 - Entities: OrderItem Package Body
 - Boundaries: OrderOptions Package Specification

- Boundaries: OrderOptions Package Body
- Boundaries: OrderDetail Package Specification
- Boundaries: OrderDetail Package Body
- Control: TransactionMgr Package Specification
- Control: TransactionMgr Package Body
- Control: OrderMgr Package Specification
- Control: OrderMgr Package Body
- OrderClientExe Task Specification
- OrderServerExe Task Specification

生成C++代码

1. 打开System Component框图。
2. 选择System Component框图中的所有对象。
3. 从菜单中选择Tools>C++>Code Generation。

小结

本章介绍了各个Rose模型元素如何用C++实现。利用类、包、属性、操作、关联、累加和其他Rose模型元素的代码生成属性，可以很好地控制生成的代码。

生成代码的步骤如下：

1. 生成组件。
2. 将类赋予组件。
3. 选择代码生成属性。
4. 选择Class或Component框图中要生成的类和组件。
5. 选择Tools>C++>Code Generation。
6. 选择Tools>C++>Browse Header或Browse Body浏览生成的代码。

第13章 Java代码生成

- 设置Java代码生成属性
- 从Rose模型生成Java代码
- 将Rose模型映射到Java结构

本章介绍如何从Rational Rose模型生成Java代码。我们先介绍Java可设置的代码生成属性，然后介绍每个Rose模型元素如何在代码中实现。

要生成代码，步骤如下：

1. 生成组件（见第10章）
2. 将类赋予组件（见第10章）。
3. 设置代码生成属性（可选）。
4. 选择Class或Component框图中要生成的类或组件。
5. 选择Tools►Java►Code Generation。
6. 选择Tools►Java►Browse Java Source浏览生成的代码。

Rose从模型中取大量信息以生成代码。例如，它寻找每个关系的倍增性、作用名、包容和其他细节。Rose从各种规范窗口输入的模型元素信息中收集生成代码所需的信息。

Java代码生成属性

Java代码生成属性可以在两处设置。缺省用Tools►Options菜单项目并选择Java标签设置，如图13.1。可以设置下列项目的属性：

- Attributes（属性）
- Classes（类）
- Module bodies（模块体）
- Module specifications（模块规范）
- Operations（操作）
- Projects（项目）
- Roles（作用）

本章介绍这些属性，首先简要介绍各种代码生成属性，然后介绍项目、模块、类、属性、操作与作用，最后一步一步介绍ATM系统的Java代码生成，还有从订单输入模型生成Java代码的练习。

项目属性

项目属性影响Rose中Java代码生成的各个方面。这里列出项目属性及其常见设置。

项目属性是适用于整个项目而不是各个具体模型元素（如类或关系）的代码生成属性。

本节的选项包括生成代码时使用的缺省目录、使用的文件扩展名、以及发生错误时是否停止生成代码。表13.1列出各个项目属性及其用途和缺省值。

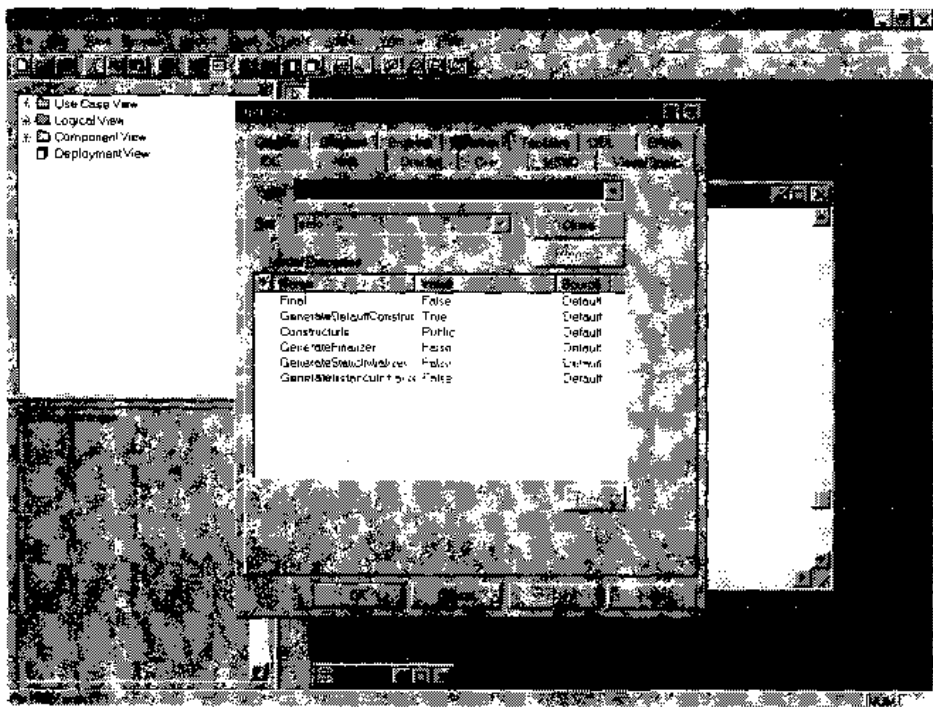


图13.1 Java代码生成属性窗口

在本表和后面的代码生成属性表中，我们列出最常用的属性。Rose 98与98i中的属性稍有不同。

表13.1 Java项目属性

属性	用途	缺省值
CreateMissingDirectories	如果为真，则生成需要而不存在的任何目录	缺省情况下，Rose生成缺失目录
StopOnError	如果为真，则Rose遇到第一个错误时即停止生成代码	缺省情况下，Rose遇到错误时即停止生成代码
Editor (98i)	选择浏览代码时使用的编辑器	缺省为Rose内置编辑器
VM (98i)	选择Java虚拟机版本（Sun或Microsoft）	缺省VM为Sun
ClassPath (98i)	指定生成代码的目录	缺省情况下，Rose在当前目录中生成代码
Directory (98)	指定生成代码的目录	缺省情况下，Rose在当前目录中生成代码
UsePrefixes	如果为真，则Rose在变量名前面上加上用户定义前缀	缺省情况下，Rose在类变量和实例前加上前缀“the”如果为真，则Rose在变量名前面上加上用户定义前缀
InstanceVariablePrefix	如果UsePrefixes为真，则在所有实例变量前加上这个前缀	缺省实例变量前缀为m_

(续表)

属性	用途	缺省值
ClassVariablePrefix	如果UsePrefixes为真, 则在所有类变量前加上这个前缀	缺省类变量前缀为's_
DefaultAttributeDataType	如果属性没有选择数据类型, 则用这个类型	缺省属性数据类型为整形
DefaultOperationReturnType	如果操作没有选择返回类型, 则用这个类型	缺省操作返回类型为void

生成代码时, Rose自动生成某种关系的属性。例如, 如果Client与Supplier类之间具有单向关联, 而这个关系的倍增性为一对一, 则Rose在Client内生成Supplier类型的属性。

类属性

本节介绍适用于类的Java代码生成属性。这些属性可以改变类名, 确定是否对类生成构造器和设置其他类特定属性。

这些属性可以在两处设置。要对所有类设置类属性, 选择Tools►Options, 然后选择Java标签, 并从下拉列表框选择Class。如果要对一个类设置类属性, 则要选择类规范窗口的Java标签并在此编辑属性。

表13.2列出了Java类属性及其用途与缺省值。

表13.2 Java类属性

属性	用途	缺省值
Final	在生成代码中包括final修饰符	缺省不包括final修饰符
Static (98i)	将嵌套Java类声明为静态, 只能有一个类实例	false
GenerateDefaultConstructor	控制是否自动对类生成构造器	缺省生成构造器
ConstructorIs	设置构造器的可见性 (public, private, protected)	缺省设置为Public
GenerateFinalizer	在类中包括最后化器	缺省不包括最后化器
GenerateStaticInitializer	在类中包括静态初始化器	缺省不包括静态初始化器
GenerateInstanceInitializer	在类中包括实例初始化器	缺省不包括实例初始化器

属性的属性

本节介绍与属性相关的代码生成属性。这些属性可以确定代码中是否生成属性等。

这些属性可以在两处设置。要对所有属性设置属性的属性, 选择Tools►Options, 然后选择Java标签, 并从下拉列表框选择Attribute。如果要对一个属性设置属性的属性, 则要选择属性规范窗口的Java标签并在此编辑属性。

表13.3列出了Java属性属性及其用途与缺省值。

表13.3 Java属性的属性

属性	用途	缺省值
GenerateDataMember (98)	控制是否生成属性的成员变量	缺省生成每个属性
Final	在属性中包括最后化器	缺省不包括最后化器
Transient	在属性中包括临时修饰符	缺省不包括临时修饰符
Volatile	在属性中包括易失修饰符	缺省不包括易失修饰符
PropertyType (98i)	指定Java bean的属性类型	非属性
IndividualChangeMgt (98i)	指定Java bean是否有自己的注册机制	false
Read/Write (98i)	设置Rose是否生成Get/Set方法	读写

操作属性

下面介绍与操作相关的代码生成属性。这些属性可以确定操作是否抽象。

这些属性可以在两处设置。要对所有操作设置属性，选择Tools►Options，然后选择Java标签，并从下拉列表框选择Operation。如果要对一个操作设置操作的属性，则要选择操作规范窗口的Java标签并在此编辑属性。

表13.4列出了Java操作属性及其用途与缺省值。

表13.4 Java操作属性

属性	用途	缺省值
Abstract	在操作中包括抽象修饰符	缺省不包括抽象修饰符
Static	在操作中包括静态修饰符	缺省不包括静态修饰符
Final	在操作中包括最后修饰符	缺省不包括最后修饰符
Native	在操作中包括自然修饰符	缺省不包括自然修饰符
Synchronized	在操作中包括同步修饰符	缺省不包括同步修饰符

模块属性

模块属性和体属性与从Rose生成的文件相关。这些属性可以确定是否在文件中包括版权声明等。

这些属性可以在两处设置。要对所有头文件设置头文件的属性，选择Tools►Options，然后选择Java标签，并从下拉列表框选择Module Specification。如果要对一个头文件设置头文件的属性，则要选择模块规范窗口的Java标签并在此编辑属性。

表13.5列出了Java模块属性、其用途与缺省值。

表13.5 Java模块属性

属性	用途	缺省值
Generate (98)	如果为真，则模块规范生成代码	缺省情况下，模块规范生成代码
CMIdentification	指定用户定义配置管理标识字符串	缺省用户定义配置管理标识字符串为空
CopyrightNotice	指定作为代码中说明语句的用户定义版权字符串	缺省版权字符串为空
AdditionalImports (98)	指定生成代码中要包括的其他输入语句	缺省不在代码中包括其他输入语句

在Rose 98i中, Java组件的组件规范窗口还显示CMIdentification和CopyrightNotice属性值。缺省属性值在组件规范字段中提供。

作用属性

作用属性是影响关系代码生成的代码生成属性。利用这些属性可以设置属性使用的容器类和改变作用代码生成的其他细节。

这些属性可以在两处设置。要对所有关系设置关系的属性, 选择Tools►Options, 然后选择Java标签, 并从下拉列表框选择Role。如果要对一个头文件设置头文件的属性, 则要选择作用规范窗口的Java标签并在此编辑属性。

表13.6列出了Java使用属性及其用途与缺省值。

表13.6 Java作用属性

属性	用途	缺省值
GenerateDataMember (98)	控制是否对关系生成属性	true
ContainerClass	指定关系倍增性大于1时使用的容器	缺省用数组
InitialValue	指定属性初始值	缺省不用属性初始值
Final	在属性中包括最后化器	缺省不包括最后化器
Transient	在属性中包括临时修饰符	缺省不包括临时修饰符
Volatile	在属性中包括易失修饰符	缺省不包括易失修饰符
PropertyType (98i)	指定Java bean的属性类型	非属性
IndividualChangeMgt (98i)	指定Java bean是否有自己的注册机制	false
Read/Write (98i)	设置Rose是否生成Get/Set方法	读写

生成代码

下面几节介绍类、属性、操作与不同类间关系的Java代码生成方法。每节都列出一些样本代码, 以便了解Rose模型生成的内容。

Rose用模型元素中的规范信息生成代码。例如, 它用类规范(可见性、属性、操作等)生成类代码。

下面先介绍典型类的代码生成。

类

生成代码时, 对象模型中的类变成Java类。类的所有属性、操作和关系均在生成代码中体现。每个类生成的主要元素包括:

- 类名
- 类可见性
- 类构造器
- 类文档
- 属性
- 操作
- 关系

如果没有组件映射，则模型中的每个类生成一个扩展名为.java的文件。每个文件用类名命名。例如，Employee类生成Employee.java文件。

Rose模型中的大多数信息在代码生成时直接采用。例如，每个类的属性、操作、关系和类名直接影响代码生成。其他模型属性（如输入类文档）不直接影响代码，而是成为生成代码中的说明语句。

表13.7列出了类规范窗口（和Rose 98i标准规范窗口）中的属性，并注意哪些属性直接影响代码生成。

表13.7 类规范对生成代码的影响

属性	对生成代码的影响
Name	模型中的名称成为类名
Type	直接影响生成的类类型
Stereotype	说明语句
Export Control	直接影响代码可见性
Documentation	说明语句
Persistence	影响类是否生成DDL
Abstract	生成抽象类
Formal Arguments	参数化类的代码中包括
Operations	在代码中生成正式变元
Attributes	在代码中生成
Relationships	在代码中生成

下面看看下几页所示类的代码。

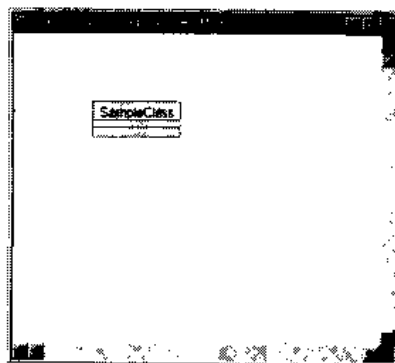
下列代码是这个类的Java文件：

```
// Source file: SampleClass.java

/*
Copyright Notice
*/

/**
SampleClass documentation
*/

public class SampleClass {
    public SampleClass() {
    }
}
```



缺省情况下，Rose生成带有公开构造器的声明。这与Rose生成的C++代码迥然不同。Java不生成操作符赋值、备份构造器、删除器、Get和Set，但Rose生成抽象类的修饰符。

版权声明段

版权声明段包括下列内容：

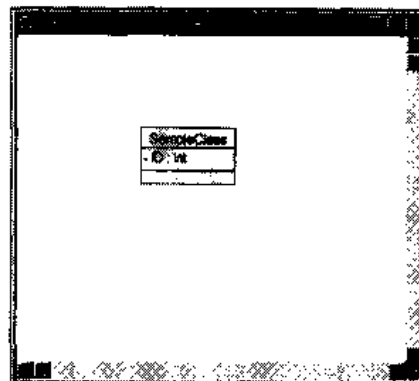
```
/*  
Copyright Notice  
*/
```

缺省情况下，代码不生成版权声明。但如果要在所有文件中增加版权声明段，可以改变代码生成属性如下：选择菜单中的Tools►Options，然后选择Java标签，从下拉列表框选择Module Specification打开模块规范代码生成属性。将CopyrightNotice字段变成包括任何版权信息。输入信息后，即可生成版权声明段。

如果要覆盖特定组件的缺省值，可以改变组件规范窗口的Copyright字段值或组件标准规范窗口Java标签的CopyrightNotice字段值。

属性

除类本身外，Rose还生成类的属性。对于每个属性，Rose在代码中放上属性可见性、数据类型和缺省值信息。看看下图所示类生成的代码：



```
// Source file: SampleClass.java  
  
/*  
Copyright Notice  
*/  
  
/**  
SampleClass documentation  
*/  
public class SampleClass {  
    private int ID;  
  
    public SampleClass() {  
    }  
}
```

可以看出，代码包括属性可见性、数据类型和缺省值信息。这只是属性的部分属性。可以阻止属性生成或对象属性采用**final**、**transient**或**volatile**修饰符。下面列出属性的属性表。

属性	用途	缺省值
GenerateDataMember	控制是否生成属性的成员交互	缺省生成每个属性
Final	在属性中包括最后化器	缺省不包括最后化器
Transient	在属性中包括临时修饰符	缺省不包括临时修饰符
Volatile	在属性中包括易失修饰符	缺省不包括易失修饰符

要阻止属性生成，改变属性的**GenerateDataMember**代码生成属性。在属性规范窗口，选择**Java**标签，并将**GenerateDataMember**属性变为**False**。

要对属性采用某个修饰符，如临时修饰符，将相应属性设置为**True**。这里将临时属性设置为**True**在属性中加上临时修饰符如下：

```
// Source file: SampleClass.java
/*
Copyright Notice
*/
/**
SampleClass documentation
*/
public class SampleClass {
→   private transient int ID;

    SampleClass() {
    }
}
```

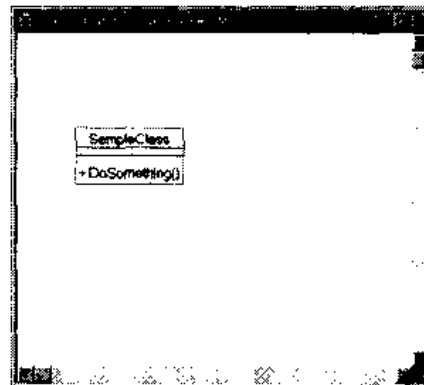
可见性选项影响生成的属性。上例中（缺省）采用专用可见性。如果将属性设置为**protected**或**public**，则会得到下列代码：

```
// Source file: SampleClass.java
/*
Copyright Notice
*/
/**
SampleClass documentation
*/
public class SampleClass {
→   private int ID;
→   public String Name;
→   protected String SSN;

    SampleClass() {
    }
}
```

操作

Rose生成类中每个操作。对于每个操作，生成的代码包括操作名、参数、参数数据类型和返回类型。下面对下图所要类生成代码：



```
//Source file: SampleClass.java

/**
 * SampleClass documentation
 */
public class SampleClass {
    private transient int ID;
    public String Name;
    protected String SSN;
    public SampleClass() {}

    /**
     * DoSomething documentation
     * @roseuid 372E31F800BF
     */
    public int DoSomething(int Parameter1) {}
}
```

可以看出，代码中生成整个操作签名。输入的操作文档成为说明语句。

Rose生成操作的框架代码，即生成整个操作签名。生成代码后，要插入每个操作的实现代码，放在{}之间，使逆向转出工程代码期间代码得到保护。

如果用操作规范窗口**Detail**标签的**Exceptions**字段输入操作的异常信息，则这些异常出现在代码中如下：

```
//Source file: SampleClass.java

/**
 * SampleClass documentation
 */
public class SampleClass {
    private transient int ID;
```

```

public String Name;
protected String SSN;

public SampleClass() {}

/**
    DoSomething documentation
    @roseuid 372E31F800BF
*/
→ public int DoSomething(int Parameter1) throws Exception {}
    }

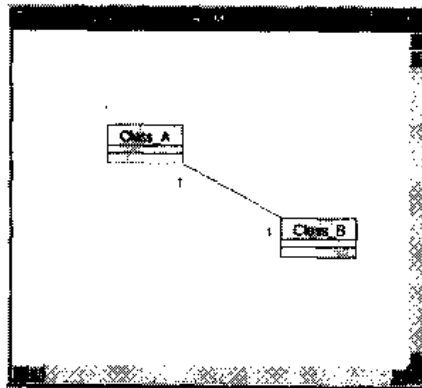
```

和其他模型元素一样，可以修改代码生成属性以控制操作代码的生成。表13.4列出了操作的代码生成属性，下面再次列出以供参考：

属性	用途	缺省值
Abstract	在操作中包括抽象修饰符	缺省不包括抽象修饰符
Static	在操作中包括静态修饰符	缺省不包括静态修饰符
Final	在操作中包括最后修饰符	缺省不包括最后修饰符
Native	在操作中包括自然修饰符	缺省不包括自然修饰符
Synchronized	在操作中包括同步修饰符	缺省不包括同步修饰符

双向关联

要支持双向关联，Rose在代码中生成属性。关系中的每个类包含一个支持关联的属性。缺省情况下，Rose 98i在类名前面加上“the”命名作用。



A类的代码如下：

```

//Source file: Class_A.java

public class Class_A {
→ public Class_B theClass_B;
    public Class_A() {}
}

```

B类的代码如下：

```
//Source file: Class_B.java  
  
public class Class_B {  
→ public Class_A theClass_A;  
  
    public Class_B() {}  
}
```

可以看出, **Rose**在双向关联关系的两端自动生成属性。利用 **_Class_B**属性, **Class_A**可以方便地访问**Class_B**。利用 **_Class_A**属性, **Class_B**可以方便地访问**Class_A**。如果属性用不同名称而不用**Class_A**和**Class_B**, 则可以设置关联的作用名。代码中使用这些名称如下:

Class_A代码:

```
//Source file: Class_A.java  
  
public class Class_A {  
→ public Class_B Supplier_to_A;  
  
    public Class_A() {}  
}
```

Class B代码:

```
//Source file: Class_B.java  
  
public class Class_B {  
→ public Class_A Supplier_to_B;  
  
    public Class_B() {}  
}
```

Rose用作用名生成属性名。缺省情况下, 每个生成属性的可见性为**Public**。要改变可见性, 打开关联规范窗口, 选择**Role A General**或**Role B General**标签, 并改变输出控制属性。

注意, 这个关联的倍增性为一对一。下面会介绍倍增性对代码生成的影响。

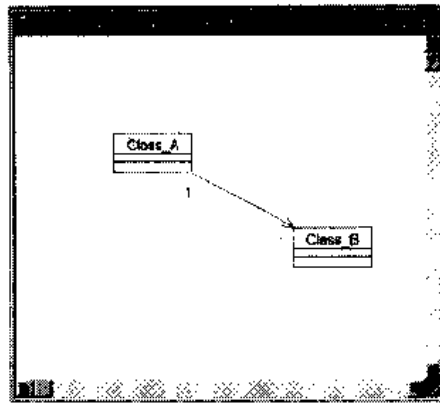
每个生成属性的可见性为**Public**。如果在关联中增加作用名, 则**Rose**用作用名生成属性名, 否则生成属性用项目属性**InstanceVariablePrefix**作为前缀。缺省情况下, **Rose 98**中为**m_**加上类名, **Rose 98i**中为**the**加上类名。例如**the Employee**。

除了指定可见性外, 还可以指定生成属性的初始值。打开所要关联的规范窗口, 选择**Java A**或**Java B**标签, 将**InitialValue**属性修改成包含生成属性的初始值。

注意, 这个关联的倍增性为一对一。下面会介绍倍增性对代码生成的影响。

单向关联

和双向关联一样, **Rose**也生成支持单向关联的属性。但对于单向关联, 只在关系一端生成属性。



对上面的A和B类，生成如下代码：

Class A代码：

```
//Source file: Class_A.java

public class Class_A {
    public Class_B theClass_B;
    public Class_A() {}
}
```

Class B代码：

```
//Source file: Class_B.java

public class Class_B {
    public Class_B() {}
}
```

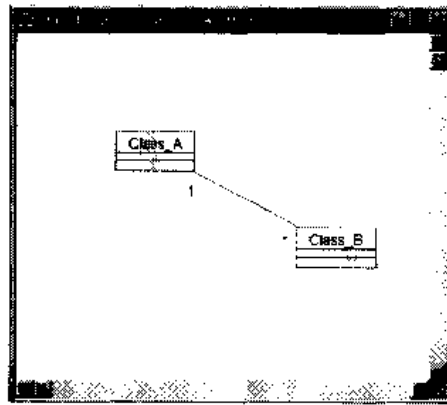
可以看出，Rose只在关系一端生成专用属性。具体地说，是在Client类中生成属性，而不在Supplier类中生成。

在Supplier类中生成的代码包括上面双向关联中介绍的所有头和实现文件行。对于双向关联，每个类提供一个新属性，两个类中都包括上节介绍的代码。而对于单向关联，则只在Supplier类中生成代码。

注意，这个关联加进一对一的倍增性。下面将介绍倍增性设置对代码生成影响。

倍增性为一对多的关联

在一对一关系中，Rose只是生成支持关联的相应属性。而对于一对多关系，一个类要包含其他类的设置。下面举例说明。



本例中是一对多关系。这个关系生成的代码如下：

Class A代码：

```

//Source file: Class_A.java

public class Class_A {
    public Class_B theClass_B[];
    public Class_A() {}
}

```

Class B代码：

```

//Source file: Class_B.java

public class Class_B {
    public Class_A theClass_A;
    public Class_B() {}
}

```

这里B类只是包含一个A类型的属性，因为倍增性为每个B实例对应一个A实例，但每个A实例又对应多个B实例，因而A中生成B对象的数组。

如果不用数组，则可以将代码生成属性变成使用不同的容器类。在关联规范窗口的Java标签中，将ContainerClass代码生成属性设置为要用的容器类名。例如，下列代码用于上述关联，但容器类为Container类型：

Class A代码：

```

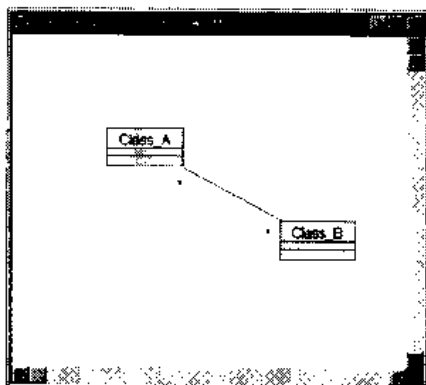
// Source file: A.java

public class Class_A {
    public Container theClass_B;
    public Class_A() {
    }
}

```


倍增性为多对多的关联

这里生成的代码与一对多关系中相似，但这里Rose在关系两端都生成数组。下面介绍下图所示关系的代码。



Class A代码:

```
//Source file: Class_A.java
```

```
public class Class_A {  
→     public Class_B theClass_B[];  
     public Class_A() {}  
}
```

Class B代码:

```
//Source file: Class_B.java
```

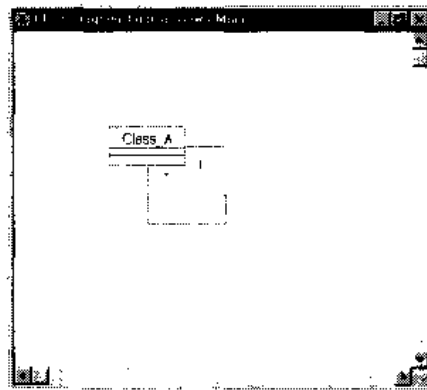
```
public class Class_B {  
→     public Class_A theClass_A[];  
     public Class_B() {}  
}
```

在这个情形中，Rose在关系两端都生成数组。缺省使用数组，但可以改变成用容器类。为此，使用关联的关系规范窗口，在Java A或Java B标签中，改变ContainerClass属性。将这个属性设置为要用的容器类名。

要改变所有多对多关联关系的容器类，从菜单中选择Tools►Options，在Java标签中，从下拉列表框选择Role。将ContainerClass代码生成属性值变为要用的容器类名。

反身关联

反身关联与两个类之间的关联用相似方法处理。下面是下图情形的代码:



//Source file: Class_A.java

```

public class Class_A {
    →      public Class_A theClass_A[];
    public Class_A() {}
}
  
```

和普通关联一样，类中生成属性以支持关系。如果倍增性为 -，则生成简单属性，如果倍增性为多，则使用容器类。

累积

累积关系有两种：按值和按引用。在按值关系中，一个类包含另一个类。在按引用关系中，一个类包含另一个类的引用。Java中两种累积生成相同的代码。

这里要介绍的代码生成属性为ContainerClass作用属性。

下面先介绍按值累积（复合关系）。在UML中，复合用下列符号表示：

和关联关系一样，Rose在生成累积代码时生成属性。本例中，Class_B是Class_A中的属性。生成的代码可以简化如下：

Class A:

//Source file: Class_A.java

```

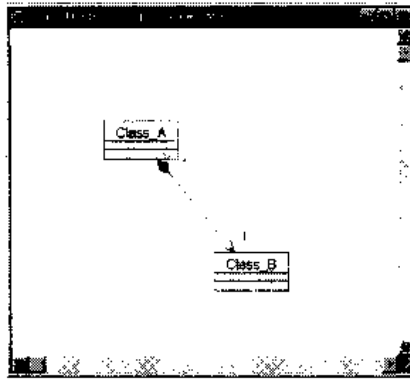
public class Class_A {
    →      public Class_B theClass_B;
    public Class_A() {}
}
  
```

Class B:

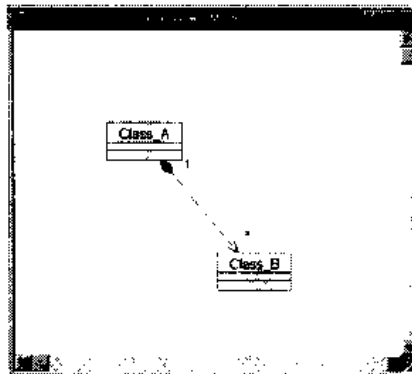
//Source file: Class_B.java

```

public class Class_B {
    public Class_B() {}
}
  
```



如果关系的倍数大于一，则Rose在生成关系时使用容器类。下面看这个关系：



Class A:

```
//Source file: Class_A.java
import java.awt.Container;

public class Class_A {
    public Container theClass_B;

    public Class_A() {}
}
```

Class B:

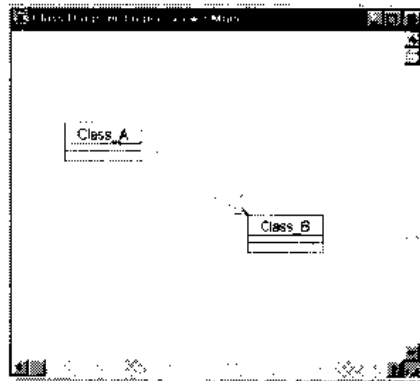
```
//Source file: Class_B.java

public class Class_B {
    public Class_B() {}
}
```

关系的倍数大于一时，Rose使用ContainerClass属性值。如果ContainerClass不含数值，则Rose使用包含多个对象的数组。

依赖性关系

对于依赖性关系，不生成属性。如果类A与类B有依赖性关系，则A和B中都不生成属性。



生成的代码样子如下：

```
//Source file: Class_A.java
```

```
public class Class_A {
    public Class_A() {}
}
```

和

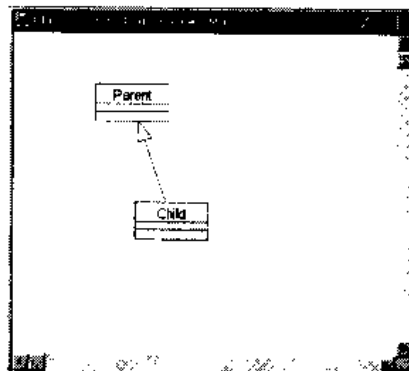
```
//Source file: Class_B.java
```

```
public class Class_B {
    public Class_B() {}
}
```

Rose不在A中放入B的引用。依赖性关系不生成关系的代码。

一般化关系

UML一般化关系在Java中变成继承关系。在Rose模型中，继承关系显示如下：



对于一般化关系，Rose产生如下代码：

```
//Source file: Parent.java
```

```
public class Parent {
    public Parent() {}
}
```

和

```
//Source file: C:/Program Files/Rational/Rose
98i/java/source/Child.java
```

```
public class Child extends Parent {
    public Child() {}
}
```

在父类代码中，没有子类的说明语句，使父类更一般化，可以派生许多子类而不影响父类代码中。在子类中，支持从父类继承的代码。类声明如下：

```
Public class Child extends Parent
```

生成Java举例

本书多次提到ATM例子应用程序。本节生成ATM例子的框架Java代码，其组件模型见图13.2和图13.3。

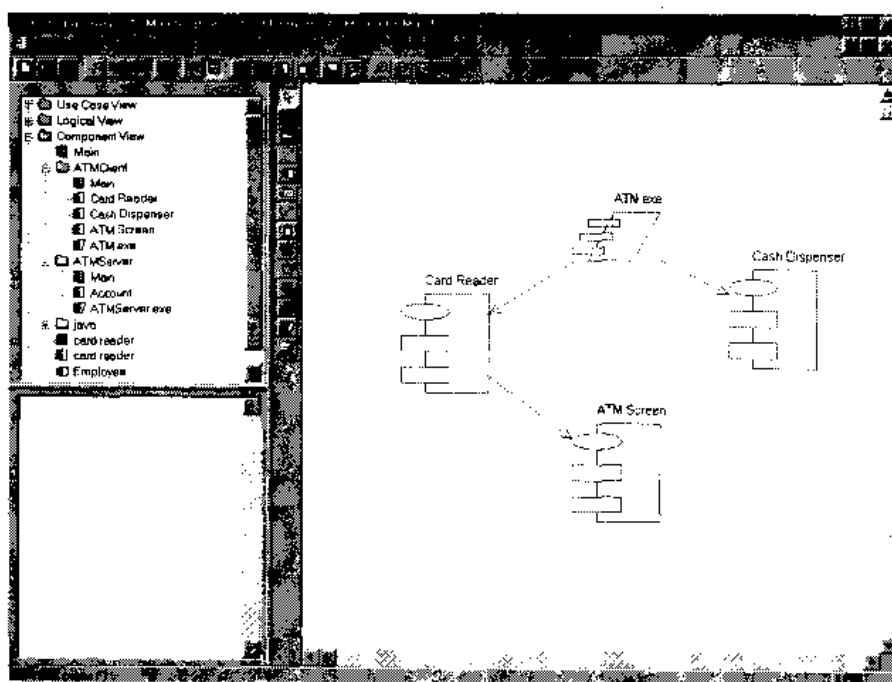


图13.2 ATM Client Component模型

对于ATM Client Component，将所有组件的语言设置为Java。然后选择Main Component框图中的ATMServer和ATMClient包，再选择Tools>Java>Generation Java。本书选配光盘中包括了每个组件生成的代码。

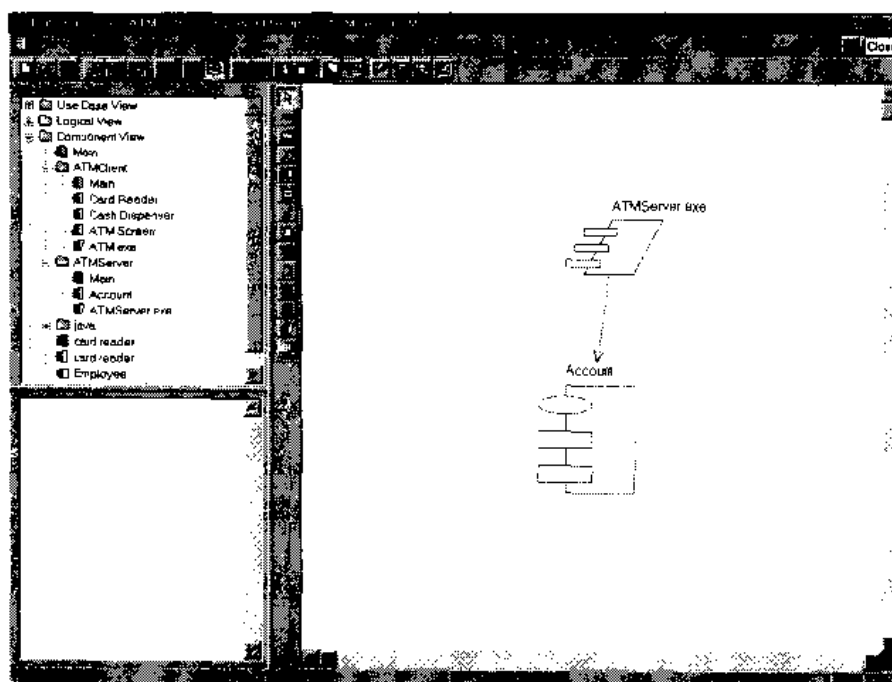


图13.3 ATM Server Component模型

练习

第3到第10章完成了订单输入系统的模型。现在要生成订单输入系统的代码。我们用图13.4所示的System Component框图。要生成代码，步骤如下，本书选配光盘中包括了本练习的代码。

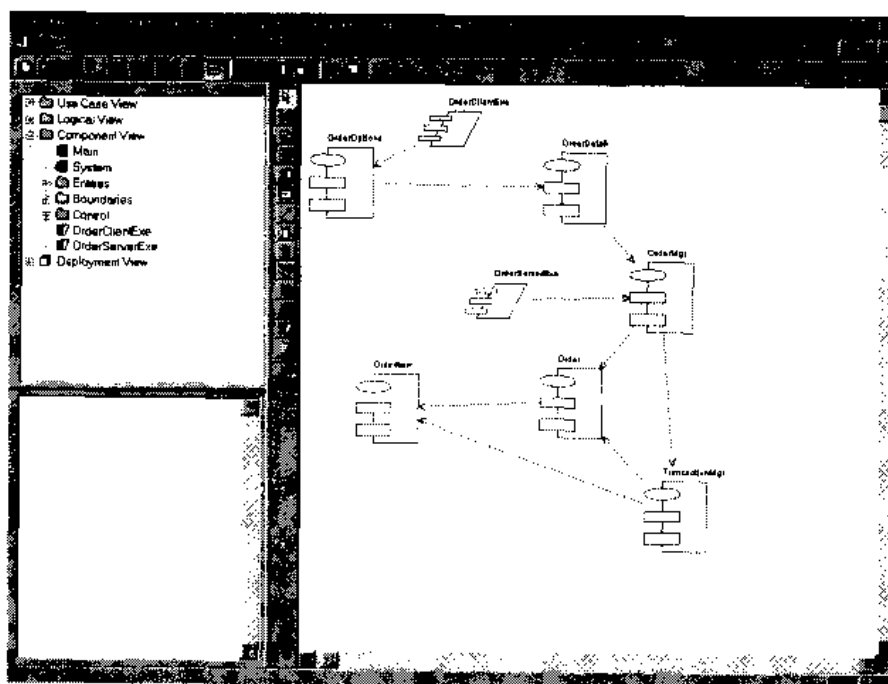


图13.4 订单输入系统的System Component框图

练习步骤

删除包体

1. 第9章练习中生成了每个类的包规范和包体。由于Java不用包体，因此首先要从模型中删除包体。
2. 打开System Component规范。
3. 选择OrderOptions包体。
4. 按Ctrl+D删除包体。
5. 对下列组件重复第3和第4步：
 - OrderDetail package body
 - OrderMgr package body
 - Order package body
 - OrderItem package body
 - TransactionMgr package body

将语言设置为Java

1. 打开实体组件包中Order组件的组件规范窗口（包规范）。
2. 将语言设置为Java。
3. 对下列组件重复第1和第2步：
 - Entities: OrderItem Package Specification
 - Boundaries: OrderOptions Package Specification
 - Boundaries: OrderDetail Package Specification
 - Control: TransactionMgr Package Specification
 - Control: OrderMgr Package Specification
 - OrderClientExe Task Specification
 - OrderServerExe Task Specification

输入Java数据类型

1. 我们使用的模型文件应包括Java数据类型，才能集成Java。这种类型已放在本书选配光盘的Data Types目录中。打开Main Class框图。
2. 从菜单中选择File►Import，从本书选配光盘中导入Data Types\Java Classes.ptl文件。
3. 打开Main Component框图。
4. 从菜单中选择File►Import，从本书选配光盘输入Data Types\Java Components.ptl文件。

设置Java数据类型

1. 由于Java数据类型的命名方法与C++数据类型稍有不同，因此要把所有属性和操作的数据类型设置为有效的Java数据类型。注意：如果完成了前面的练习，则只要将Date数据类型变为java.util.Date。

2. 打开Add Order Class框图。
3. 按图13.5设置属性、操作和变元的数据类型。

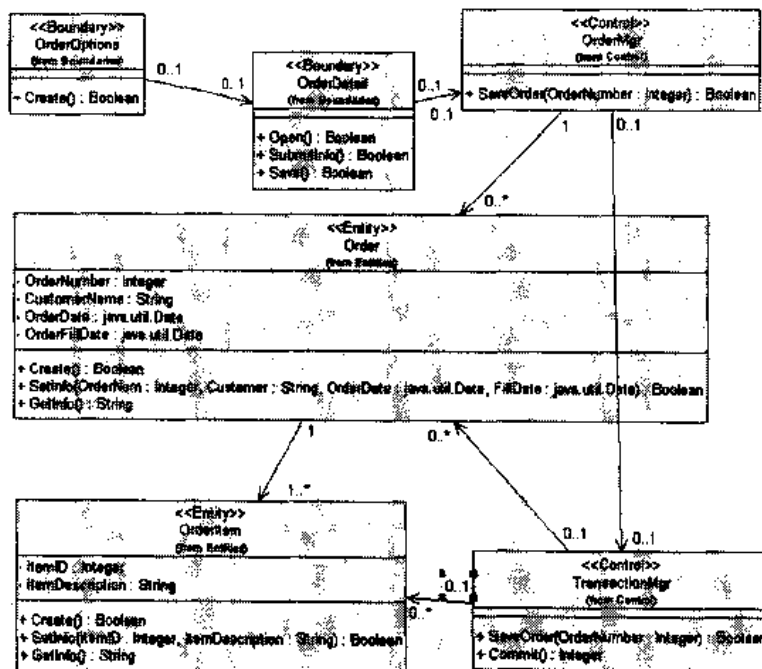


图13.5 订单输入类的Java数据类型

生成Java

1. 打开System Component框图。
2. 选择框图中的所有对象。
3. 选择菜单中的Tools>Java>Generate Java。本书选配光盘附有本章生成的Java代码。

小结

本章介绍了各个Rose模型元素如何用C++实现。利用类、包、属性、操作、关联、累加和其他Rose模型元素的代码生成属性，可以很好地控制生成的代码。

生成代码的步骤如下：

1. 生成组件。
2. 将类赋予组件。
3. 选择代码生成属性。
4. 选择Class或Component框图中要生成的类和组件。
5. 选择Tools>Java>Code Generation。
6. 选择Tools>Java>Java Source浏览生成的代码。

第14章 Visual Basic代码生成

- 设置Visual Basic代码生成属性
- 从Rose模型生成Visual Basic代码
- 将Rose模型映射Visual Basic结构

本章介绍如何从Rational Rose模型生成Visual Basic代码。

生成代码的步骤如下：

1. 生成组件（见第10章）。
2. 将类赋予组件（见第10章）。
3. 设置代码生成属性。
4. 选择Class或Component框图上的类或组件。
5. 选择Tools>Visual Basic>Update Code（或Code Generation）开始代码生成向导。
6. 选择Tools>Visual Basic>Browse Visual Basic Source浏览器生成的代码。

我们将介绍可以设置的代码生成属性和代码中如何实现各个Rose模型元素。

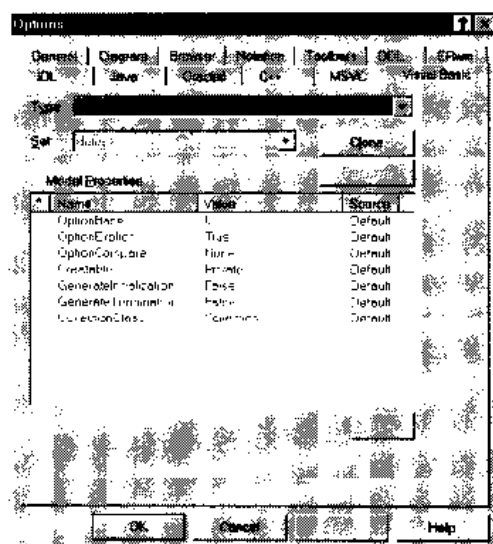
Rose用模型中的大量信息生成代码。例如，它要利用每个关系的倍增性、作用名、包和其他细节，利用每个类的属性、操作、可见性和其他细节。Rose从各个模型元素规范窗口中输入的信息收集生成代码所需的信息。

Visual Basic代码生成属性

用Rational Rose生成Visual Basic代码相当灵活。可以完全控制生成的内容和生成多少细节。例如，对于每个类，可以确定是否自动生成初始化和终止程序。对每个属性，可以控制可见性、名称和是否自动生成Get与Set操作。对每个模块，可以控制文件名。对每个一般化，可以控制是否采用实现委托。

这一切都是通过代码生成属性控制的。Rose提供涉及类、属性、操作、模块规范、关联和一般化的属性设置。

要显示所有这些属性，选择Tools>Options，然后选择Visual Basic标签。这个窗口中所作的任何改变都设置所有类、属性、操作等的缺省值。

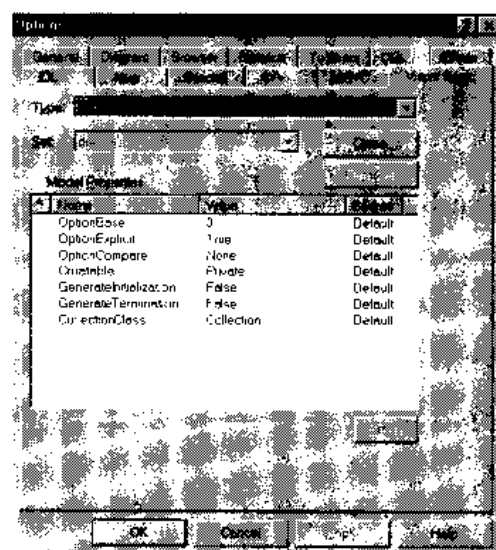


也可以对单个类、属性、操作和其他模型元素设置代码生成属性。为此，打开模型元素的规范窗口，并选择**Visual Basic**标签。在这个标签上，可以改变适用于各个模型元素的属性。

下面几节介绍类、属性、操作和模块的许多常用代码生成属性。本章稍后要介绍对各种关系和倍增性生成的代码，同时介绍其他一些代码生成属性。

类属性

类属性是适用于类的**Visual Basic**代码生成属性。这些属性可以改变类名，确定是否自动生成初始化和终止程序，以及设置其他类特定属性。



这种属性可以在两个地方设置。要设置所有类的这些属性，选择**Tools**►**Options**，然后选择**Visual Basic**标签，并从下拉列表框选择**Class**。要设置一个类的属性，在类规范窗口选择**Visual Basic**标签并编辑这些属性。

表14.1列出许多**Visual Basic**类属性及其作用和缺省值。

表14.1 类代码生成属性

属性	用途	缺省
Update Code (98i)	指定该类能否更新代码	True
Update Model (98i)	指定该类能否更新模型	True
OptionBase	设置数组基数标识符 (通常为0和1)	(none)
OptionExplicit	控制变量名是否需要显式声明	True
OptionCompare	控制比较字符串的方法	(none)
Instancing	确定类如何向其他应用程序提供	Private
Creatable (98)	控制其他模块对类的实例类	Private
GenerateInitialization(98)	控制类初始化程序的生成	False
GenerateTermination (98)	控制类终止程序的生成	False
CollectionClass(98)	设置类中集合的对象类型	Collectin

属性属性

属性属性是与属性相关的Visual Basic属性。利用这些属性可以确定代码中是否生成属性,生成代码中用什么属性名和属性是否生成Get、Set或Let操作。

这种属性可以在两个地方设置。要设置所有属性的这些属性,选择Tools►Options,然后选择Visual Basic标签,并从下拉列表框选择Attribute。要设置一个属性的属性,在属性规范窗口选择Visual Basic标签并编辑这些属性。

表14.2列出许多Visual Basic属性属性及其作用和缺省值。

表14.2 属性代码生成属性

属性	用途	缺省
IsConst (98)	控制属性是否常量	False
New	控制属性是否用new修饰符生成	False
WithEvents	控制属性是否用With Events修饰符生成	False
Subscript (98i)	指定属性的数组下标	Empty
NameIfUnlabeled (98)	设置属性名 (如未标出)	The\$supplier
GenerateDataMember (98)	控制属性是否生成	True
DataMemberName (98)	指定数据成员名	模型中的操作名
GenerateGetOperation (98)	控制属性是否生成Get操作	False
GenerateSetOperation (98)	控制属性是否生成Set操作	False
GenerateLetOperation (98)	控制属性是否生成Let操作	False

操作属性

操作属性是针对操作的Visual Basic代码生成属性。这些属性可以设置操作名,控制操作是否静态和设置操作的其他代码生成属性。

这种属性可以在两个地方设置。要设置所有操作的这些属性,选择Tools►Options,然后选择Visual Basic标签,并从下拉列表框选择Attribute。要设置一个操作的属性,在操作规范窗口选择Visual Basic标签并编辑这些属性。

表14.3列出许多Visual Basic操作属性及其作用和缺省值。

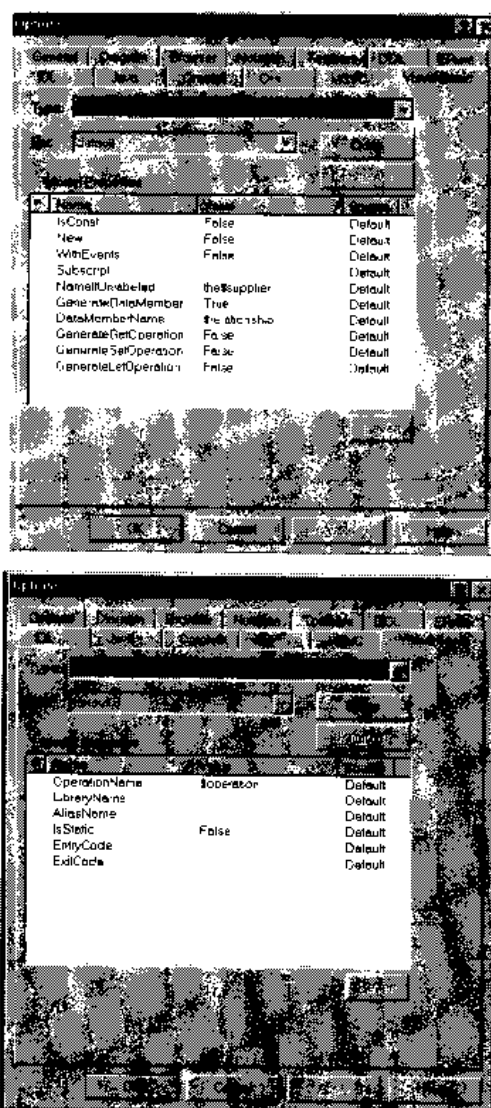


表14.3 操作代码生成属性

属性	用途	缺省值
OperationName (98)	设置生成的操作名	模型中的操作名
LibraryName	设置生成操作的库名	Empty
AliasName	设置操作别名	Empty
IsStatic	控制操作是否静态	False
ReplaceExistingBody (98i)	指定是否用缺省体代码覆盖现有体代码	False
EntryCode	用于输入和操作一起生成的代码或说明语句 (这个代码不放在保护区, 在逆向转出工程代码期间不受保护)	Empty
ExitCode	用于输入的操作一起生成的代码或说明语句 (这个代码不放在保护区, 在逆向转出工程代码期间不受保护)	Empty
DefaultBody	如果ReplaceExistingBody为真, 则用它指定体中包括的缺省文本(代码或说明语句)	Empty

模块规范属性

模块规范属性与Rose中生成的项目文件有关。只有一个模块规范属性，即项目文件名。

这种属性可以在两个地方设置。要设置所有头文件的这些属性，选择Tools►Options，然后选择Visual Basic标签，并从下拉列表框选择Module Specification。要设置一个头文件的属性，在头文件规范窗口选择Visual Basic标签并编辑这些属性。

表14.4列出许多Visual Basic头文件属性及其作用和缺省值。

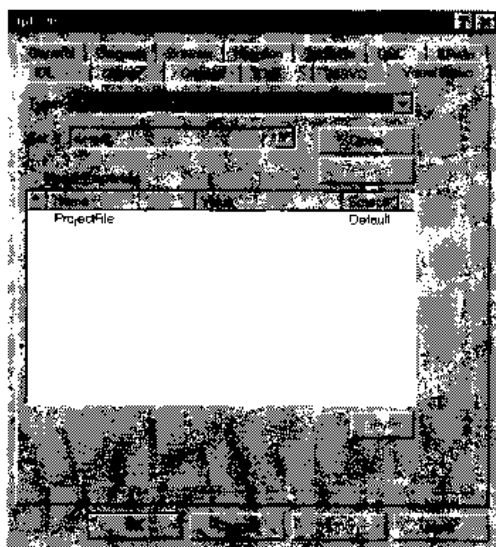
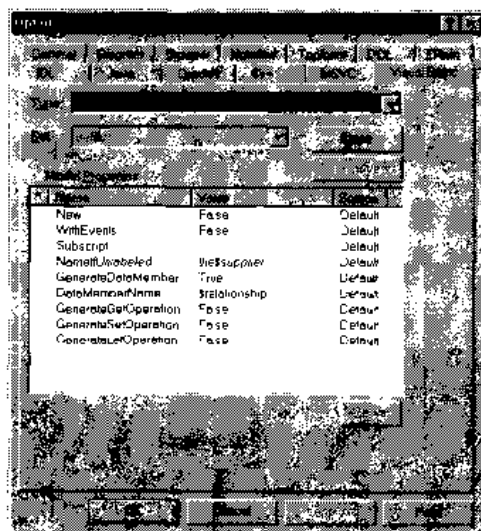


表14.4 头文件代码生成属性

属性	用途	缺省值
ProjectFile	设置项目文件名	<用代码生成向导设置>
UpdateCode (98i)	指定可否为这个组件生成代码	True
UpdateModel (98i)	指定可否为这个组件更新模型	True
ImportReferences (98i)	指定是否输入ActiveX组件	True
QuickImport (98i)	指定只输入ActiveX接口类或输入所有类，包括方法和操作	True

作用属性

作用属性是影响关系代码生成的Visual Basic代码生成属性。作用属性有好几个，可以设置生成的属性名、控制Get、Set和Let操作的生成和改变生成代码的其他部分。



这种属性可以在两个地方设置。要设置所有作用的这些属性，选择Tools►Options，然后选择Visual Basic标签，并从下拉列表框选择Role。要设置一个作用的属性，在作用规范窗口选择Visual Basic标签并编辑这些属性。

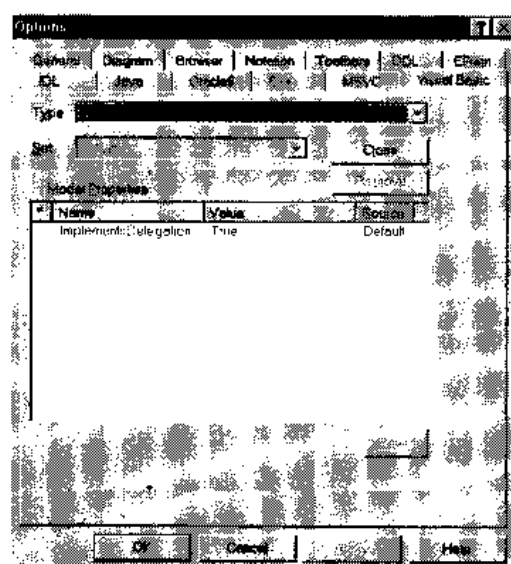
表14.5列出许多Visual Basic作用属性及其作用和缺省值。

表14.5 作用代码生成属性

属性	用途	缺省值
UpdateCode (98i)	指定可否为这个作用生成代码	True
New	控制关联是否用new修饰符生成	False
WithEvents	控制关联是否用With Events修饰符生成	False
Fullname (98i)	指定属性声明中是否用引用类的全名	False
Subscript (98i)	指定关联的数组下标	Empty
NameIfUnlabeled (98)	设置关联名（如未标出）	The\$supplier
GenerateDataMember (98)	控制关联是否生成	True
DataMemberName (98)	指定数据成员名	模型中的操作名
GenerateGetOperation (98)	控制关联是否生成Get操作	False
GenerateSetOperation (98)	控制关联是否生成Set操作	False
GenerateLetOperation (98)	控制关联是否生成Let操作	False

一般化属性

一般化属性是影响一般化关系代码生成的Visual Basic代码生成属性。由于Visual Basic不支持继承，因此只有一个一般化关系属性，使用实现委托。



这种属性可以在两个地方设置。要设置所有一般化的这些属性，选择Tools►Options，然后选择Visual Basic标签，并从下拉列表框选择Generalize。要设置一个一般化的属性，在一般化规范窗口选择Visual Basic标签并编辑这些属性。一般化的唯一代码生成属性是Implements-Delegation，控制一般化关系是否由实现委托功能实现。缺省情况下，这个属性为真。

在Rose 98中使用代码生成向导

在Rose模型中生成类和关联后，可以用代码生成向导（Code Generation Wizard）生成Visual Basic代码。要启动代码生成向导，选择要生成的对象，然后选择菜单中的Tools►Visual Basic►Generate Code。

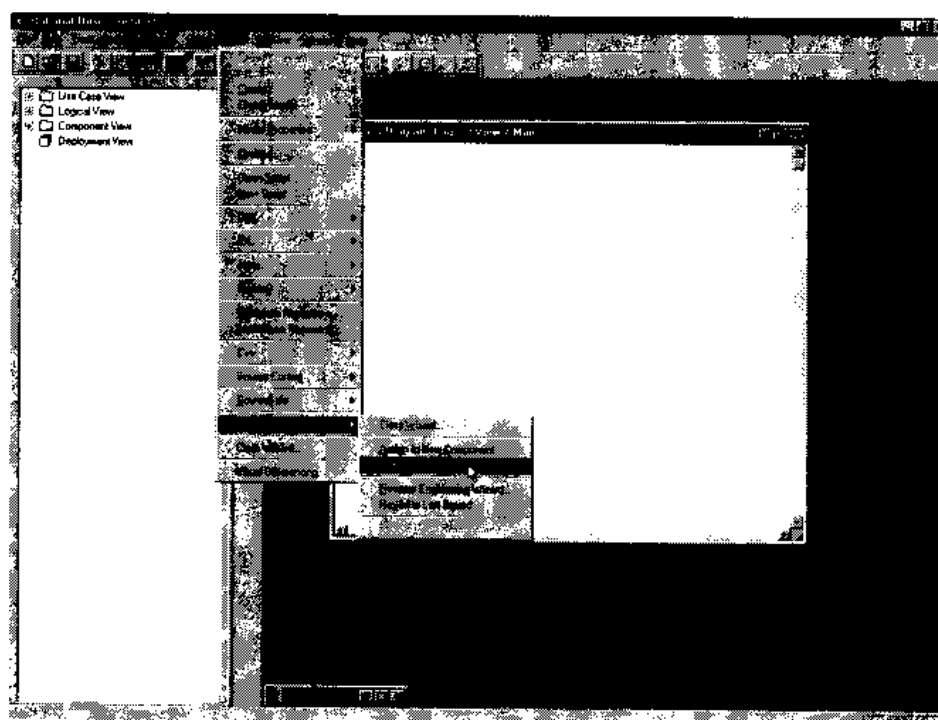


图14.1 代码生成向导

如果还没有将对象赋予组件，则会出现图14.2所示的窗口。选择要用的Visual Basic项目类型，然后单击OK。注意可以直接选择类并从菜单中选择Tools>Visual Basic>Assign to New Component将类赋予新组件。

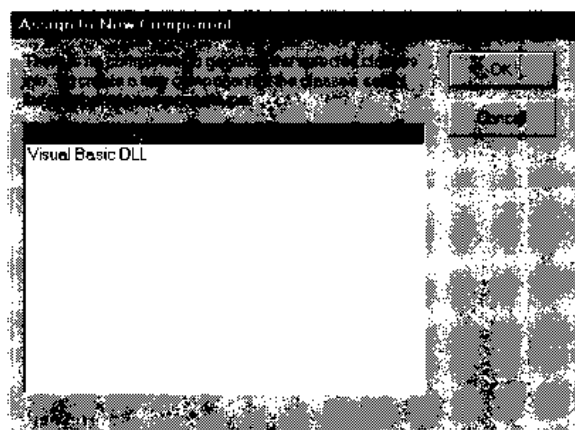


图14.2 将类赋予新组件

代码生成向导显示图14.3所示的欢迎窗口。如果不想再显示窗口，可以单击Don't show this page in the future复选框。然后单击Next按钮继续。

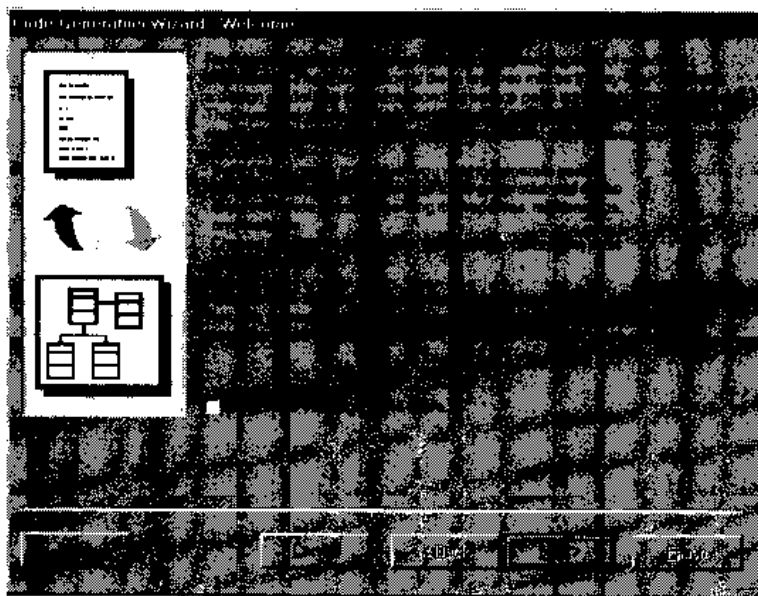


图14.3 代码生成向导欢迎窗口

接着出现图14.4所示窗口，提示选择类。可以自动或手工选择类。自动选择使模型中所有类与Visual Basic项目同步，手工选择则可以指定要生成或更新的类。单击Next按钮继续。

这时向导显示所有要生成或更新的类的预览，如图14.5。选择一个类并单击Preview按钮即可预览这个类的代码生成属性。如果不想预览任何类，则单击Next按钮继续。

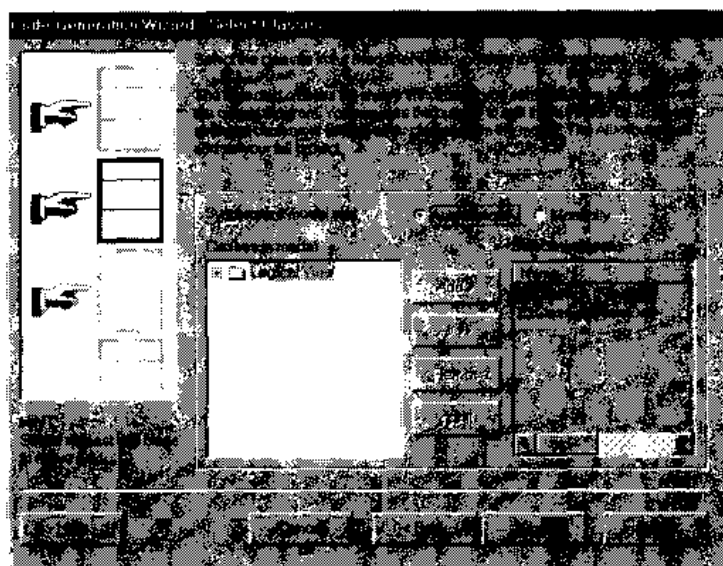


图14.4 选择要生成的类

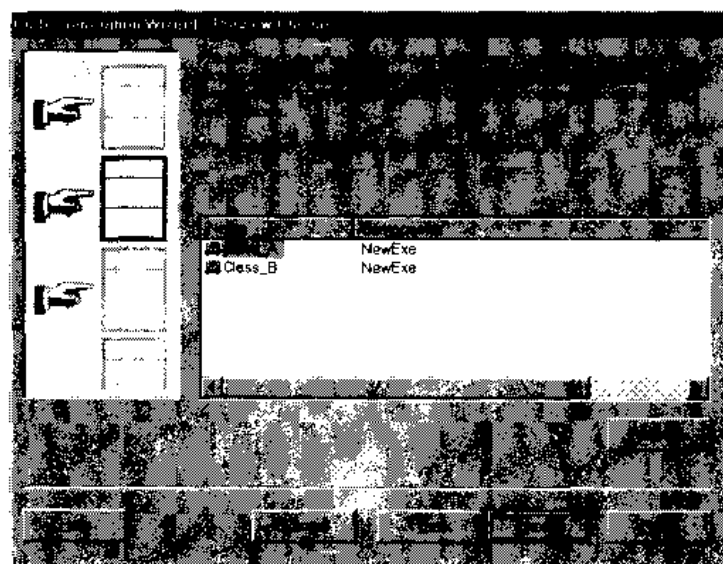


图14.5 预览要生成的类

如果预览类，则出现第一个预览窗口Class Options，如图14.6。从这个窗口可以设置实例和收集类属性。然后单击Next按钮继续。

这时出现Property Options窗口，如图14.7。选择属性并修改下列属性代码生成属性：

- Generate Variable
- Constant
- New
- WithEvents
- Array bounds
- Property Get
- Property Set
- Property Let

此外，可以显示声明语句的预览。对所有属性的属性感到满意后，单击Next按钮继续。

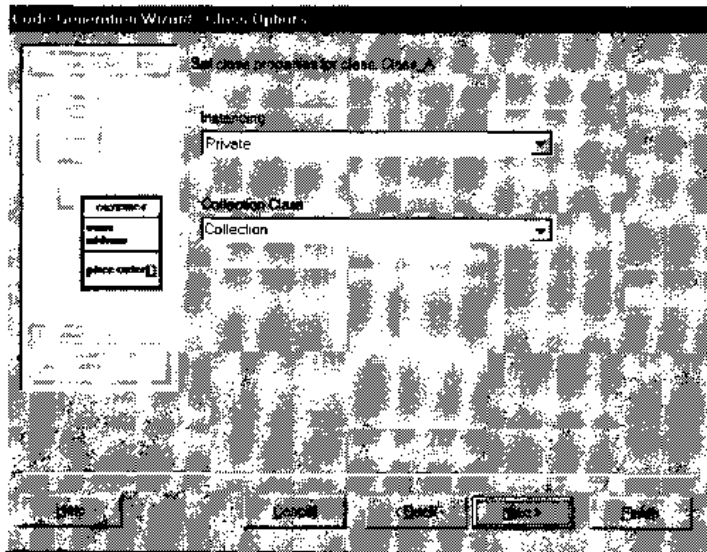


图14.6 设置类生成选项

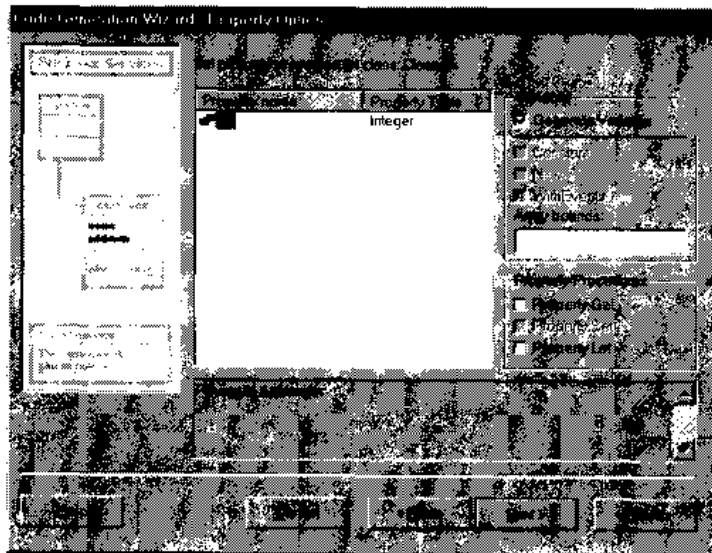


图14.7 设置属性选项

这时出现Role Options窗口，如图14.8。这个窗口设置关联代码生成属性，就像Role Options窗口设置属性代码生成属性一样。然后单击Next按钮继续。

这时出现Method Options窗口，如图14.9，显示类中的所有操作。可以对所选操作设置下列代码生成属性：

- Method Type
- Static
- DLL Library Name (仅用于DLL Component Type)
- DLL Alias Name (仅用于DLL Component Type)

此外，可以显示声明语句的预览。对所有属性的属性感到满意后，单击Finish按钮。回到预览窗口时，可以预览所要的其他类，然后单击Next按钮继续。这样就完成了类的预览。

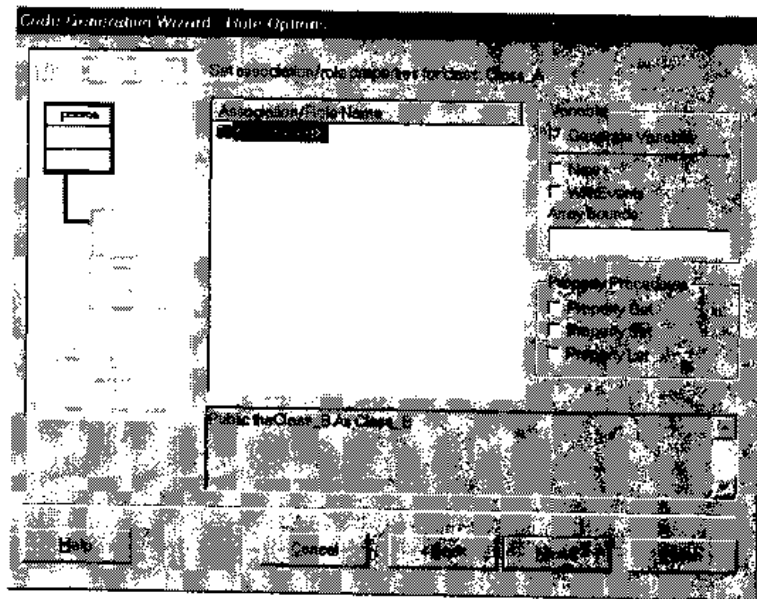


图14.8 设置作用选项

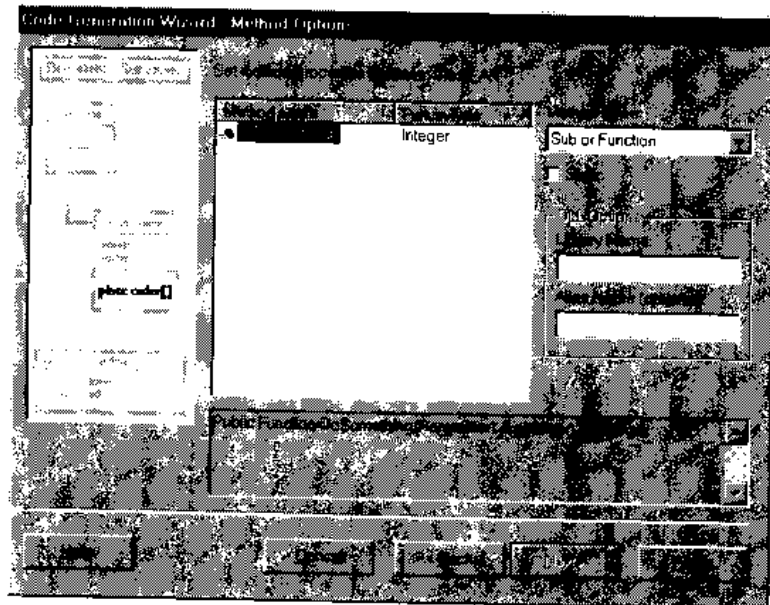


图14.9 设置方法选项

然后出现General Options窗口，如图14.20，从中可以设置整个项目的代码生成属性。可以设置的属性如下：

- 包括调试码
- 在所有生成方法中包括Err.Raise
- 包括说明语句
- 生成新的集合类

对所选的一般选项感到满意后，单击Next按钮继续。

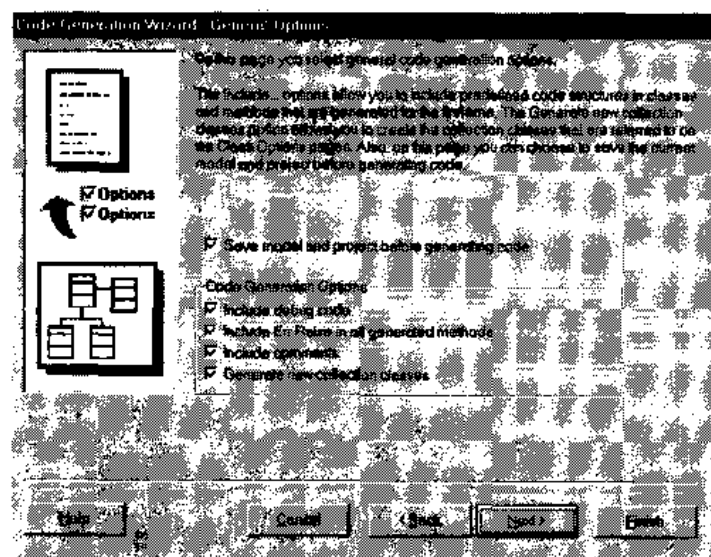


图14.10 设置一般选项

最后，出现Finish窗口，如图14.11。检查一般选项，然后单击Finish生成代码。

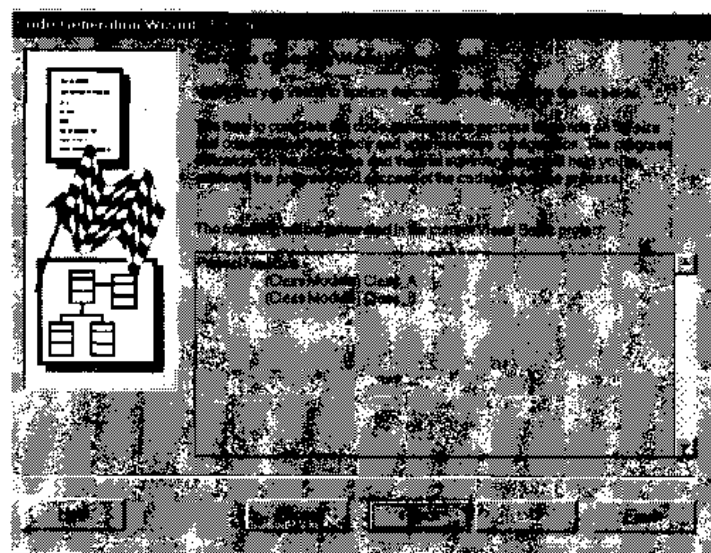


图14.11 代码生成向导Finish窗口

在Rose 98i中使用代码生成向导

在Rose模型中生成类和关联后，可以用代码生成向导（Code Generation Wizard）生成Visual Basic代码。要启动代码生成向导，选择要生成的对象，然后选择菜单中的Tools►Visual Basic►Update Code，出现图14.12所示屏幕。

图14.13中模型的Visual Basic组件显示为“Click next to continue”。如果没有Visual Basic组件，则要选择Visual Basic语言并单击Create a Visual Basic Component and Assign New Classes to It或按Ctrl+R生成新组件。

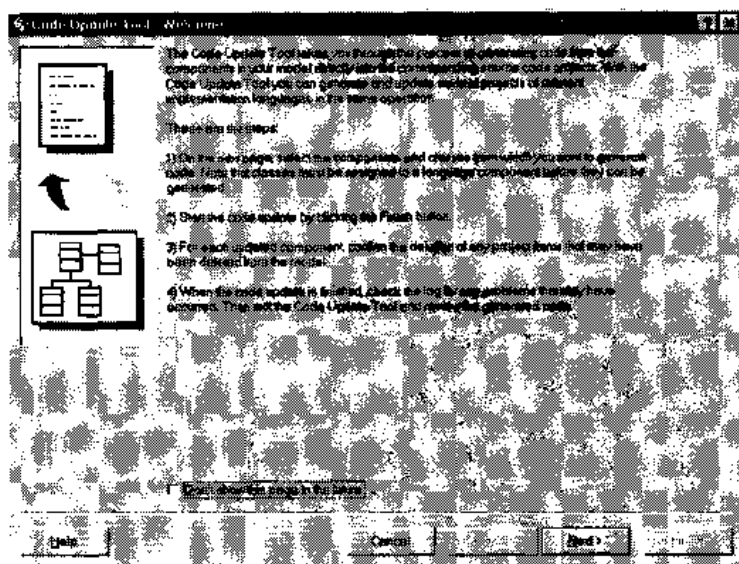


图14.12 代码更新工具欢迎屏幕

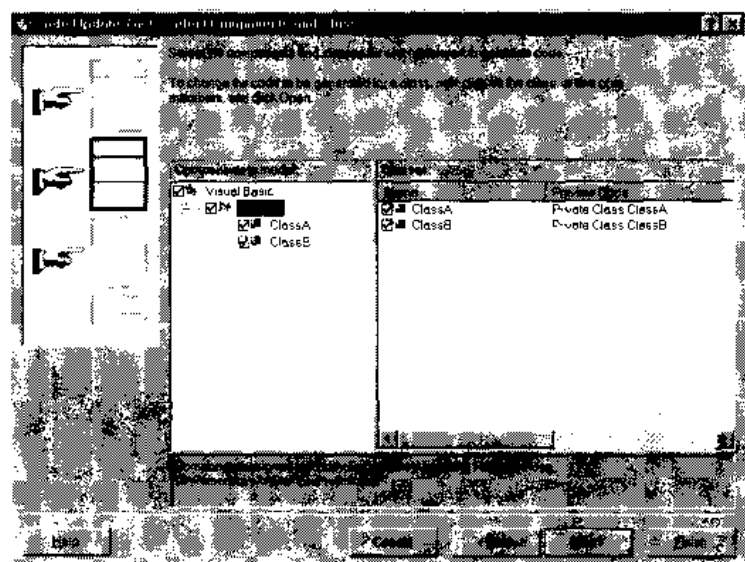


图14.13 选择组件和类

复选要更新的类或组件旁边的框。右键单击并选择Open浏览对象属性。右键单击Visual Basic项目浏览Visual Basic属性，如图14.14。

从这个窗口中，可以改变整个模型的许多代码生成属性。例如，可以设置数据成员使用的前缀。

右击组件将类赋予组件或浏览Visual Basic组件属性，如图14.15。例如，可以从这个窗口设置项目文件和组件版型。组件必须有项目文件和组件版型才能顺利进行代码生成。

要赋予类，右击组件并从弹出菜单选择Assign Classes，出现图14.16所示的组件指定工具。单击一个未指定类并将其拖动到组件上。

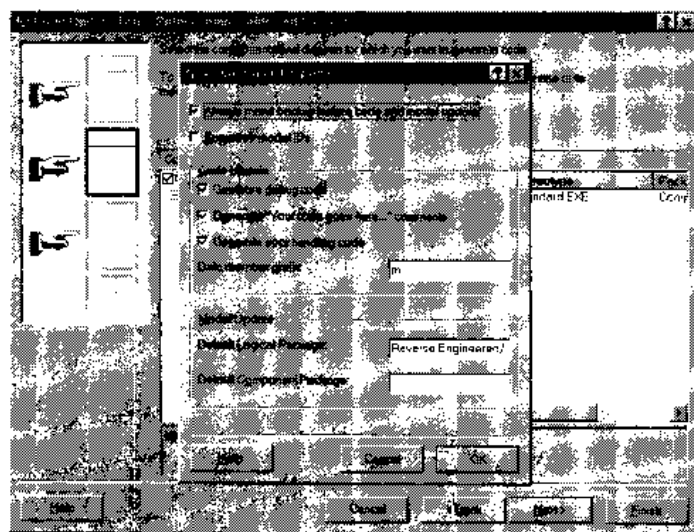


图14.14 浏览Visual Basic属性

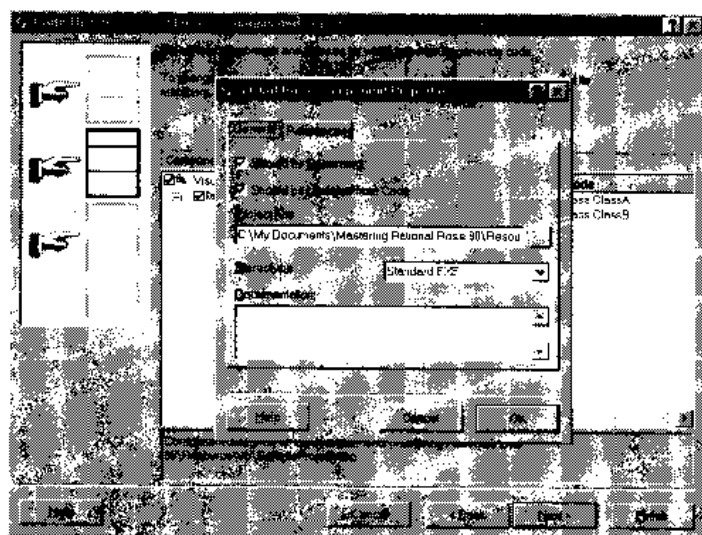


图14.15 Visual Basic组件属性

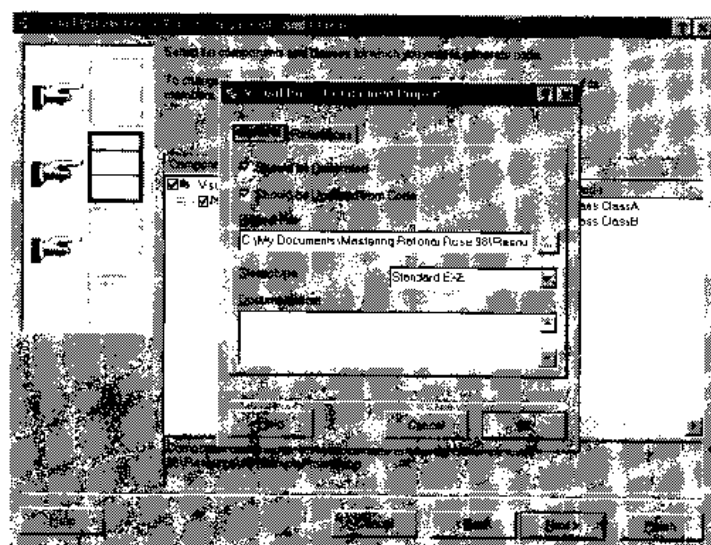


图14.16 组件指定工具

从Select Components and Classes窗口中，右击类或类成员（属性和操作）打开Model Assistant窗口，如图14.17。Model Assistant可以浏览和改变每个类、属性与方法的代码生成属性。

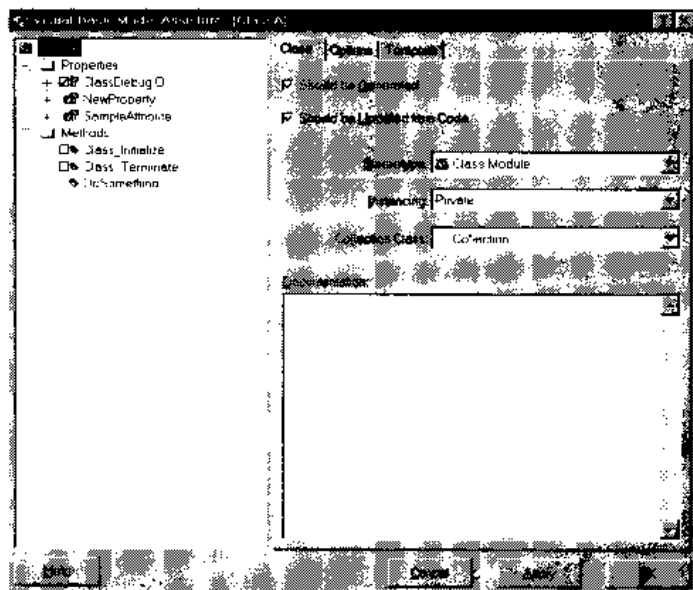


图14.17 打开Model Assistant窗口

展开属性可以显示其Get、Set和Let属性。复选生成Get、Set和Let属性的框。单击Get、Set和Let属性改变该属性的选项，如图14.18。

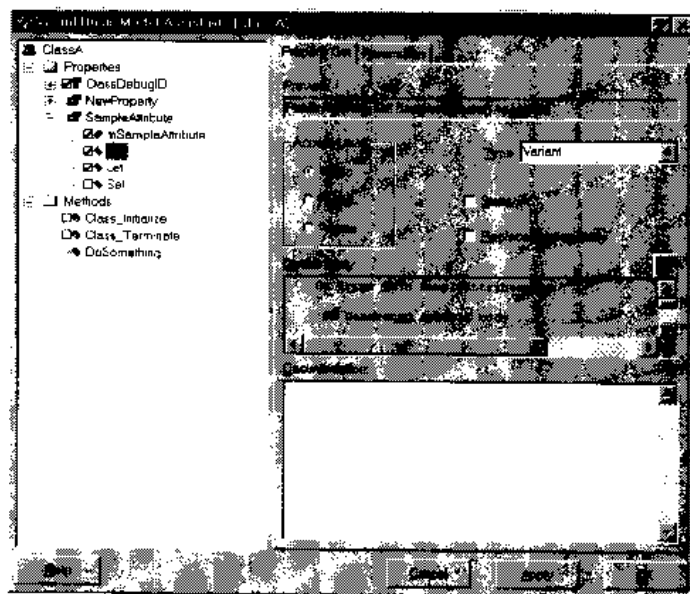


图14.18 SampleAttribute的GetProperty选项

单击属性名本身可以显示该属性的数据成员属性，如图14.19。利用这些属性可以设置属性的访问级、数据类型和其他代码生成属性。

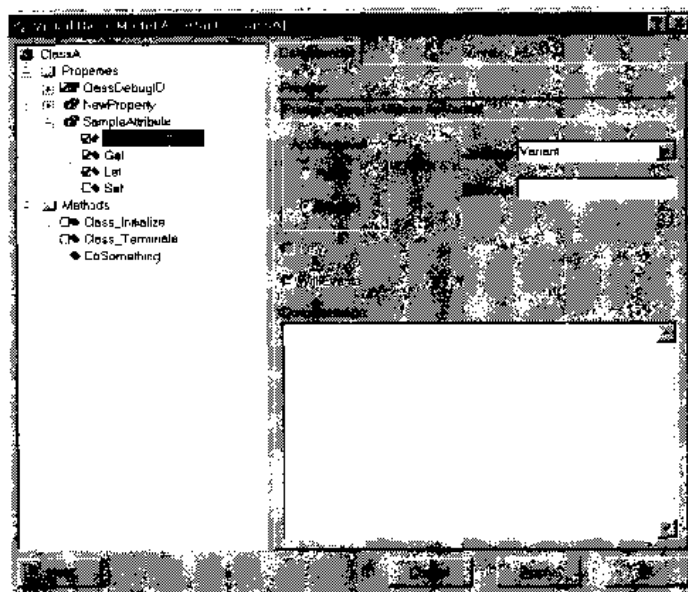


图14.19 m_SampleAttribute的数据成员属性

检查和修改完Model Assistant窗口中的代码生成属性后，单击OK关闭Model Assistant窗口。然后单击Next按钮完成代码生成并打开图14.20所示窗口，其中介绍了代码更新过程。单击Finish按钮开始更新代码。

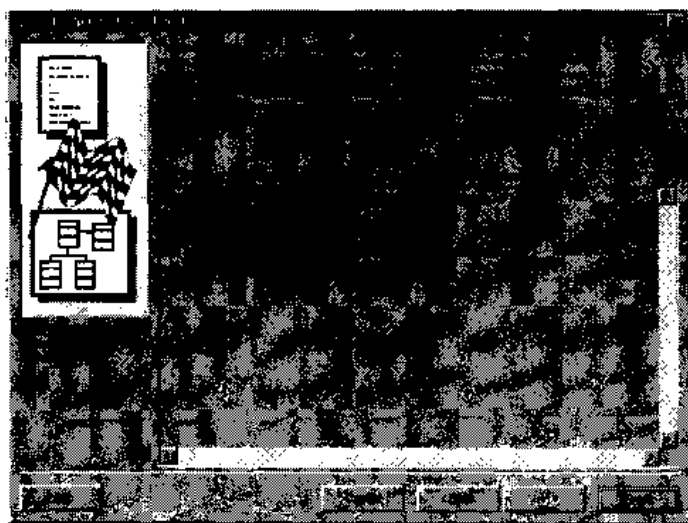


图14.20 代码更新工具的结果窗口

代码更新完毕后，出现代码更新小结窗口，如图14.21。单击Class按钮关闭代码更新小结窗口，返回模型中。

生成的代码

下面几节介绍对类、属性与操作和类间各种关系生成的Visual Basic代码。每一节都通过一些样本代码显示从Rose模型生成的代码样子。

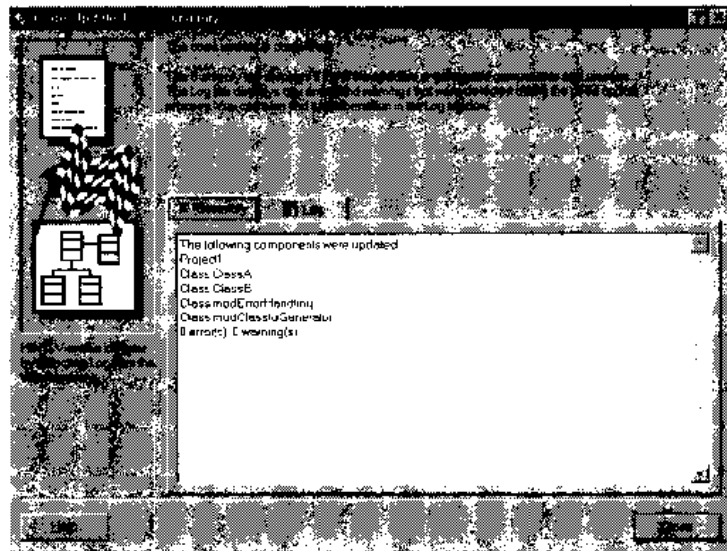


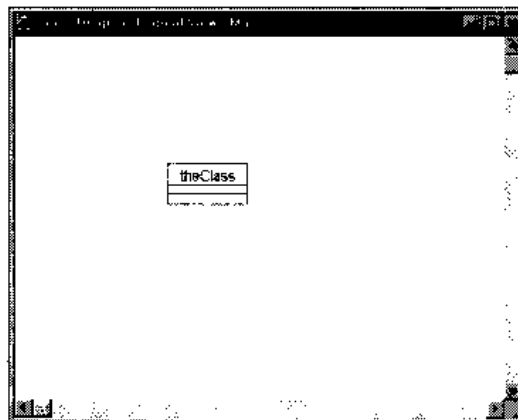
图14.21 代码更新小结窗口

Rose生成代码时，使用模型元素规范中的信息。例如，它在生成类代码时要利用类的不同规范（可见性、属性、操作等）。

下面先介绍典型类的代码生成。

Classes

类生成代码时，对象模型中的类成为**Visual Basic**类。每个类产生如下代码：



Class Module: theClass

```
Option Explicit

'##ModelId=372E96CF00AA
Public mlClassDebugID As Long

'##ModelId=372E96CF0079
Public Property Get ClassDebugID() As Variant
    On Error GoTo ClassDebugIDErr

    Call RaiseError(MyUnhandledError, "ClassDebugID Property")
End Property
```

模块theClass包含该类的生成代码。如果这是个新项目，则还生成下列两个文件。modClassIDGenerator模块包含一个选择下一个类调试ID的函数，modErrorHandler类包含一个简单错误处理程序。这两个文件在每个项目中只生成一次，而每个类都生成类似于theClass的不同模块文件。

Module: modClassIDGenerator

```
Option Explicit

'Class ID generator
'##ModelId=372E96F60226
Public Function GetNextClassDebugID() As Long
    Static lClassDebugID As Long
    lClassDebugID = lClassDebugID + 1
    GetNextClassDebugID = lClassDebugID
End Function
```

Module: modErrorHandler

```
Option Explicit

'##ModelId=372E96F603B0
Public Const MyObjectError1 = 1000

'##ModelId=372E96F603B1
Public Const MyObjectError2 = 1010

'##ModelId=372E96F603B2
Public Const MyObjectErrorN = 1234

'##ModelId=372E96F603B3
Public Const MyUnhandledError = 9999

'There are a number of methods for retrieving the error
'message. The following method uses a resource file to
'retrieve strings indexed by the error number you are
'raising.
'##ModelId=372E96F603AD
Public Sub RaiseError(ErrorNumber As Long, Source As String)
    Dim strErrorText As String

    strErrorText = GetErrorTextFromResource(ErrorNumber)

    'raise an error back to the client
    Err.Raise vbObjectError + ErrorNumber, Source, strErrorText
End Sub

' This function will retrieve an error description from a resource
' file (.RES). The ErrorNum is the index of the string
' in the resource file. Called by RaiseError
'##ModelId=372E96F60370
Private Function GetErrorTextFromResource(ErrorNum As Long) As String
```

```

On Error GoTo GetErrorTextFromResourceError
Dim strMsg As String

' get the string from a resource file
GetErrorTextFromResource = LoadResString(ErrorNum)

Exit Function
GetErrorTextFromResourceError:
If Err.Number <> 0 Then
    GetErrorTextFromResource = "An unknown error has occurred!"
End If
End Function

```

本例显示了一个类缺省生成的项目。但代码中还可以生成大量细节信息。稍后将介绍更完整的例子。一个类的所有属性、操作和关系都会在生成代码中体现。每个类生成的主要元素包括：

- 类名
- 类的初始化程序
- 类的终止程序
- 每个属性的Get、Set和Let操作
- 类文档
- 属性
- 操作
- 关系
- 文档

模型中的每个类生成一个Visual Basic类模块文件。每个文件用类名命名。例如，Employee类生成Employee类模块文件。

生成代码时，Rose用模型Component视图中建立的包结构生成相应目录。模型中的每个包生成一个目录。在目录中，Rose生成该包中类的模块文件。如果Component视图中没有生成组件和包，则Rose用logical视图中的包结构生成目录结构。

Rose模型中的大量信息都直接用于生成代码。例如，每个类的属性、操作和关系、类名直接影响生成的代码。其他模型属性（如输入类文档）不直接影响代码，而是在生成代码中成为说明语句。

表14.6列出了类规范窗口中的属性，并注明哪些类直接影响生成的代码。

表14.6 类规范对生成代码的影响

属性	对代码的影响
Name	模块中的名称成为类名
Type	直接影响生成类类型
Stereotype	直接影响生成的模块文件类型（Class Module, MDI Form等）
Export Control	不影响生成代码
Documentation	说明语句

(续表)

属性	对代码的影响
Cardinality	不影响代码生成
Space	不影响代码生成
Persistence	不影响Visual Basic代码生成, 但影响类能否生成DDL
Concurrency	不影响代码生成
Abstract	不影响代码生成
Formal Arguments	不影响代码生成
Operations	参数化类的代码中包括正式变元
Attributes	代码中生成
Relationships	代码中生成

属性

除了类本身外, Rose还生成类的属性。对于每个属性, Rose包括:

- 可见性 (只能为public或private, 不允许protected)
- 数据类型
- 缺省值
- Get操作
- Set操作
- Let操作

对属性, Rose生成如下代码:

```

'##ModelId=3734B0F6006E
Private mPrivateAttribute As integer

'##ModelId=3734B10D0118
Public Property Get PrivateAttribute() As integer
    On Error GoTo PrivateAttributeErr

    '## Generated default body ...
    Let PrivateAttribute = mPrivateAttribute

    Exit Property
PrivateAttributeErr:
    Call RaiseError(MyUnhandledError, "PrivateAttribute Property Get")
End Property

'##ModelId=3734B10D011B
Public Property Let PrivateAttribute(ByVal vNewValue As integer)
    On Error GoTo PrivateAttributeErr

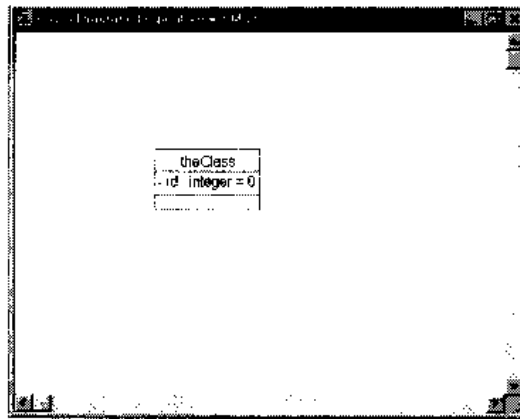
    '## Generated default body ...
    Set mPrivateAttribute = vNewValue

    Exit Property
PrivateAttributeErr:
    Call RaiseError(MyUnhandledError, "PrivateAttribute Property Set")
End Property

```

这个代码中，属性带有Get和Let语句。

下面看看下图所示类中生成的代码：



属性生成的第一个代码是声明，如下所示：

```
'##ModelId=3734B22F023A
Private mid As Integer
```

缺省情况下，只生成这些内容。但如果将Visual Basic选项配置成生成Get、Set和Let操作，则生成如下代码：

```
'##ModelId=3734B27F0174
Public Property Let id(ByVal vNewValue As Integer)
    On Error GoTo idErr

    '## Generated default body ...
    mid = vNewValue

    Exit Property
idErr:
    Call RaiseError(MyUnhandledError, "id Property Let")
End Property

'##ModelId=3734B27F0173
Public Property Get id() As Integer
    On Error GoTo idErr

    '## Generated default body ...
    id = mid

    Exit Property
idErr:
    Call RaiseError(MyUnhandledError, "id Property Get")
End Property
```

此外，Rose模型更新后包括新操作，如图14.22。

可以将代码生成属性设置成生成Get、Set和Let操作，并用表14.2所示属性改变代码的其他方面。

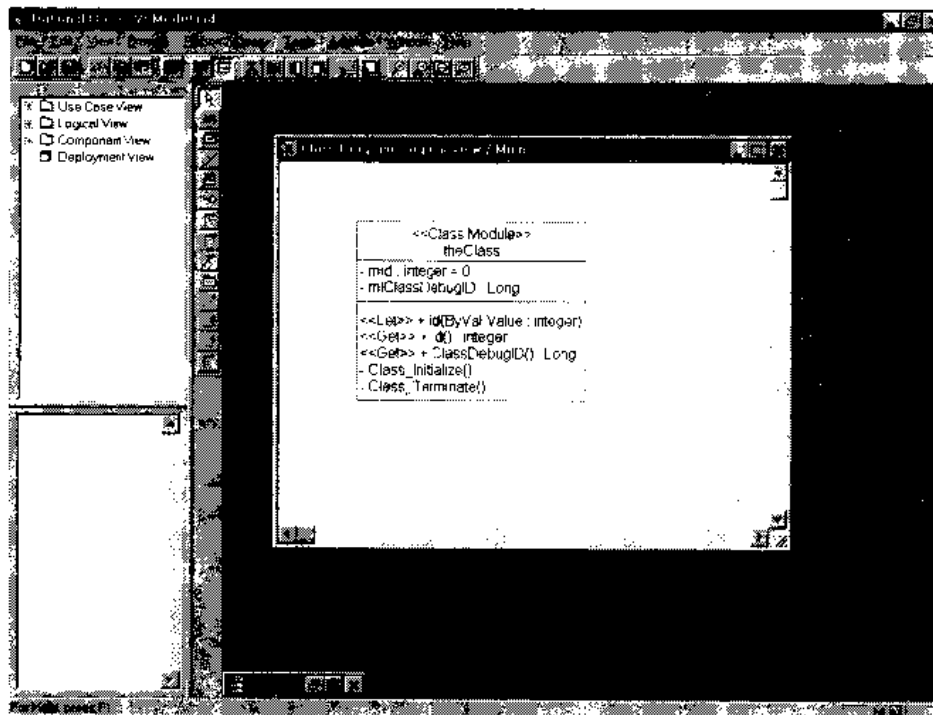


图14.22 VB代码生成之后的类

操作

Rose对类中每个操作生成代码。对每个操作，生成代码包括操作名、参数、参数数据类型和返回值。每个操作生成如下代码：

```
'##ModelId=3734B41B005A
Private Sub PrivateOperation()
    On Error GoTo PrivateOperationErr

    '## Your code goes here ...

    Exit Sub
PrivateOperationErr:
    Call RaiseError(MyUnhandledError, "PrivateOperation Sub")
End Sub

'##ModelId=3734B3FA001E
Friend Sub ProtectedOperation()
    On Error GoTo ProtectedOperationErr

    '## Your code goes here ...

    Exit Sub
ProtectedOperationErr:
    Call RaiseError(MyUnhandledError, "ProtectedOperation Sub")
End Sub

'##ModelId=3734B3DE0190
Public Sub PublicOperation()
```

```

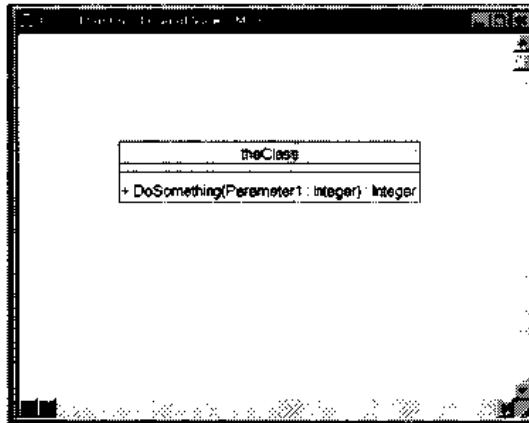
On Error GoTo PublicOperationErr

'## Your code goes here ...

Exit Sub
PublicOperationErr:
    Call RaiseError(MyUnhandledError, "PublicOperation Sub")
End Sub

```

本例中，操作没有变元和返回值。下例中的操作则有：变元和返回值。



```

'Operation documentation
'##ModelId=3734B4850366
Public Function DoSomething(Parameter1 As Integer) As Integer
    On Error GoTo DoSomethingErr

    '## Your code goes here ...

Exit Function
DoSomethingErr:
    Call RaiseError(MyUnhandledError, "DoSomething Function")
End Function

```

可以看出，代码中生成完整的操作签名。输入的操作文档也成为代码中的说明语句。如果输入操作协议、限定、异常、时间、空间、前提条件、词法和事后条件的信息，则这些信息不放进生成代码中。

生成代码后，在**Code Goes Here...**行中插入每个操作的实现代码。通过在这个保护代码区放上代码，可以在逆向转出工程代码期间得到保护。

和其他模块元素一样，可以通过修改代码生成属性控制生成的操作代码。例如，修改**IS**属性可以生成静态函数。表14.3列出了操作的代码生成属性。

双向关联

要支持双向关联，**Rose**在代码中生成属性。关系中的每个类包含一个支持关联的属性。本节介绍下列代码生成属性：

- **DataMemberName** role property
- **GenerateGetOperation** role property

- GenerateSetOperation role property .
- GenerateLetOperation role property

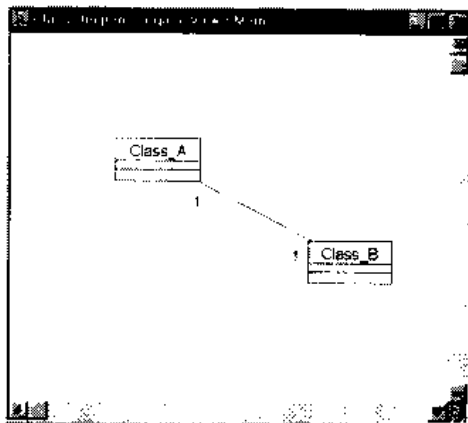
两个类相关联时生成的代码如下:

Class_A:

```
##ModelId=3734B5B70244
Public NewProperty As Class_B
```

Class_B:

```
##ModelId=3734B5B70280
Public NewProperty As Class_A
```



可以看出, Rose在双向关联关系两端自动生成属性。利用NewProperty属性, A类可以方便地访问B类。利用NewProperty属性, B类可以方便地访问A类。如果不提供作用名, 则用缺省的(用处不大的)NewProperty。如果提供作用名, 则用作用名作为属性名。

下面看看上述两个类生成的代码。关联反映在Class_A和Class_B的类模块文件中。A类的类模块文件如下。这里我们在Model Assistant窗口选择Get与Set属性选项。在Rose 98中, 使用GenerateGetOperation和GenerateSetOperation代码生成属性。

Class_A:

```
##ModelId=3734B5B70244
Private mNewProperty As Class_B

##ModelId=3734B6F10099
Public Property Set NewProperty(ByVal vNewValue As Class_B)
    On Error GoTo NewPropertyErr
    ## Generated default body ...
    Set mNewProperty = vNewValue
    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property
```



```

'##ModelId=3734B6F10096
Public Property Get NewProperty() As Class_B
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set NewProperty = mNewProperty

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

```

Class_B:

```

'##ModelId=3734B5B70280
Private mNewProperty As Class_A

'##ModelId=3734B6FD00F5
Public Property Set NewProperty(ByVal vNewValue As Class_A)
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

'##ModelId=3734B6FD00F2
Public Property Get NewProperty() As Class_A
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set NewProperty = mNewProperty

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

```

生成代码时，Rose在A类中放上属性NewProperty。由于每个属性生成Get与Set操作，而关联生成属性，因此可以对关联属性生成Get与Set操作。缺省情况下，属性名为NewProperty。要改变属性名，在关联中设置作用名。Rose用作用名生成属性名。每个生成属性缺省可见性为Private。要改变可见性，打开关联规范窗口，选择Role A General或Role B General标签，并改变Export Control。


```

'##ModelId=3734385400FA
Private mNewProperty As Class_B

'##ModelId=373438B40284
Public Property SetNewProperty(ByVal vNewValue As Class_B)
    On Error GoToNewPropError

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropError:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

'##ModelId=373438B40281
Public Property GetNewProperty() As Class_B
    On Error GoToNewPropError

    '## Generated default body ...
    SetNewProperty = mNewProperty

    Exit Property
NewPropError:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

```

B类不增加关联代码，因为关联是单向的。

可以看出，**Rose**只在关系一端生成专用属性。具体地说，是在**Client**类中生成属性，而不在**Supplier**类中生成。

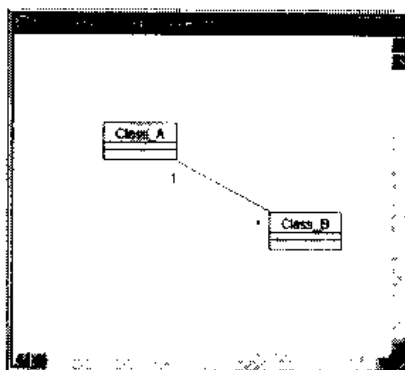
在**Supplier**类中生成的代码包括上面双向关联中介绍的所有头和实现文件行。对于双向关联，每个类提供一个新属性，两个类中都包括上节介绍的代码。而对于单向关联，则只在**Supplier**类中生成代码。

注意，这个关联加进一对一的倍增性。下面将介绍倍增性设置对代码生成影响。

倍增性为一对多的关联

在一对一关系中，**Rose**只是生成支持关联的相应属性。而对于一对多关系，一个类要包含其他类的设置。本节介绍的代码生成属性是**Subscript**作用属性。

首先举一个例子。



这里具有一对多关系。如上所示，B类可以生成指向A类的属性，但A类中仅有简单指针属性是不够的，A类中的属性要用某种容器类或数组作为数据类型。Rose生成的代码如下：

Class_A:

```

'##ModelId=3734BA5C037A
Private mNewProp As Collection

'##ModelId=3734BA87033A
Public Property SetNewProp(ByVal vNewValue As Collection)
    On Error GoToNewPropErr
    '## Generated default body ...
    Set mNewProp = vNewValue

    Exit Property
NewPropErr:
    Call RaiseError(MyUnhandledError, "NewProp Property Set")
End Property

'##ModelId=3734BA870337
Public Property GetNewProp() As Collection
    On Error GoToNewPropErr
    '## Generated default body ...
    SetNewProp = mNewProp

    Exit Property
NewPropErr:
    Call RaiseError(MyUnhandledError, "NewProp Property Get")
End Property

```

Class_B:

```

'##ModelId=36C6433603B6
Private mNewProp As Class_A

```

可以看出，B类只是引用A类，而A类中生成B类属性时用了容器类。

Rose提供Collection类型的容器类。如果愿意，也可以指定Subscript作用属性，使用数组。对B类的类模块文件，A类的引用以属性形式包括，并对这个属性生成Get与Set操作。由于B类的属性不需要容器类，B类生成的代码与上节“双向关联”和“单向关联”中相同。但A类的代码包括Get与Set操作，需要在代码生成后完成。由于使用容器类，Rose不能生成Get与Set操作的有用缺省代码，要由用户自己填写。

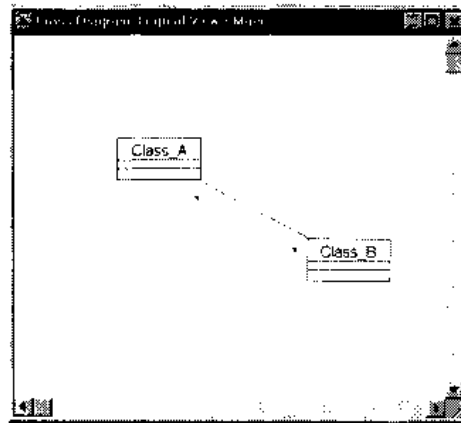
倍增性为多对多的关联

这里生成的代码类似于一对多关系，但Rose在关系两端生成容器类。

本节介绍的代码生成属性包括：

- ContainerClass作用属性
- Subscript作用属性

下面生成下图所示关系的代码:



这里Rose在关系两端生成容器类。生成的代码如下:

Class_A:

```

'##ModelId=3734BA5C037A
Private mNewProperty As Collection

'##ModelId=3734BA8703BA
Public Property Set NewProperty(ByVal vNewValue As Collection)
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

'##ModelId=3734BA8703B7
Public Property Get NewProperty() As Collection
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set NewProperty = mNewProperty

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property
  
```

Class_B:

```

'##ModelId=3734BA5C037B
Private mNewProperty As Collection

'##ModelId=3734BBF303B1
Public Property Set NewProperty(ByVal vNewValue As Collection)
    On Error GoTo NewPropertyErr
  
```

```

'## Generated default body ...
Set mNewProperty = vNewValue

Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

'##ModelId=3734BBF303AE
Public Property Get NewProperty() As Collection
    On Error GoTo NewPropertyErr

'## Generated default body ...
Set NewProperty = mNewProperty

Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

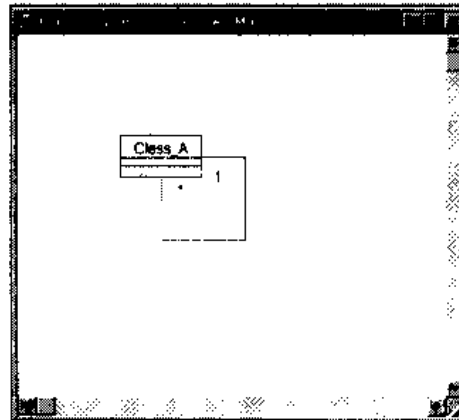
```

A类的代码与“倍增性为 一对多的关联”中一样，不同之处在于B类也用容器类生成属性。

和一对多关联一样，可以指定Subscript使用属性以使用数组，也需要改变Get和Set操作的代码。

反身关联

反身关联与两个类之间的关联用相似方法处理。下面是下图情形的代码：



```

'##ModelId=3734BA5C037A
Private mNewProperty As Class_A

'##ModelId=3734BCA9001E
Public NewProperty2 As Collection

'##ModelId=3734BA8703BA
Public Property Set NewProperty(ByVal vNewValue As Class_A)
    On Error GoTo NewPropertyErr

```

```

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

'##ModelId=3734BA8703B7
Public Property Get NewProperty() As Class_A
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set NewProperty = mNewProperty

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

```

和普通关联一样，类中生成支持关系的属性。如果倍数为1，则生成简单属性。如果倍数大于1，则用容器类。

可以看出，生成的代码与典型一对多关系中相似。这里A类包含A类类型的属性。

按值累积（复合关系）

累积关系有两种：按值和按引用。在按值关系中，一个类包含另一个类。在按引用关系中，一个类包含另一个类的引用。

这里要介绍的代码生成属性为ContainerClass作用属性。

下面先介绍按值累积（复合关系）。在UML中，复合用下列符号表示：

和关联关系一样，Rose在生成累积代码时生成属性。本例中，Class_B是Class_A中的属性。生成的代码可以简化如下：

Class_A:

```

'##ModelId=3734BEF00234
Private mNewProperty as Class_B

'##ModelId=3734BEF00235
Public Property Set NewProperty(ByVal vNewValue As Class_B)
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

'##ModelId=3734BEF00232

```

```

Public Property Get NewProperty() As Class_B
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set NewProperty = mNewProperty

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

```

Class_B:

```

'##ModelId=3734B5B70280
Private mNewProperty As Class_A

'##ModelId=3734B6FD00F5
Public Property Set NewProperty(ByVal vNewValue As Class_A)
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Set")
End Property

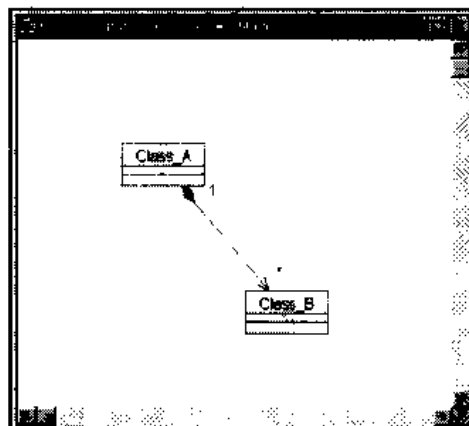
'##ModelId=3734BA8703B7
Public Property Get NewProperty() As Class_A
    On Error GoTo NewPropertyErr

    '## Generated default body ...
    Set NewProperty = mNewProperty

    Exit Property
NewPropertyErr:
    Call RaiseError(MyUnhandledError, "NewProperty Property Get")
End Property

```

值得一提的是，Visual Basic中累积与关联代码没有差别。

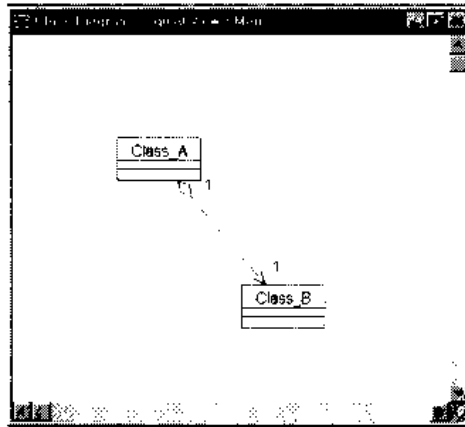


如果关系倍增性大于一，则Rose生成关系时使用容器类。

按引用累积

Visual Basic中按引用累积生成的代码与按值累积生成的代码相同。这里介绍的代码生成属性是ContainerClass作用属性。

例如，对前面的关系用按引用累积版本：



这时仍然是一对一关系，但按引用累积。生成代码的简化版本如下：

Class_A:

```

'##ModelId=3734BEF00234
Private mNewProperty as Class_B

'##ModelId=3734BEF00235
Public Property SetNewProperty(ByVal vNewValue As Class_B)
    On Error GoToNewPropPropertyErr
    '## Generated default body ...
    Set mNewProperty = vNewValue
    Exit Property
NewPropPropertyErr:
    Call RaiseError(MyUnhandledError, "NewPropProperty Property Set")
End Property

'##ModelId=3734BEF00232
Public Property GetNewProperty() As Class_B
    On Error GoToNewPropPropertyErr
    '## Generated default body ...
    SetNewPropProperty = mNewProperty
    Exit Property
NewPropPropertyErr:
    Call RaiseError(MyUnhandledError, "NewPropProperty Property Get")
End Property
  
```

Class_B:

```
'##ModelId=3734B5B70280
Private mNewProperty As Class_A

'##ModelId=3734B6FD00F5
Public Property SetNewProperty(ByVal vNewValue As Class_A)
    On Error GoToNewPropPropertyErr

    '## Generated default body ...
    Set mNewProperty = vNewValue

    Exit Property
NewPropPropertyErr:
    Call RaiseError(MyUnhandledError, "NewPropProperty Set")
End Property

'##ModelId=3734BA8703B7
Public Property GetNewProperty() As Class_A
    On Error GoToNewPropPropertyErr

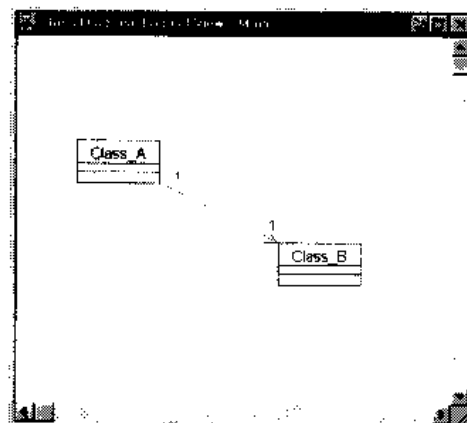
    '## Generated default body ...
    SetNewPropProperty = mNewProperty

    Exit Property
NewPropPropertyErr:
    Call RaiseError(MyUnhandledError, "NewPropProperty Get")
End Property
```

值得一提的是，按引用累积生成的代码与按值累积生成的代码相同。和其他关系一样，如果倍数大于一，则用容器类。

依赖性关系

依赖性关系不生成属性。如果Class_A和Class_B之间有依赖性，则Class_A和Class_B之间不生成属性。生成的代码如下：



Class_A:

```

Option Explicit

'##ModelId=3734C4EC0067
Private mClassDebugID As Long

'##ModelId=3734C4EC0066
Public Property Get ClassDebugID() As Variant
    On Error GoTo ClassDebugIDErr

    ClassDebugID = mClassDebugID

    Exit Property
ClassDebugIDErr:
    Call RaiseError(MyUnhandledError, "ClassDebugID Property")
End Property

```

Class_B:

```

Option Explicit

'##ModelId=3734C51D005A
Public mClassDebugID As Long

'##ModelId=3734C51C03D4
Public Property Get ClassDebugID() As Variant
    On Error GoTo ClassDebugIDErr

    ClassDebugID = mClassDebugID

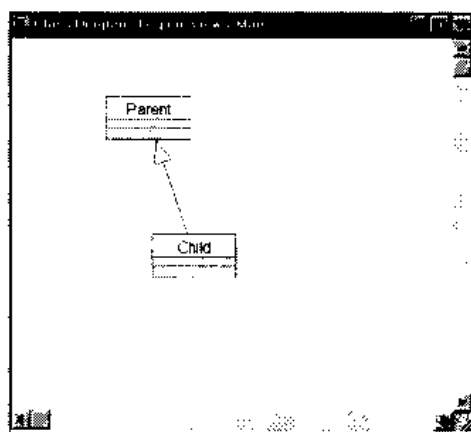
    Exit Property
ClassDebugIDErr:
    Call RaiseError(MyUnhandledError, "ClassDebugID Property")
End Property

```

Rose不在A中放上B的引用。

一般化关系

UML中的一般化关系在面向对象语言中变成继承关系。但Visual Basic不支持继承。Visual Basic中一般化关系变成实现委托。在Rose模型中，继承关系如下：



对于这种关系，Rose生成如下代码：

```
Option Explicit

'##ModelId=3734C5A40028
Public mClassDebugID As Long

'##ModelId=3734C5A303DE
Public Property Get ClassDebugID() As Variant
    On Error GoTo ClassDebugIDErr

    ClassDebugID = mClassDebugID

    Exit PropertyClassDebugIDErr:
    Call RaiseError(MyUnhandledError, "ClassDebugID Property")
End Property
```

Child (子) :

```
Option Explicit

'##ModelId=3734C577003C
Implements Parent

'##ModelId=3734C586014A
Private mParentObject As New Parent

'##ModelId=3734C58601BF
Private mClassDebugID As Long

'##ModelId=3734C58601BE
Public Property Get ClassDebugID() As Variant
    On Error GoTo ClassDebugIDErr

    ClassDebugID = mClassDebugID

    Exit Property
ClassDebugIDErr:
    Call RaiseError(MyUnhandledError, "ClassDebugID Property")
End Property
```

从代码中可以看出，父类中没有提到子类，使父类保持一般化，许多类可以从父类继承，而不影响父类的代码。

在子类型，生成实现父类功能的代码。实现委托如下：

```
Implements Parent
```

此外，子类包括父类的实例如下：

```
'##ModelId=36C66A3F00A0
Private mParentObject As New Parent
```

参数化类

Visual Basic中不能生成参数化类。

生成Visual Basic代码举例

本书一直使用ATM应用程序例子。本节生成ATM例子的框架Visual Basic代码，组件模型如图14.23所示。

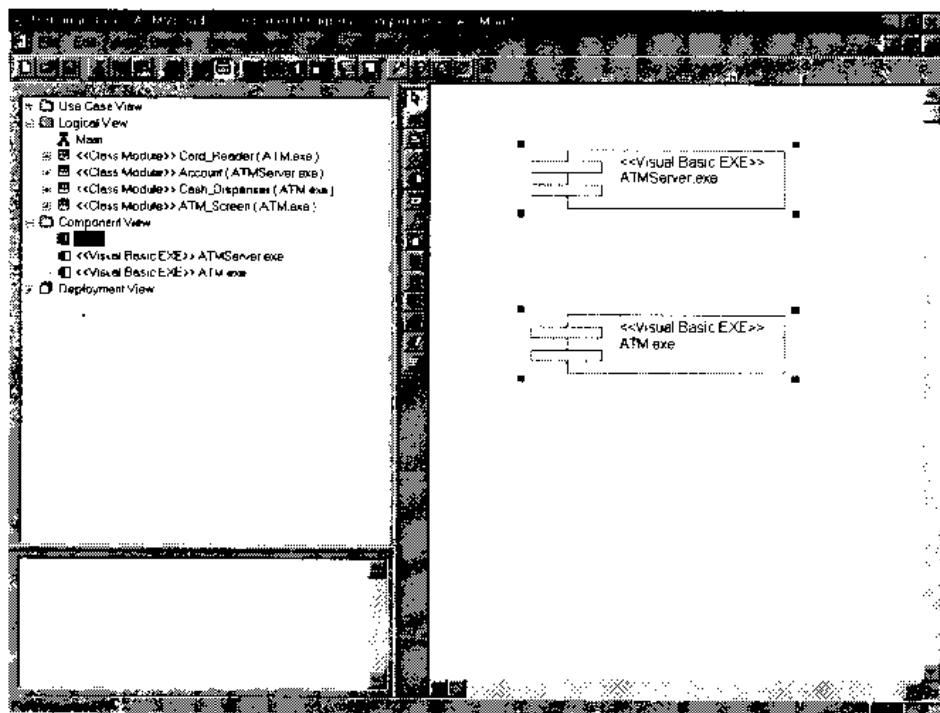


图14.23 ATM组件模型

对ATM系统，我们将ATM.exe和ATMServer.exe组件的语言设置为Visual Basic，然后将两个组件的版型设置为Visual Basic EXE。然后将所有类的版型设置为Class Module。我们选择Main Component框图中的ATM.exe和ATMServer.exe组件，然后选择Tools►Visual Basic►Generate Code。本书选配光盘中附有每个组件的代码。

练习

第3到第10章完成了订单输入系统的模型。现在要生成订单输入系统的代码。我们用图14.24所示的Component框图。要生成代码，步骤如下，本书选配光盘中包括了本练习的代码。

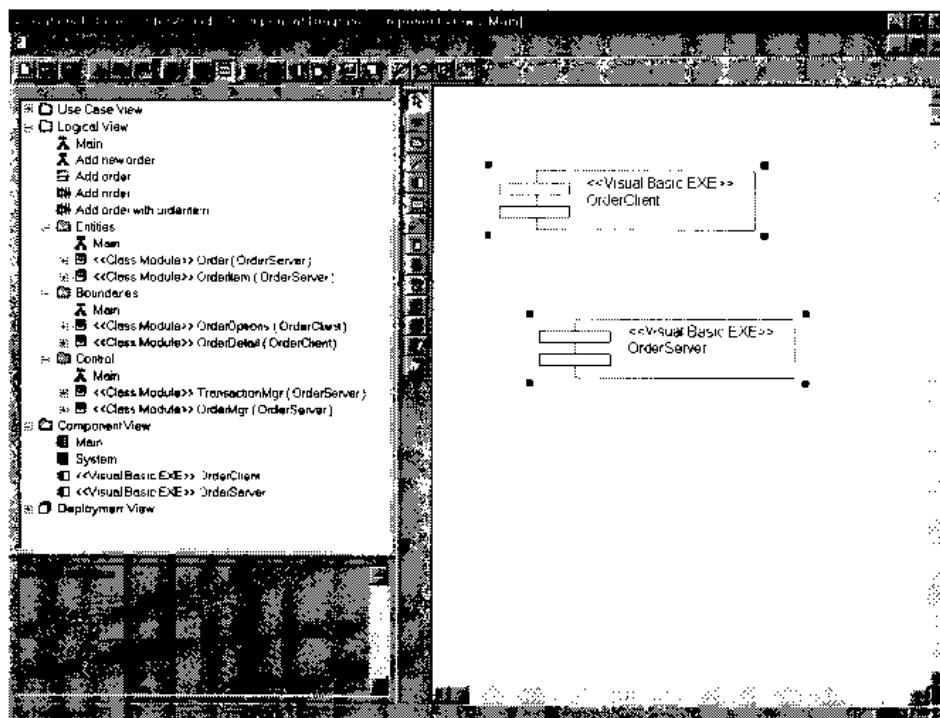


图14.24 订单输入系统的Component框图

练习步骤

将语言设置为Visual Basic

1. 打开Main Component框图。
2. 生成新组件OrderClient。
3. 打开新组件OrderServer。
4. 打开OrderClient的组件规范窗口。
5. 将语言设置为Visual Basic。
6. 单击Apply按钮。
7. 将OrderClient组件的版型设置为Standard EXE。
8. 对OrderServer组件重复第4到第7步。
9. 第9章对每个类生成了一个组件。Visual Basic不需要这个，因此要删除多余的组件。

打开System Component框图。

10. 选择框图中的所有组件。
11. 按Ctrl+D删除框图中的所有组件。
12. 打开Main Component框图。
13. 选择Boundaries、Entities和Control包。
14. 按Ctrl+D删除包及其所有组件。
15. 在浏览器中，将OrderDetail拖放到OrderClient组件上。
16. 对OrderOptions类重复第15步。
17. 在浏览器中，将Order类拖放到OrderServer组件上。

18. 对下列类重复第17步：
 - OrderItem
 - TransactionMgr
 - OrderMgr
19. 打开OrderDetail类的规范窗口。
20. 将版型设置为Class Module。
21. 对下列类重复第19和第20步：
 - OrderOptions
 - OrderItem
 - Order
 - TransactionMgr
 - OrderMgr

设置作用名

1. 打开Add Order Class框图。
2. 打开OrderDetail和OrderOptions间关联的属性。
3. 将Role A Name设置为theOrderDetail。
4. 将Role B Name设置为theOrderOptions。
5. 单击OK关闭属性窗口。
6. 对所有关联重复第2到第5步，用the加类名作为作用名。

生成Visual Basic

1. 选择菜单中的Tools►Visual Basic►Update Code。
2. 用向导生成这两个组件中所有类的代码。本书选配光盘中附有每个组件的代码。

小结

本章介绍了各个Rose模型元素如何用Visual Basic实现。利用类、包、属性、操作、关联、累加和其他Rose模型元素的代码生成属性，可以很好地控制生成的代码。

生成代码的步骤如下：

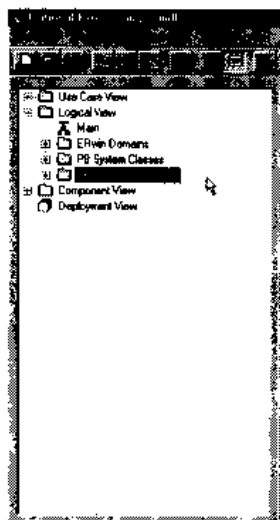
- 生成组件（见第10章）。
- 将类赋予组件（见第10章）。
- 设置代码生成属性。
- 选择Class或Component框图上的类或组件。
- 选择Tools►Visual Basic►Update Code（或Code Generation）开始代码生成向导。
- 选择Tools►Visual Basic►Browse Visual Basic Source浏览器生成的代码。

第15章 PowerBuilder代码生成

- 设置PowerBuilder代码生成属性
- 从Rose模型生成PowerBuilder代码
- 将Rose模型映射到PowerBuilder结构
- 从PB System Classes或PB Enumerated Types类派生类

PowerBuilder是Rose的语言插入件选项。如果有PowerBuilder插入件，就可以生成和逆向转出工程代码PowerBuilder代码。本章介绍如何从Rational Rose模型生成PowerBuilder代码。安装PowerBuilder插入件时，可以选择PowerBuilder5、6或6.5版本。本章的例子用PowerBuilder 6.5版本。但其他PowerBuilder版本所用的代码生成步骤和过程也是一样的。

安装PowerBuilder插入件后，Logical视图中有两个类包：PB System Classes和PB Enumerated Types。



从模型生成PowerBuilder代码的步骤与生成C++、Java或Visual Basic代码差不多，但稍有不同。具体地说，使用PowerBuilder时，所有类和属性都要有版型（stereotype）。另一个限制是所有类都要继承PB System Classes和PB Enumerated Types中的已知PowerBuilder类。这两个限制使Rose能确定对每个类生成哪种PowerBuilder对象（窗口、不可见对象等）。

生成PowerBuilder代码的步骤：

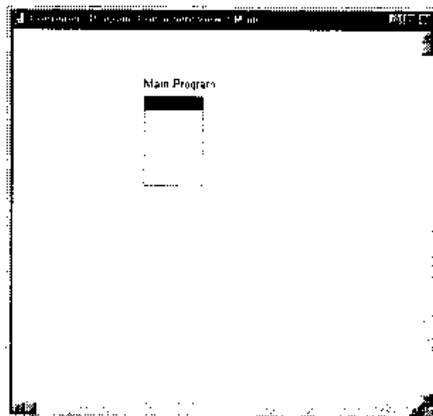
1. 生成主组件。
2. 生成其他组件。
3. 将类赋予组件。
4. 设置代码生成属性。
5. 对类指定版型。
6. 从PB System Classes和PB Enumerated Types中的类派生类。

7. 选择Class或Component框图中要生成的类或组件。

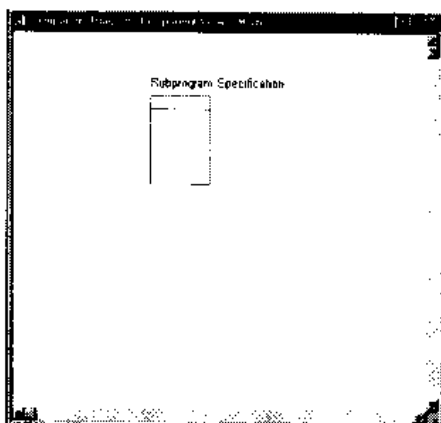
8. 选择Tools►PowerBuilder►Code Generation。

第一步要生成保存类的组件。在PowerBuilder中，可以将许多类映射到一个组件。每个组件成为一个PowerBuilder库（PBL）文件。

PowerBuilder中使用两种组件类型，第一种是主程序（Main Program）：



这是保存应用程序对象的PBL。要设置应用程序对象名，先生成版型为应用程序的类并将这个类与组件相关联。第二种组件是子程序规范（Subprogram Specification）。



这种组件是不包含应用程序对象的PBL文件。所有类都应映射到Main Program组件或一个Subprogram Specification组件。组件映射控制类在哪个PBL文件中生成。

下一步要设置代码生成属性。本章要介绍PowerBuilder代码生成属性，并介绍从Rose生成的代码。

生成代码之前，每个类应设置相应版型。PowerBuilder用版型设置确定PowerBuilder中生成的对象类型（窗口、用户对象、应用程序等）。每个类、属性和操作都应设置版型。表15.1列出了PowerBuilder版型。

每个类、属性和操作都设置版型后，最后要保证所有类均从PB System Classes或PB Enumerated Types中的类派生。继承关系使PowerBuilder插入件知道生成哪种PowerBuilder对象。例如，要生成不可见用户对象，就要从PB System Classes中的Nonvisualobject类继承。

我们经常会遇到，如何处理数据窗口的问题。处理数据窗口的最佳方法是用PowerBuilder生成，然后将其逆向转出工程代码回到Rose中。Rose不生成数据窗口对象。

表15.1 PowerBuilder版型

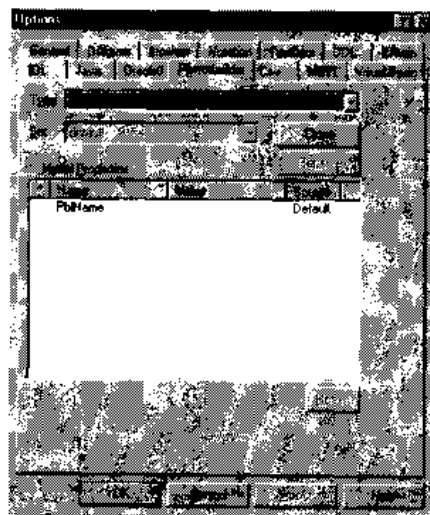
版型	适用模型元素	PowerBuilder类型
Application	类	应用程序对象
Menu	类	PowerBuilder菜单
Structure	类	PowerBuilder结构
System Class	类	PowerBuilder系统类
User Defined Object	类	用户对象
Window	类	窗口
Control	属性	PowerBuilder控件（按钮、文本框等）
Field	属性	结构元素
Property	属性	对象属性
Variable	属性	实例变量
EventExtend	属性	派生事件
EventOverride	操作	覆盖事件
EventOverrideInternal	操作	PowerBuilder中不可见的覆盖事件 （例如create和destroy）
Function	操作	PowerBuilder函数
Subroutine	操作	PowerBuilder子程序
UserEvent	操作	用户定义事件

PowerBuilder代码生成属性

和其他语言一样，PowerBuilder代码也是从Rose模型产生，由一系列代码生成属性控制。对PowerBuilder，Rose包括与属性、模块体和模块规范相关的属性。

要浏览和设置这些属性，选择Tools►Options，然后选择PowerBuilder标签。

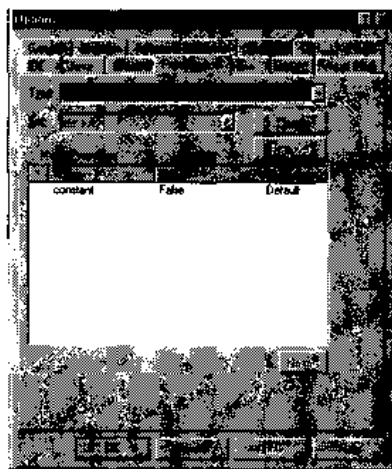
这个窗口中所作的任何改变都设置所有属性、模块体组件或模块规范组件的缺省值。



也可以设置单个属性、模块体组件或模块规范组件的代码生成属性。为此，打开模型元素的规范窗口并选择PowerBuilder标签，在这个标签中，可以改变特定元素采用的属性。下面几节介绍PowerBuilder代码生成属性。

属性属性

类中每个属性在PowerBuilder中生成一个实例变量。利用属性属性，可以控制实例变量是否常量。PowerBuilder属性只有一个代码生成属性。

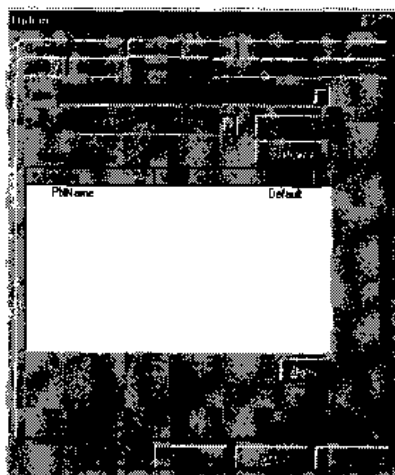


这个属性可以在两处设置。要对所有属性设置这个属性，选择Tools►Options，然后选择PowerBuilder标签，并从下拉列表框选择Attribute。要设置一个属性的属性，在属性规范窗口选择PowerBuilder标签，并在此编辑属性。

模块规范属性

模块规范属性与Rose模型中的模块规范组件相关。在PowerBuilder中，可以用子程序规范（Subprogram Specification）图标生成模块规范组件。

每个子程序规范组件表示PowerBuilder应用程序中的不同PBL文件。可以选择Tools►Options，然后选择PowerBuilder标签，并从下拉列表框选择Attribute。要设置一个属性的属性，在属性规范窗口选择Module Specification设置所有组件的属性。

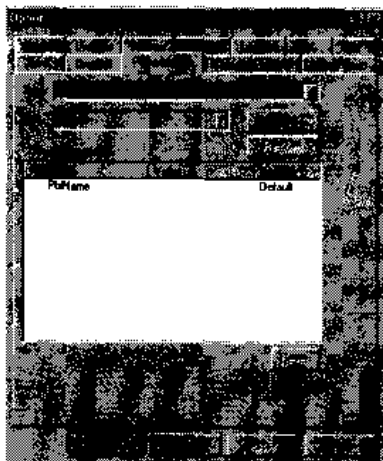


要设置一个文件的属性，在组件规范选择PowerBuilder标签。和属性一样，一个PowerBuilder子程序规范只有一个代码生成属性。PBLName属性设置要生成的PBL文件名。缺省情况下，文件名与组件名相同。

模块体属性

在PowerBuilder中，模块体是包含应用程序对象的组件，用主程序（Main Program）图标在组件框图中显示和生成。

利用代码生成属性可以设置生成的PBL文件名。要对所有主程序设置这个属性，选择Tools►Options，然后选择PowerBuilder标签，并从下拉列表框选择Module Body。要设置一个主程序的属性，在主程序规范窗口选择PowerBuilder标签，在此编辑属性。



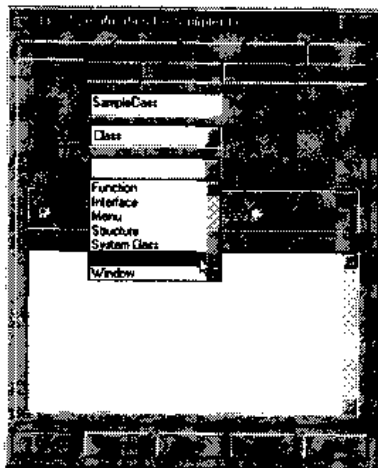
生成代码

下面几节介绍对不同类型模型元素生成的PowerBuilder代码。首先介绍独立生成一个类，然后介绍PowerBuilder插件件如何生成不同UML关系的代码。

Rose模型中每个类在PowerBuilder中生成一个类。一般来说，类的每个属性成为PowerBuilder中的一个实例变量，每个操作成为一个函数。但利用Rose模型中的不同版型可以生成不同类型的属性和操作。下面几节介绍不同版型如何影响生成的代码。

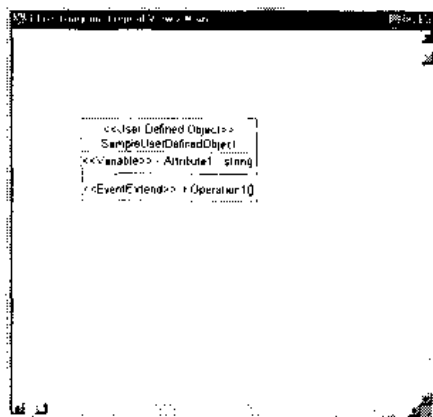
类

可以生成不同类型的类：应用程序对象、菜单、结构、系统类、用户定义对象、窗口。生成的类类型可以通过类规范窗口的版型设置：



生成不可见对象

下面介绍最常见的类类型——用户定义对象。这里有一个类SampleUserDefinedObject，具有一个属性和一个操作。



类及其属性和操作都有版型，但必须先完成上述所有步骤之后才能生成代码。类首先映射组件，控制类在哪个PBL文件中生成。图15.1显示了主程序组件SamplePB和映射这个组件的类。

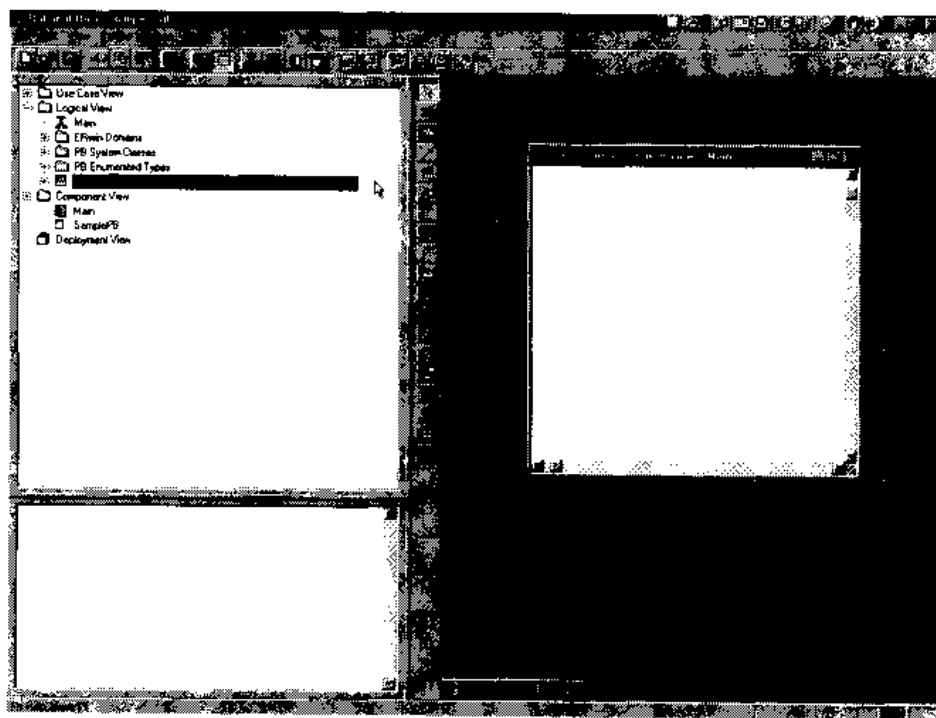


图15.1 将PowerBuilder类映射到组件

生成代码之前的最后一步是从一个已知PowerBuilder类派生这个类。本例中，要生成不可见PowerBuilder对象，我们从PB System Classes包中的不可见对象类派生这个类，如图15.2。

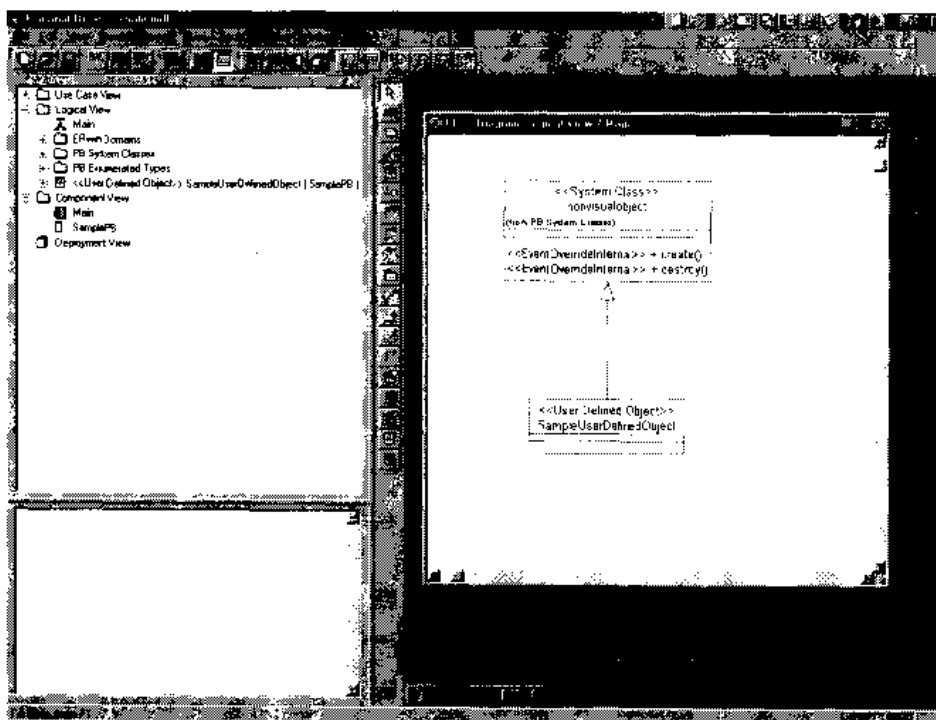


图15.2 从已知PowerBuilder类派生这个类

许多Entity类、Control类、Utility类和Helper类都实现为不可见PowerBuilder对象。模型中的Boundary类通常实现为PowerBuilder窗口对象。这里由于要生成不可见对象，因此在代码中生成不可见PowerBuilder对象，如图15.3。

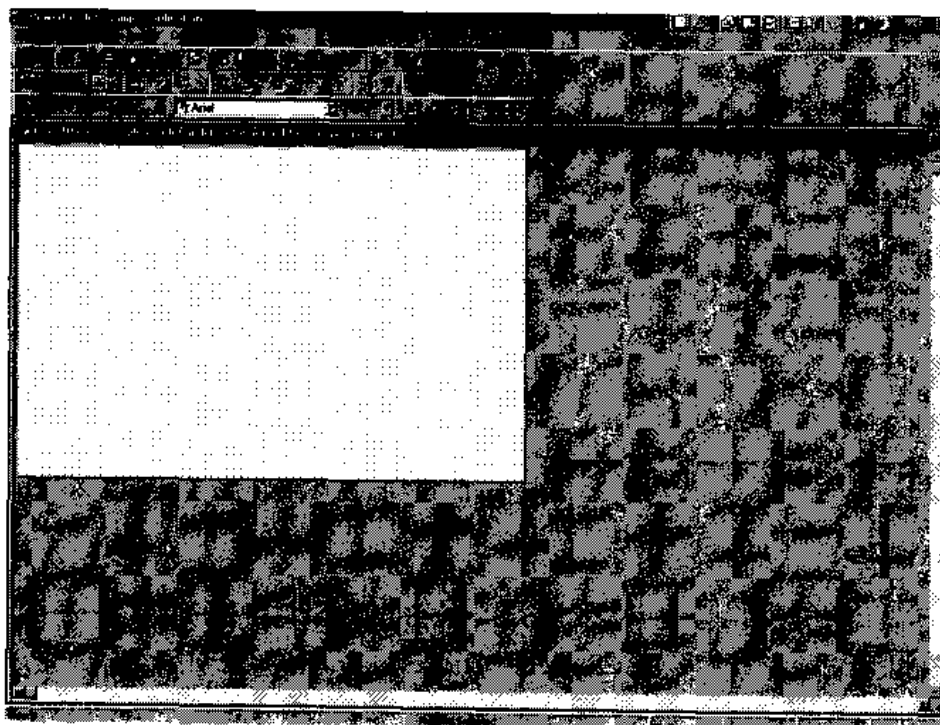


图15.3 生成不可见PowerBuilder对象

生成窗口

另一个常用PowerBuilder版型是窗口（Window）。在Rose中，可以对类指定Window版型，从而生成PowerBuilder窗口。类的属性成为窗口的实例变量，操作成为函数或事件。

要生成窗口，首先对类指定Window版型，然后从PB System Classes中的Window类派生这个类。Class框图如图15.4所示。

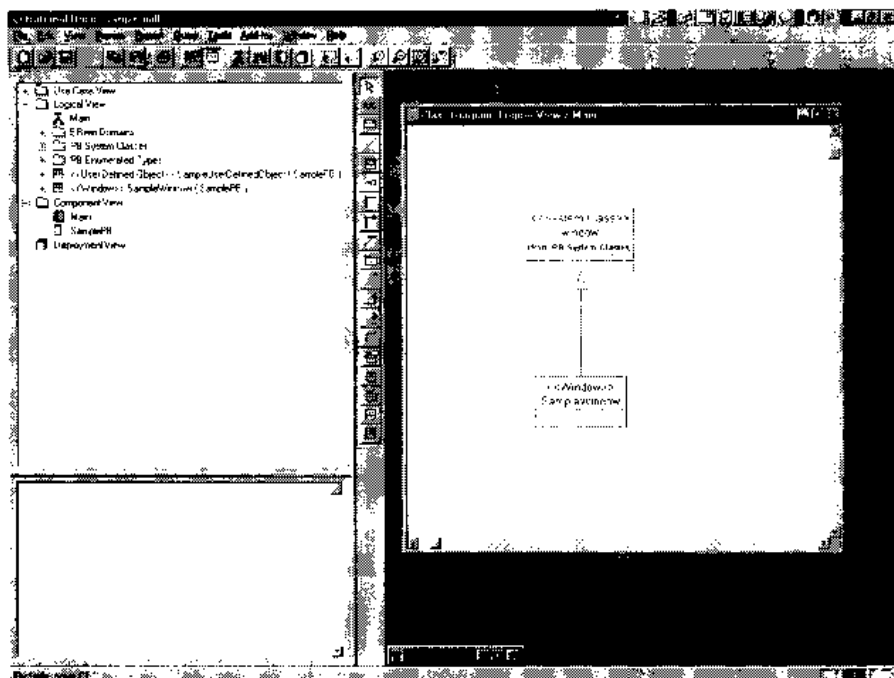


图15.4 正在生成PowerBuilder窗口

代码生成完毕后，PowerBuilder文件中即可以有Window对象。对上图所示窗口，图15.5显示了PowerBuilder中生成的窗口。

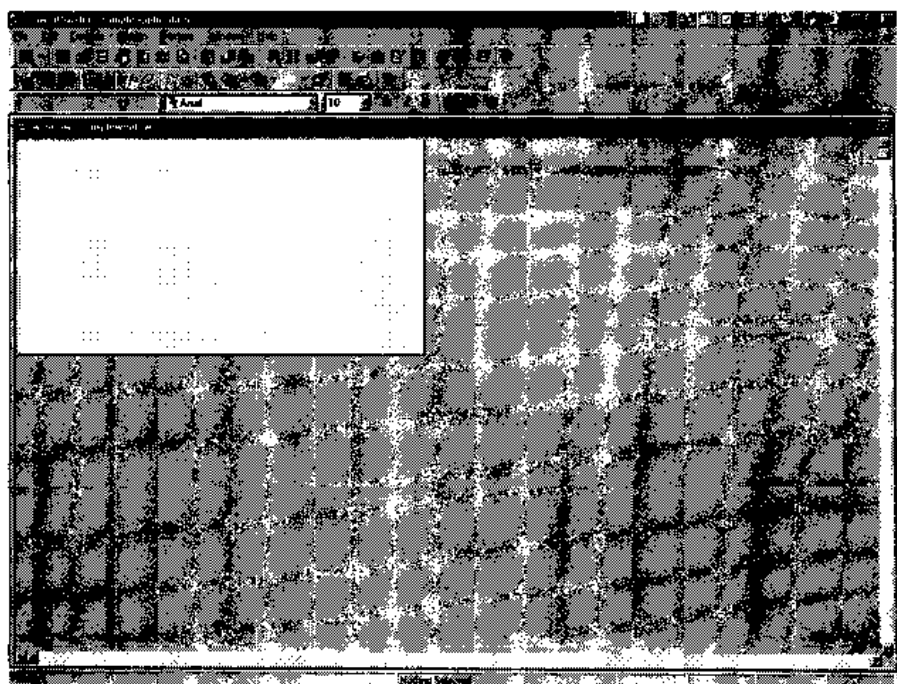


图15.5 生成的PowerBuilder窗口

生成应用程序对象

每个PowerBuilder应用程序都要有一个应用程序对象。Rose中可以通过指定版型的类生成应用程序对象。选择要生成应用程序对象的类并将其版型设置为Application。

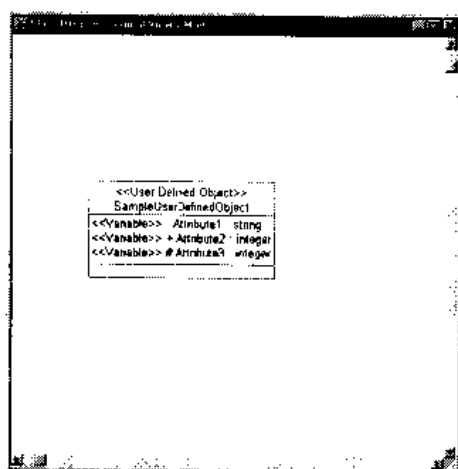
生成代码时，这个类成为PowerBuilder应用程序。

属性

类中的属性生成实例变量、共享变量或字段。字段只对结构生成。其他类型的类包括实例变量或共享变量。

生成代码时，Rose使用属性名称、版型、数据类型和缺省值。所有这些信息都会包括在PowerBuilder代码中。

属性的可见性会体现在生成的代码中。PowerBuilder直接支持public、private和protected属性。图15.6显示了对下图所示类生成的属性。



如果Rose模型中的属性为静态，则生成共享变量。

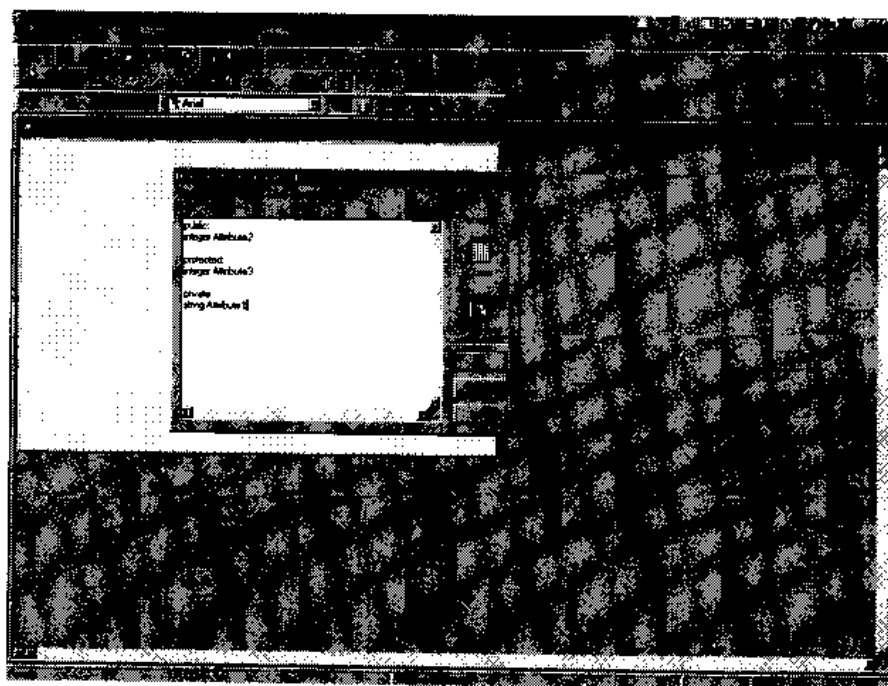


图15.6 PowerBuilder代码中的属性

操作

类中的每个操作可以实现为PowerBuilder函数或事件。操作版型确定操作如何实现。表15.2列出了几种可能的操作版型。

表15.2 操作版型

操作版型	适用模型元素	PowerBuilder类型
EventExtend	操作	扩展事件
EventOverride	操作	覆盖事件
EventOverrideInternal	操作	在PowerBuilder中不可见的覆盖事件 (例如create和destroy)
Function	操作	PowerBuilder函数
Subroutine	操作	PowerBuilder子程序
UserEvent	操作	用户定义事件

假设对用户定义事件生成代码。下图所示类的操作版型为EventExtend:



从图15.7可以看出，Rose生成用户事件以支持这个操作。

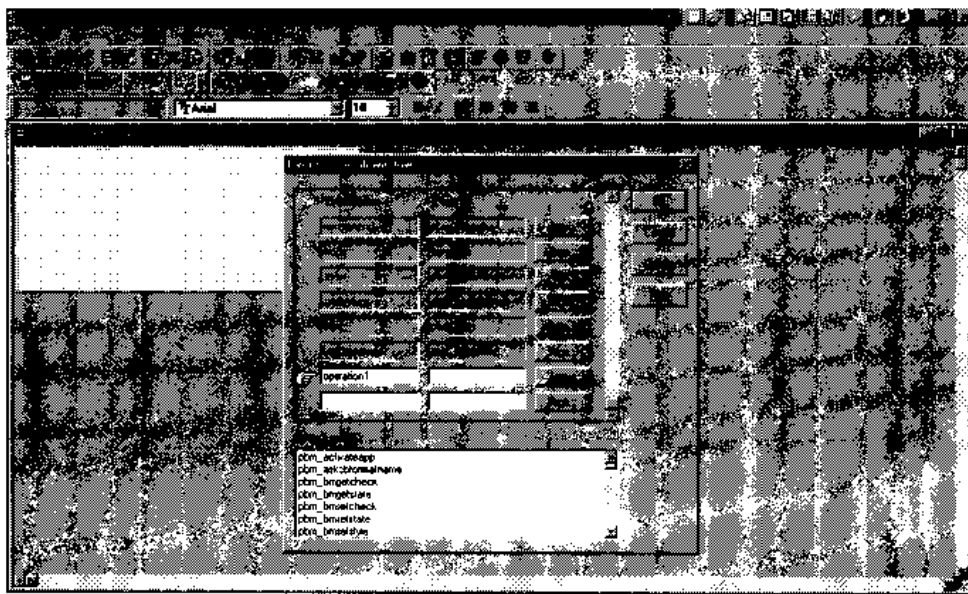
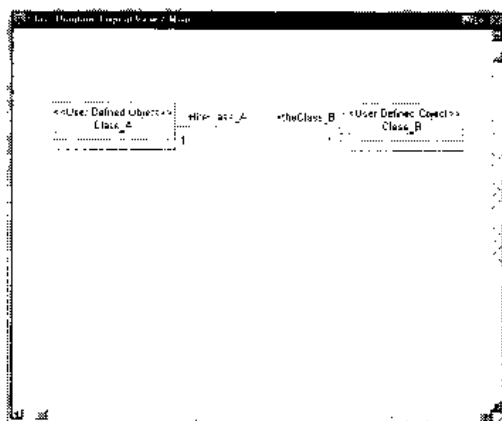


图15.7 生成用户事件

如果操作版型设置为函数或子程序，则生成代码之前还要完成其他步骤。在操作的文档窗口，输入PowerBuilder返回语句，如Return 0。输入的语句应符合有效PowerBuilder语法，返回值的数据类型应与操作的返回类型相同。生成代码时，返回语句在函数内生成。

双向关联

在双向关联中，Rose生成支持关系的属性。关系中的每个类有一个新属性，指向其他类。例如，以下列关系为例：



本例中，这两个类在生成代码时都得到新属性。如果对关系设置作用名，则作用名可以作为属性名。上例中，对A类生成的属性如下。图15.8显示了双向关联生成的实例变量。

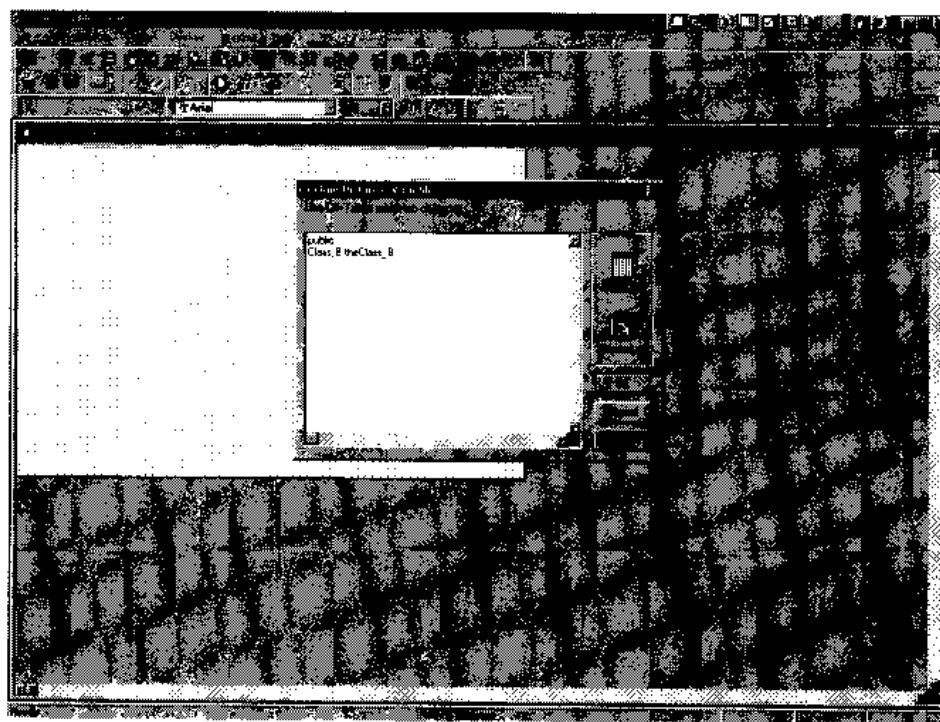


图15.8 PowerBuilder中的双向关联

关联名、限定符和限制是Rose模型中的有用信息，但不影响生成的代码。

单向关联

单向关联与双向关联相似，但只对关系一端生成属性。下例中在A类内而不在B类中生成属性。

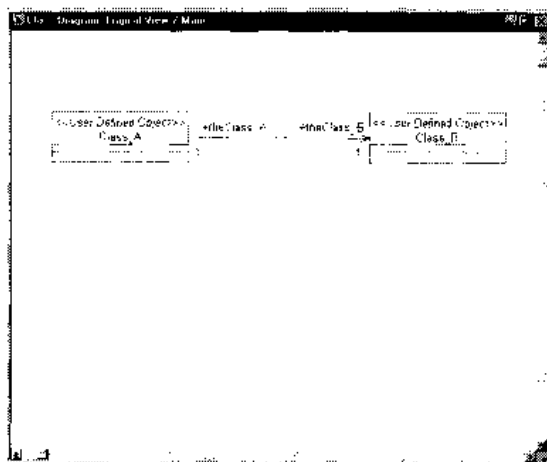


图15.9显示了A类生成的属性。注意它与双向关联中A类生成的属性相同。

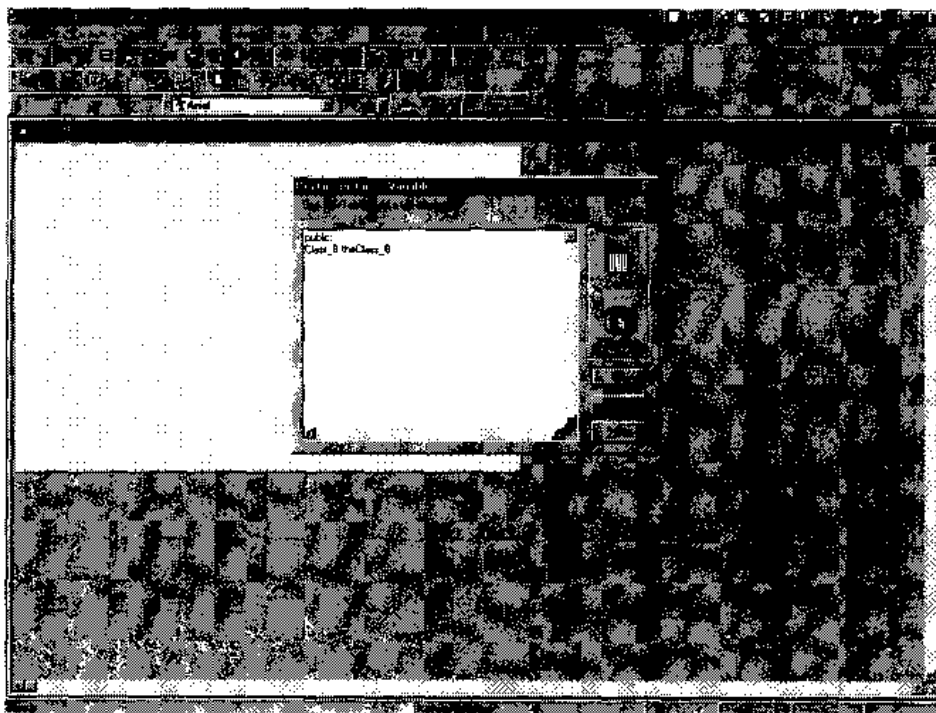
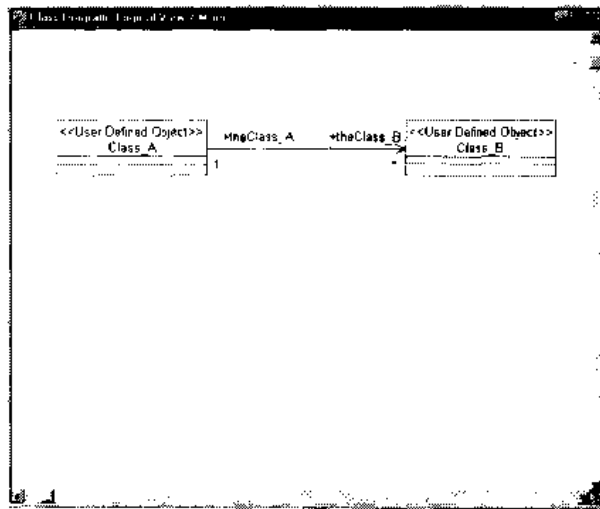


图15.9 PowerBuilder中的单向关联

前面介绍一对一关系，下面要介绍倍增性对关系的影响。

倍增性为一对多的关联

在一对多关系中，Rose像一对一关系中一样生成相应属性。如果关系是双向的，则两端生成属性；如果关系是单向的，则只在一端生成属性。一对多关系的差别在于支持关系多端的属性要使用某种容器类。对于PowerBuilder，Rose用数组作为容器类。举例如下。



本例中，属性在A类中生成。由于倍增性为一对多，因此用数组作为属性类型。图15.10显示了A类中生成的支持关系的实例变量。

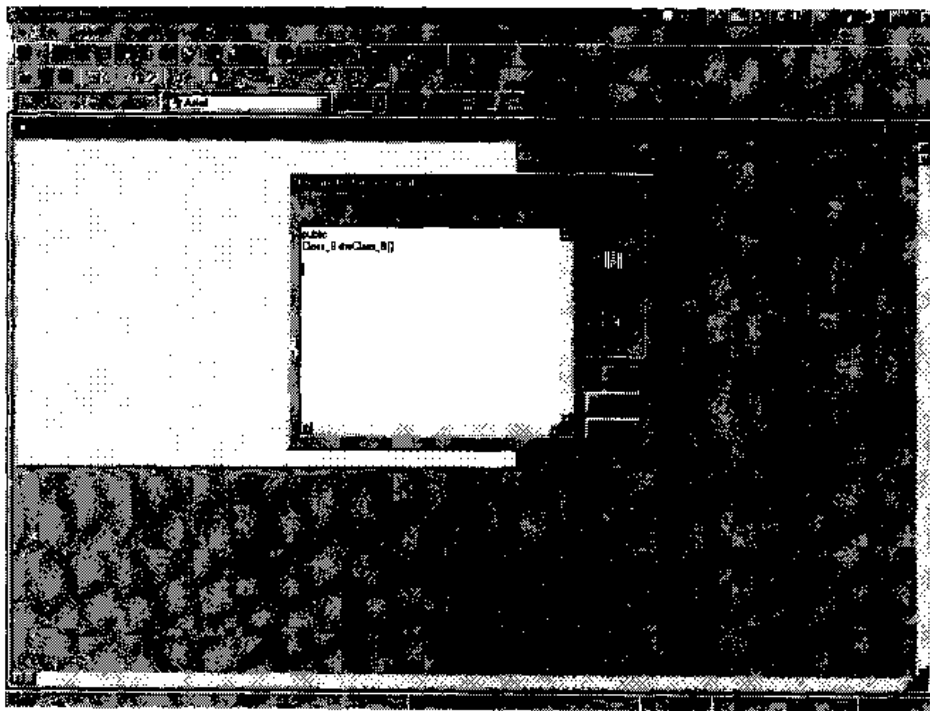


图15.10 PowerBuilder中的一对多关系

倍增性为多对多的关联

在多对多关系中，关系两端的属性都用容器类。上例中，如果倍增性为多对多，则Class_A和Class_B都用数组作为实例变量。

图15.11显示了A类中生成的实例变量。

图15.12显示了B类中生成的实例变量。

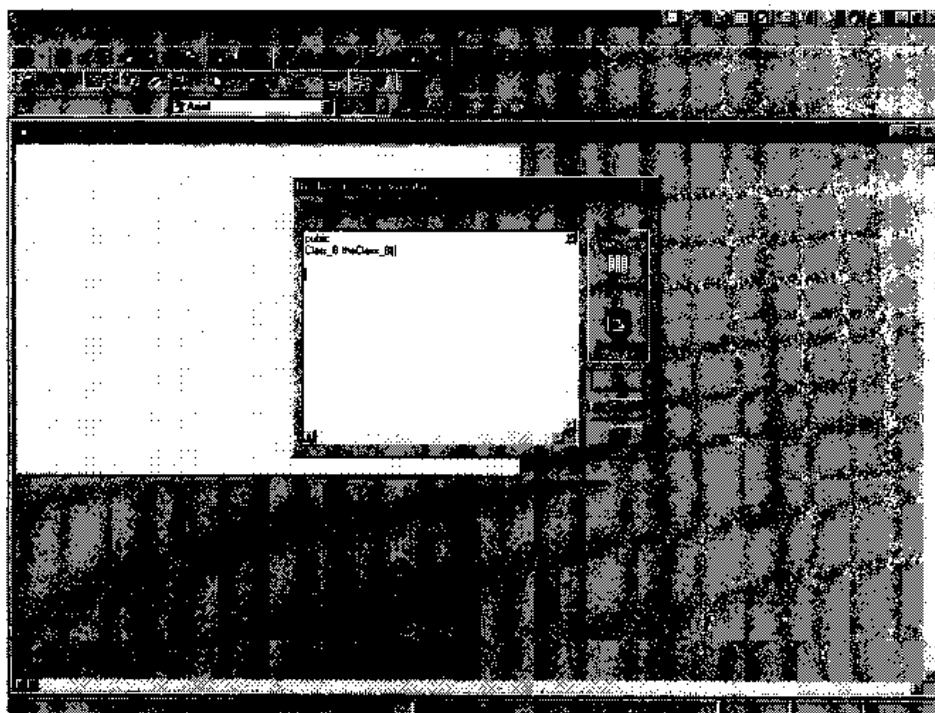


图15.11 多对多关系——A类

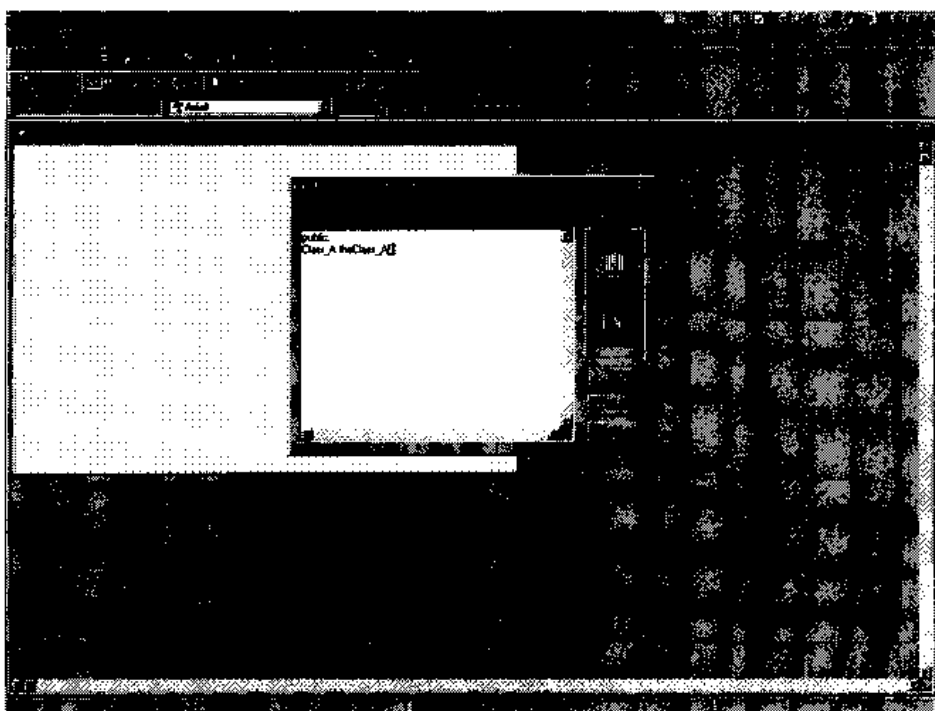
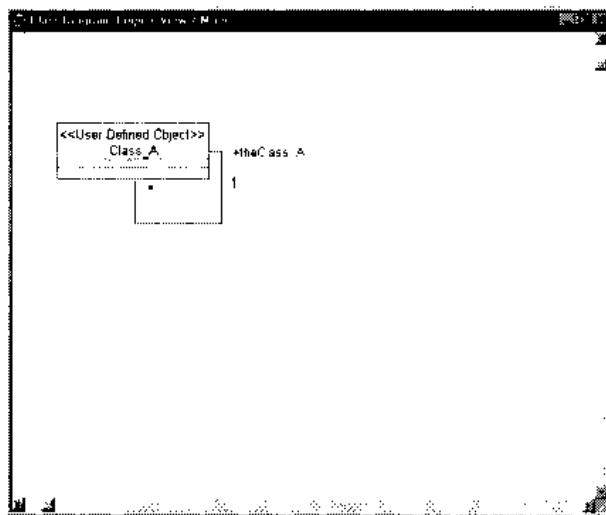


图15.12 多对多关系——B类

反身关联

在反身关联中，像普通关联一样生成实例变量。如果A类有反身关联，则它具有类型为A的实例变量。以下图中的代码为例：



上例显示了支持关系的实例变量。

累积

在PowerBuilder中，关联与累积没有差别。双向累积和双向关联一样实现，单向累积和单向关联一样实现。

尽管累积在PowerBuilder中没有意义，但在Rose中是重要和有用的建模工具。累积常用于建模窗口及其控件之间的关系。每个控件可以显示单个类，通过累积与窗口连接。

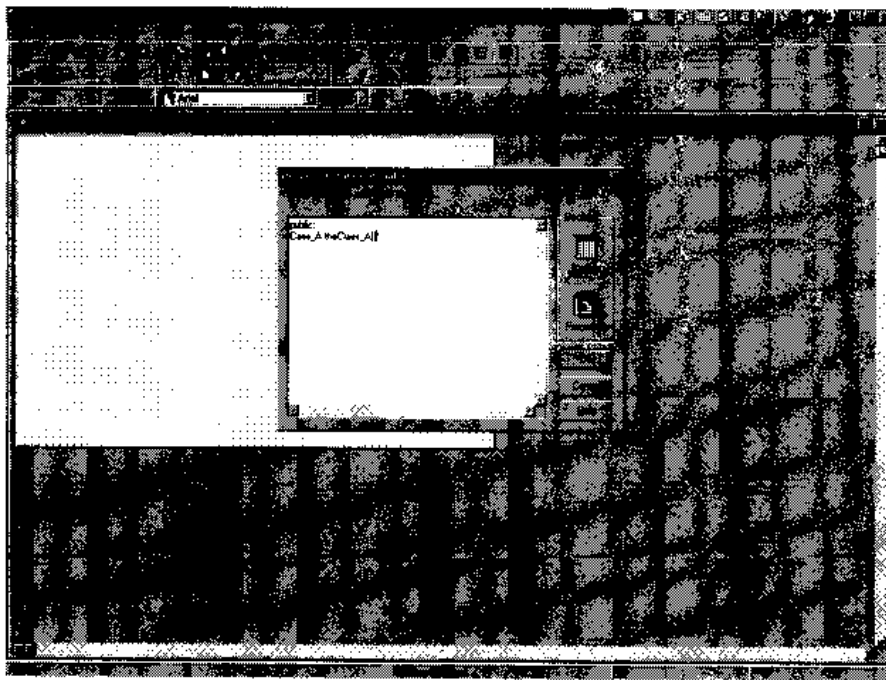


图15.13 在PowerBuilder中实现反身关联

依赖性关系

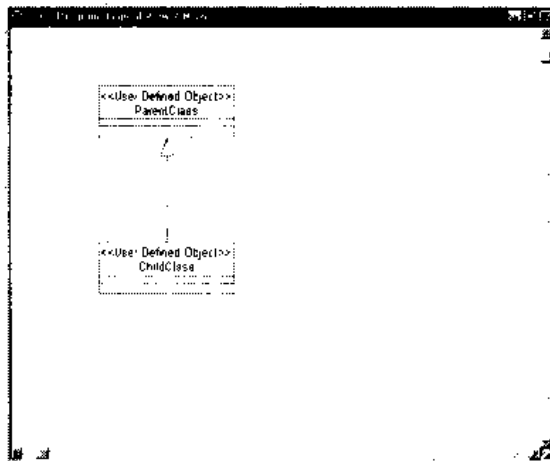
依赖性关系不生成属性。对依赖性关系生成PowerBuilder代码时，哪个类都不接收支持关系的实例变量。

一般化关系

PowerBuilder直接支持继承的概念。Rose模型中的任何一般化关系均在PowerBuilder中实现为继承关系。

所有类都要从某个类派生之后才能生成代码。继承结构的根应为PB System Classes或PB Enumerated Types中的已知PowerBuilder类。模型中的所有类都应由此派生。

下面举例说明。



本例中，ParentClass类由ChildClass类派生。这两个类都是非可见PowerBuilder对象。从图15.14可以看出，Rose在代码中实现为继承关系。

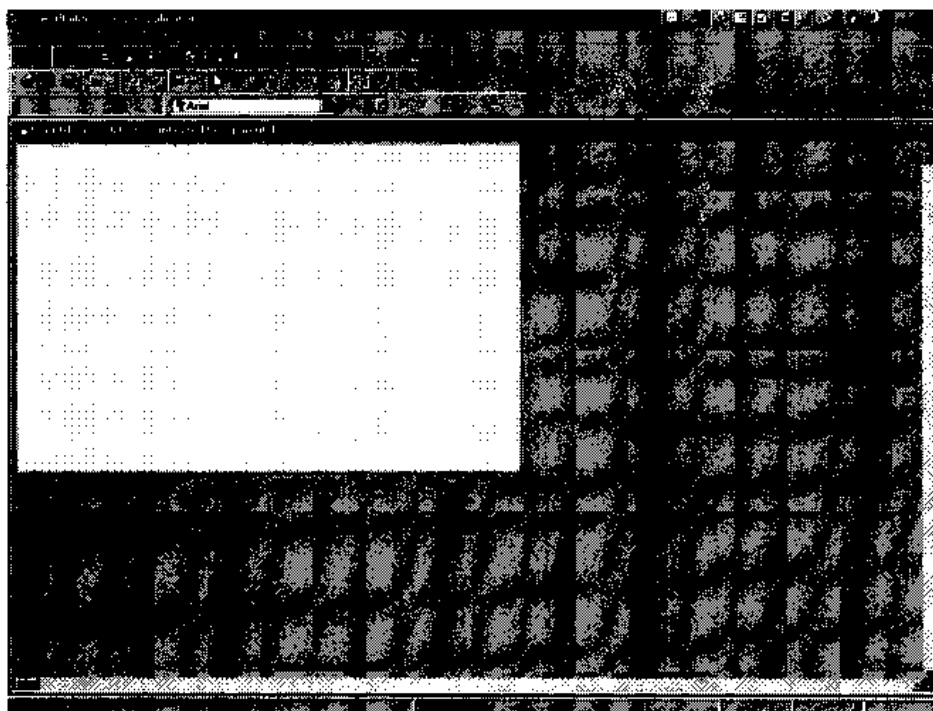


图15.14 PowerBuilder代码中的一般化关系

生成PowerBuilder举例

本书多次提到ATM例子应用程序。本节生成了ATM例子框架的PowerBuilder代码，其组件模型见图15.15和图15.16。

对于ATM Client Component，将所有组件的语言设置为PowerBulder。然后选择Main Component框图中的ATMServer和ATMClient包，再选择Tools►PowerBuilder►Code Generation。

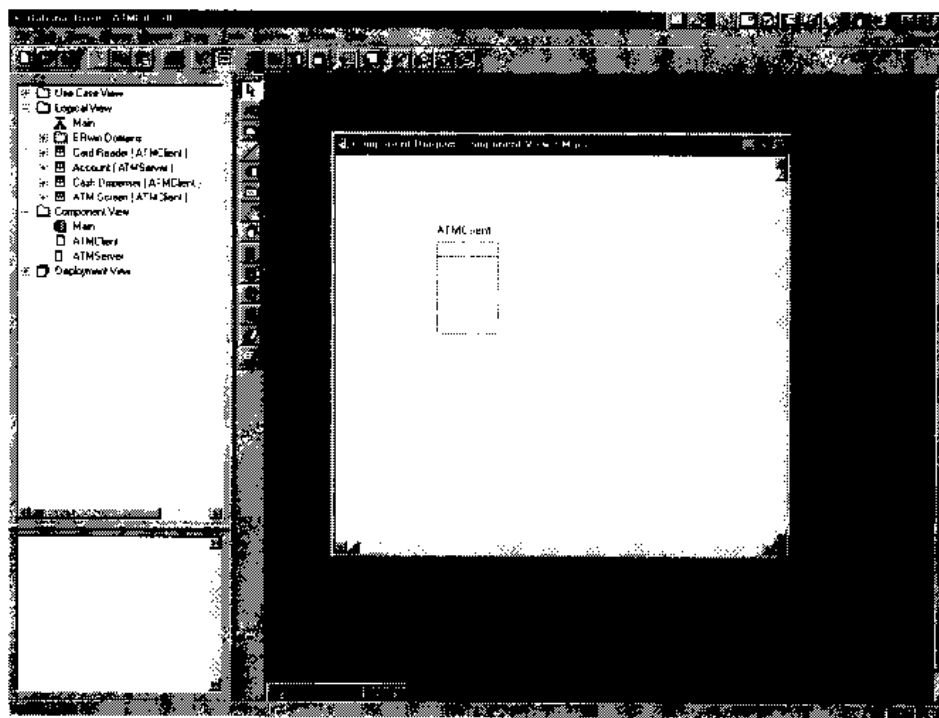


图15.15 ATM Client Component模型

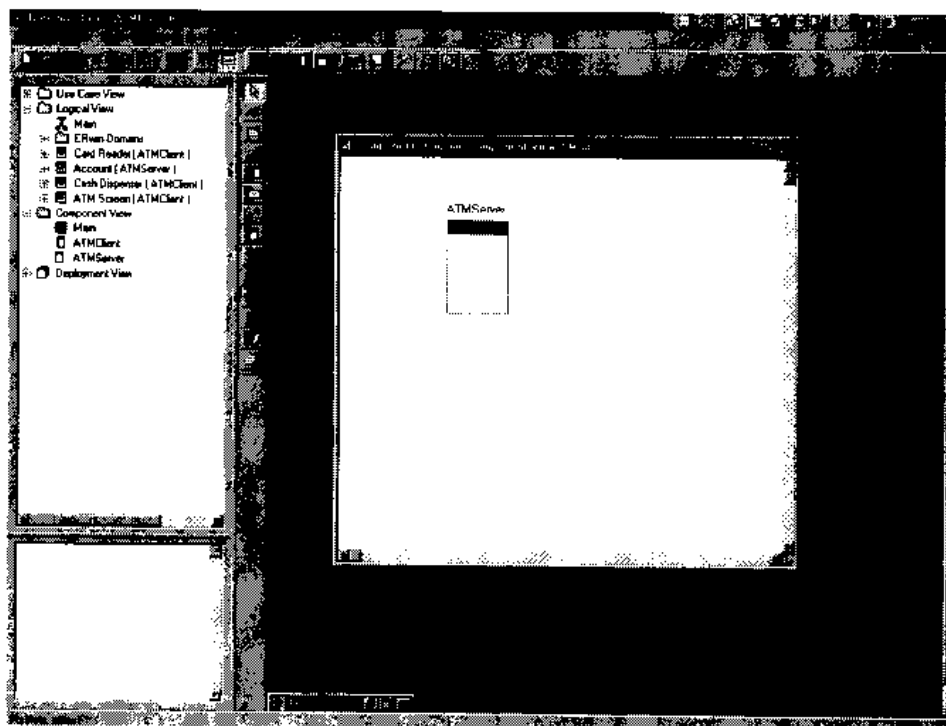


图15.16 ATM Sever Component模型

PowerBuilder中生成下列类。图15.17显示了生成所有类之后的PowerBuilder库。

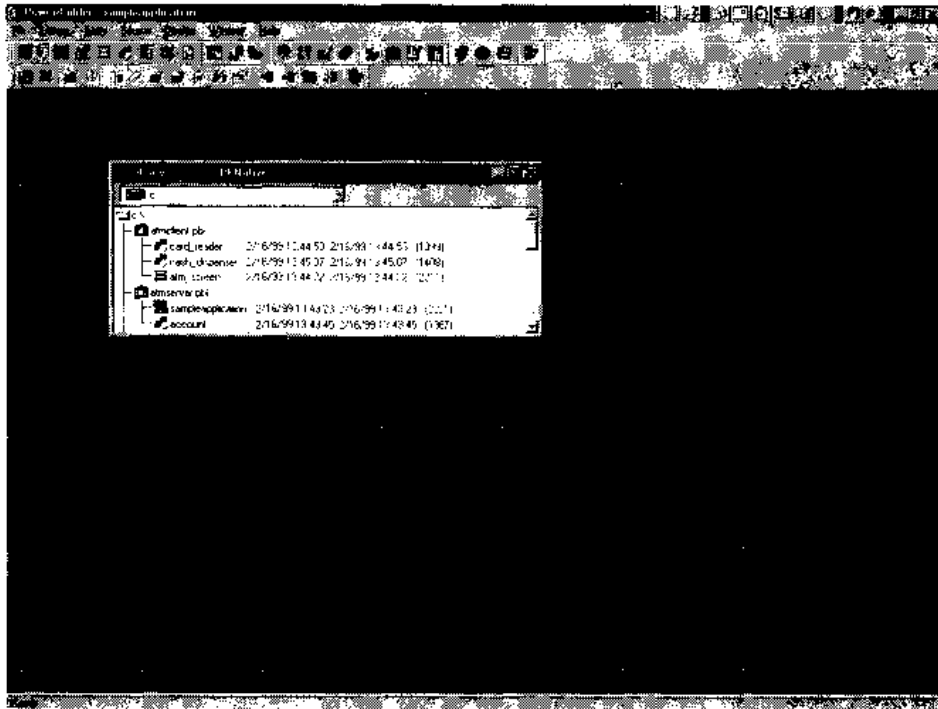


图15.17 ATM例子生成的代码

练习

第3章至第10章完成了订单输入系统的模型。现在要生成订单输入系统的代码。我们用图15.18所示的Component框图。要生成代码，步骤如下。

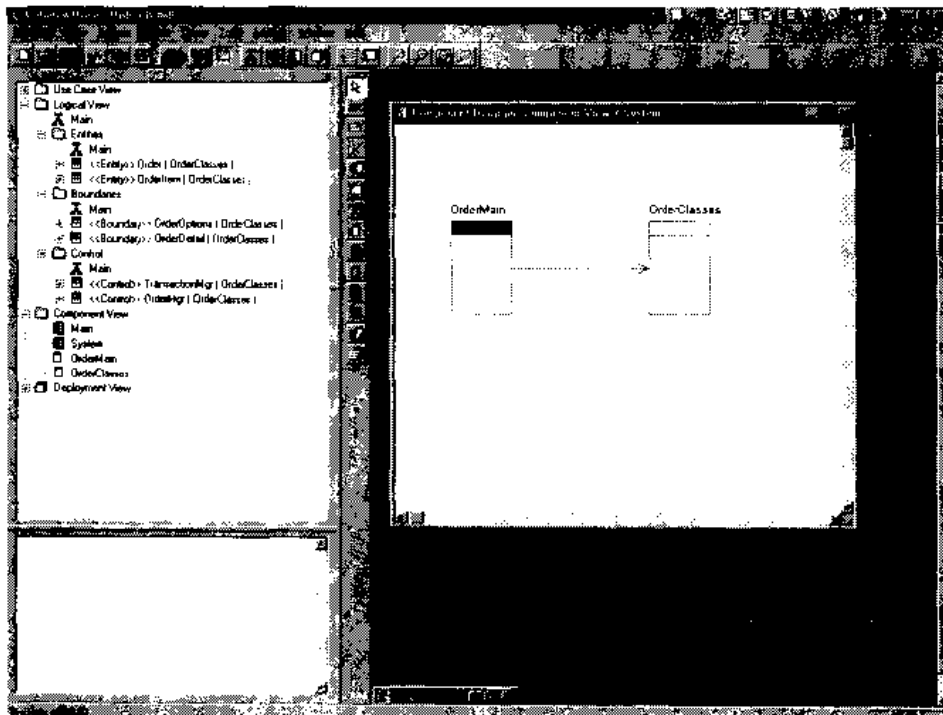


图15.18 订单输入系统的Component框图

练习步骤

将语言设置为PowerBuilder

1. 打开System Component框图。
- 第9章对每个类生成了一个组件。PowerBuilder不需要这些组件，因此要删除多余的组件。
2. 选择框图中的所有组件。
3. 按Ctrl+D删除框图中的所有组件。
4. 打开Main Component框图。
5. 选择Boundaries、Entities和控制包。
6. 按Ctrl+D删除这些包及其中包含的所有组件。
7. 生成新的Main Program组件OrderMain。
8. 打开OrderMain组件规范窗口。
9. 将语言设置为PowerBuilder。
10. 在浏览器中，将OrderItem类拖放到OrderMain组件上。
11. 对下列类重复第10步：
 - Order
 - OrderOptions
 - OrderDetail
 - TransactionMgr
 - OrderMgr

设置数据类型

打开Add Order Class框图。前几章使用Boolean数据类型，由于PowerBuilder不认识Boolean数据类型，因此要先修改之后才能生成代码。

生成PowerBuilder

1. 选择OrderMain和OrderClasses组件。
 2. 选择菜单中的Tools►PowerBuilder►Code Generation。
 3. 启动PowerBuilder并在OrderMain.pbl中打开应用程序。PowerBuilder库如图15.19所示。
-

小结

本章介绍从Rose模型的类、属性和操作生成PowerBuilder代码。要生成PowerBuilder代码步骤如下：

1. 生成主组件。
2. 生成其他组件。
3. 将类赋予组件。

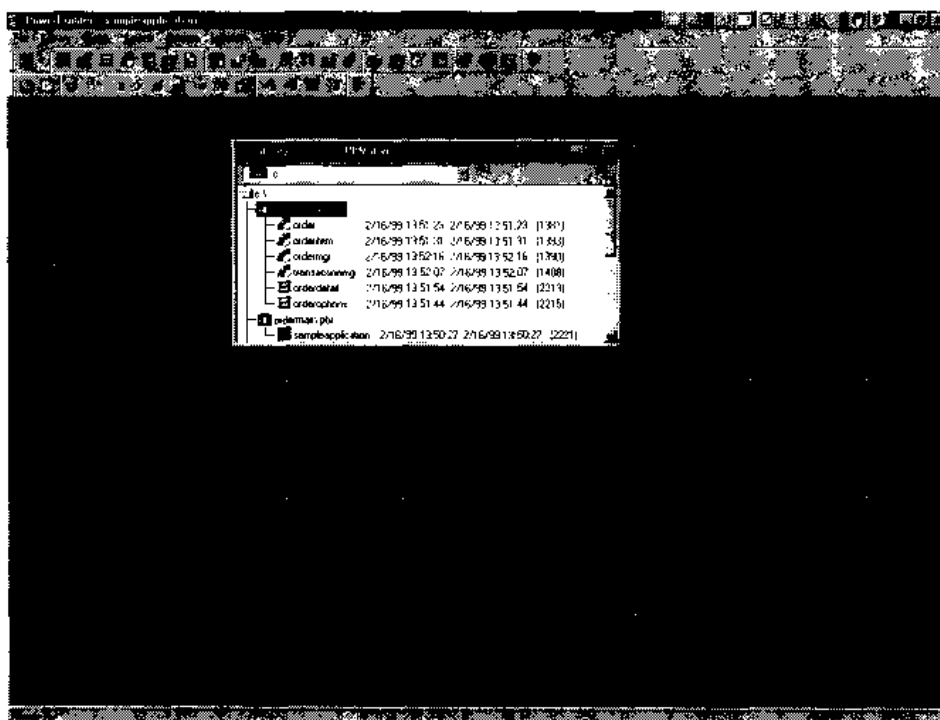


图15.19 订单输入系统的PowerBuilder代码

4. 设置代码生成属性。
5. 对类指定版型。
6. 从PB System Classes和PB Enumerated Types中的类派生类。
7. 选择Class或Component框图中要生成的类或组件。
8. 选择Tools>PowerBuilder>Code Generation。

生成代码时，Rose使用Rose模型的类、属性和操作组件。版型在PowerBuilder生成中特别重要，版型帮助Rose确定生成哪种PowerBuilder对象。与其他语言的代码生成不同，PowerBuilder要求先派生所有类并对所有类、属性和操作指定版型之后再生成代码。

第16章 CORBA/IDL代码生成

- 设置IDL代码生成属性
- 从Rose模型生成IDL代码
- 将Rose元素映射到IDL结构

本章介绍如何从Rational Rose模型生成Interface Design Language (CORBA/IDL) 代码。要从模型生成CORBA/IDL代码, 步骤如下:

1. 选择CORBA (98i) 或IDL (98) 代码生成属性。
2. 选择Class或Component框图中要生成的类或组件。
3. 选择Tools►CORBA►Generate CORBA (98i) 。
4. 选择Tools►IDL►Generate IDL (98) 。

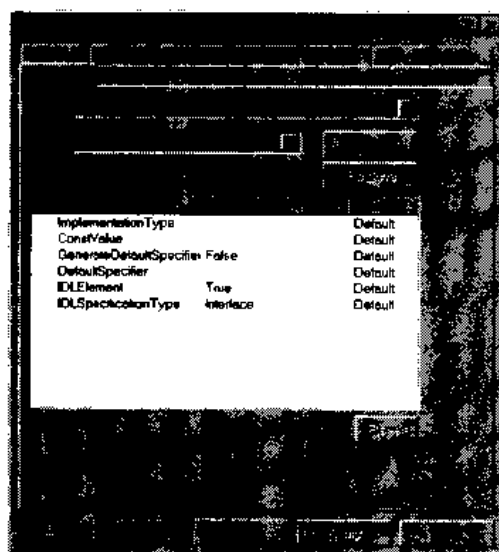
我们还将介绍类、属性和操作以及其他模型元素的各种CORBA/IDL代码生成属性, 并介绍从这些模型元素生成的CORBA/IDL。从Rose可以生成几种不同类型的CORBA/IDL: 接口、TypeDef、枚举、常量、异常、结构和联合。我们将分别介绍如何生成这几种不同类型的CORBA/IDL。

CORBA/IDL代码生成属性

从Rose模型生成的CORBA/IDL由一系列代码生成属性控制。Rose包括属性、类、依赖性、累积、模块体、模块规范、操作、关联、子系统和整个项目的属性设置。

在Rose 98i中, CORBA/IDL选项在各个菜单中标为CORBA。在Rose 98中, 这些选项标为IDL。为简单起见, 我们将菜单项目显示为CORBA。对Rose 98, 只要换成IDL即可。

要浏览和设置这些属性, 选择Tools►Options, 然后选择CORBA标签。



这个窗口中所作的任何改变都设置所有类、属性、操作等的缺省值。

也可以对单个类、属性、操作和其他模型元素设置代码生成属性。为此，打开模型元素的规范窗口，并选择CORBA标签。在这个标签上，可以改变适用于各个模型元素的属性。下面几节介绍CORBA代码生成属性。

项目属性

项目属性是适用于整个项目而不是适用于具体模型元素（如类或关系）的CORBA代码生成属性。

本节的选项包括生成代码时使用的缺省目录和代码生成期间可以发生的最大错误次数，表16.1列出了各个项目属性及其用途和缺省值。

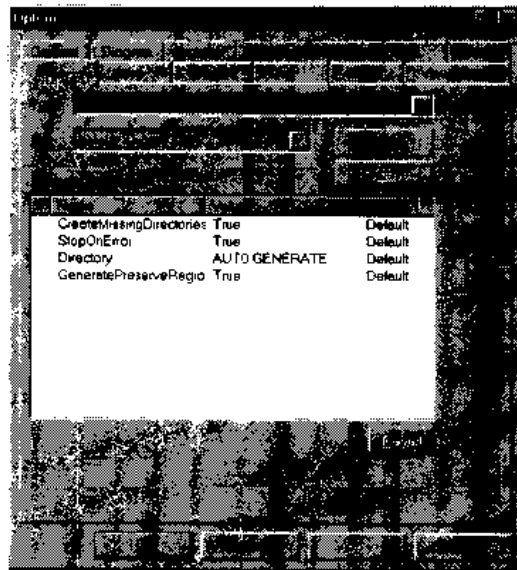
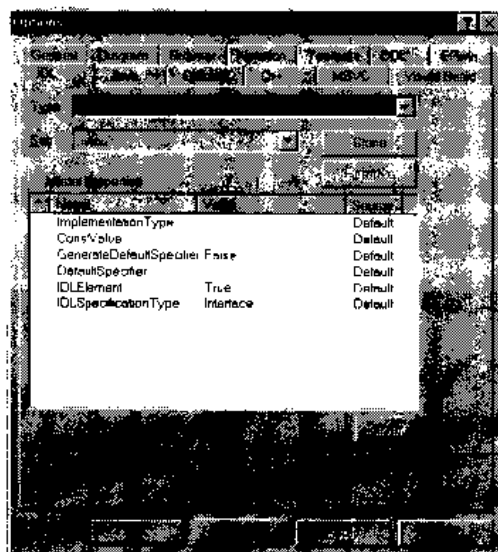


表16.1 项目CORBA生成属性

属性	用途	缺省
CreateMissingDirectoris	控制Rose是否在生成代码时生成映射包的目录	True
Editor (98i)	控制用哪个编辑器浏览和编辑CORBA文件	BuiltIn (用内置CORBA编辑器)
IncludePath (98i)	代码生成和逆向转出工程代码期间解析IDL文件地址所用的路径	空
StopOnError	控制Rose遇到错误时是否停止生成代码	True
Directory (98)	设置代码生成根目录，所有目录和CORBA文件均在其中生成	缺省情况下，Rose用路径映射中带符号\$ROSECPP_SOURCE的目录
GeneratePreserveRegions (98)	控制生成的CORBA中是否包括保留区，在逆向转出工程代码期间保护其中的代码	True

类属性

类属性是适用于类的CORBA代码生成属性。这些属性可以控制类是否生成IDL代码、确定CORBA类型和设置其他类特定属性。



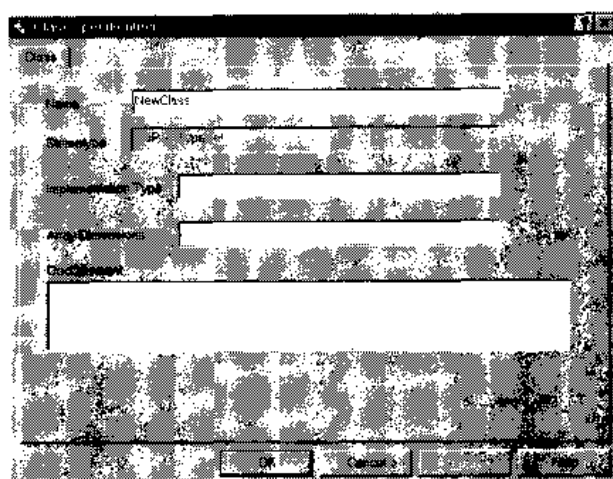
这些属性可以在两处设置。要设置所有类的属性，选择**Tools►Options**，然后选择**CORBA**标签，并从下拉列表框选择**Class**。要为一个类设置属性，选择类规范窗口的**CORBA**标签并编辑属性。

表16.2列出了CORBA类属性及其用途和缺省值。

表16.2 CORBA/IDL类代码生成属性

属性	用途	缺省
ArrayDimensions (98i)	类为TypeDef时设置类定义所用数组的维度	Empty
ImplementationType	这个值的用途随类的版型不同而不同。对于CORBAConstant，数值表示常量的数据类型；对于CORBATypeDef，数值表示数据类型；对于CORBAUnion，数值等于开关类型	Empty
ConstValue	如果生成CORBA常量，则控制常量值	Blank
GenerateDefaultSpecifier (98)	如果生成联合，则控制是否生成缺省案例	False
DefaultSpecifier (98)	如果生成联合，并且GenerateDefaultSpecifier设置为True，则设置缺省案例的标号	Blank
IDLElement (98)	控制CORBA是否从类生成	True
IDLSpecificationType (98)	控制CORBA类型 (Interface, TypeDef, Enumeration, Const, Exception, Struct, Union)	Interface

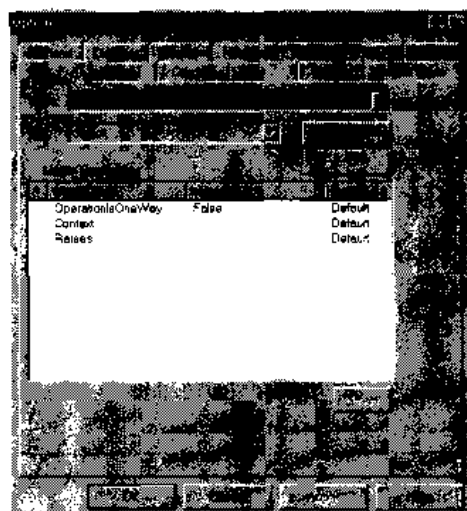
Rose 98i的类规范窗口中可以设置其中一些属性。双击CORBA类打开下列窗口：



这里可以改变文档、实现类型和数组长度选项。

属性属性

属性属性是与属性有关的CORBA属性。利用这些属性可以控制模型中每个属性生成什么。



这些属性可以在两处设置。要设置所有属性的属性，选择Tools►Options，然后选择CORBA标签，并从下拉列表框选择Attribute。要为一个属性设置属性，选择属性规范窗口的CORBA标签并在此编辑属性。

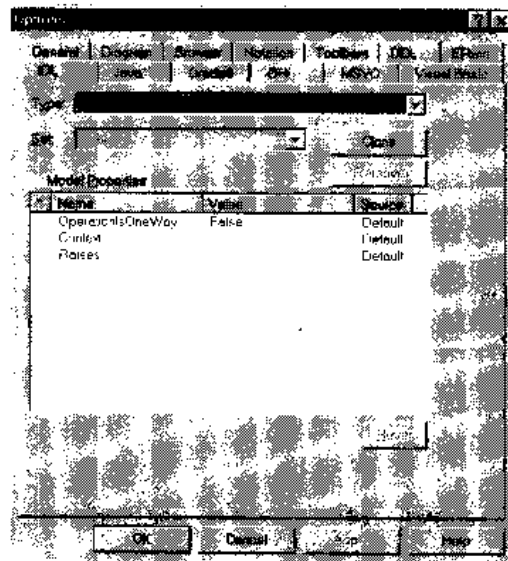
表16.3列出了CORBA属性属性及其用途和缺省值。

表16.3 CORBA/IDL属性代码生成属性

属性	用途	缺省
ArrayDimensions	BoundedRoleType关联属性设置为Array时设置异常、结构或联合时的数组维度	Blank
CaseSpecifier	设置联合的案例语句标号	Blank
IsReadOnly	控制生成的属性是否只读属性	False
Order	设置生成属性和作用的顺序	Blank
IsConst (98)	控制生成的属性是否常量	False
ConstValue (98)	IsConst为True时，设置属性值	Blank

操作属性

操作属性是与操作有关的CORBA属性。



这些属性可以在两处设置。要设置所有操作的属性，选择Tools►Options，然后选择CORBA标签，并从下拉列表框选择Operation。要对一个操作设置属性，选择操作规范窗口的CORBA标签并在此编辑属性。

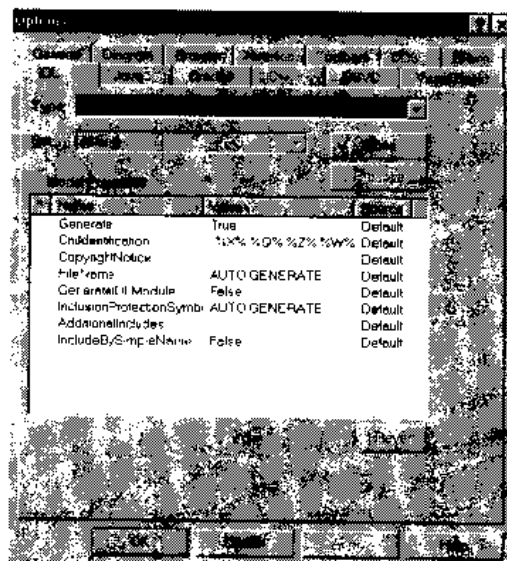
表16.4列出了CORBA操作属性及其用途和缺省值。

表16.4 CORBA/IDL操作代码生成属性

属性	用途	缺省
OperationIsOneWay	控制操作是否生成单向关键字	False
Context	包括操作的情境语句	Blank
Raises (98)	包括操作的raise语句	Blank

模块规范属性

模块规范属性是与Rose模型中的模块规范组件有关的CORBA属性。



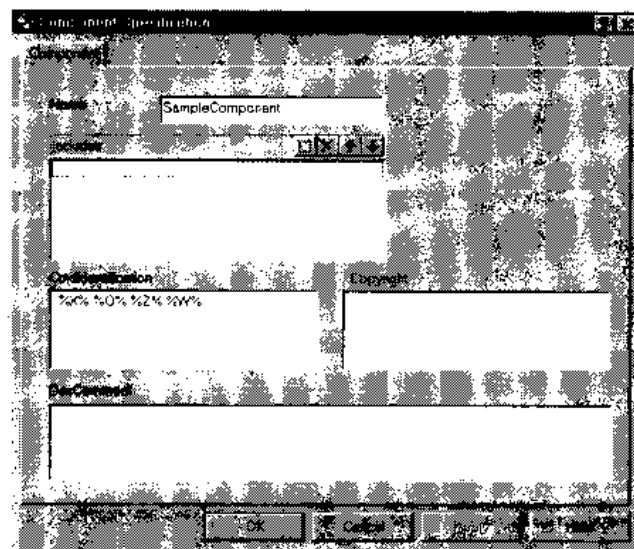
这些属性可以在两处设置。要设置所有模块规范的属性，选择Tools►Options，然后选择CORBA标签，并从下拉列表框选择Module Specification。要为一个模块规范设置属性，选择模块规范窗口的CORBA标签并在此编辑属性。

表16.5列出了CORBA模块规范属性及其用途和缺省值。

表16.5 CORBA/IDL模块规范代码生成属性

属性	用途	缺省
AdditionalIncludes	输入代码中要出现的其他#include语句	Blank
CmIdentification	输入配置管理软件可用的代码	%X% %Q% %Z% %W%
CopyrightNotice	在文件中输入版权	Blank
InclusionProtectionSymbol	设置符号，防止文件重复出现	Auto Generate
Generate (98)	控制是否生成模块规范	True
FileName (98)	设置文件名	组件名 (Auto Generate)
GenerateIDLModule (98)	确定是否生成IDL模块	False
IncludeBySimpleName (98)	控制#include语句使用路径或只用文件名	Path (False)

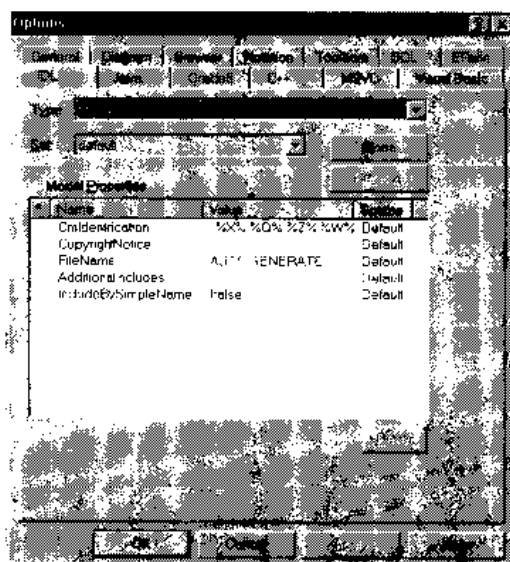
Rose 98i的组件规范窗口中可以改变一些CORBA生成属性。双击Component框图中的CORBA组件出现下列窗口：



从这个窗口中可以放上接口或改变各种代码生成选项，如文档，改变管理标识和版权。

模块体属性

模块体属性是与Rose模型中的模块体组件相关的属性。



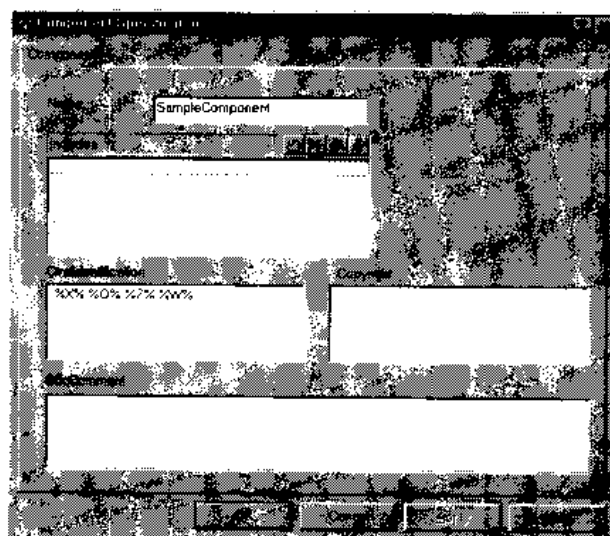
这些属性可以在两处设置。要设置所有模块体的属性，选择**Tools►Options**，然后选择**CORBA**标签，并从下拉列表框选择**Module Body**。要对一个模块体设置属性，选择模块体规范窗口的**CORBA**标签并在此编辑属性。

表16.6列出了CORBA模块体属性及其用途和缺省值。

表16.6 CORBA/IDL模块体代码生成属性

属性	用途	缺省
AdditionalIncludes	输入代码中要出现的其他#include语句	Blank
CmIdentification	输入配置管理软件可用的代码	%X% %Q% %Z% %W%
CopyrightNotice	在文件中输入版权	Blank
InclusionProtectionSymbol	设置符号，防止文件重复出现	Auto Generate
FileName (98)	设置文件名	组件名
IncludeBySimpleName (98)	控制#include语句使用路径或只用文件名	Path (False)

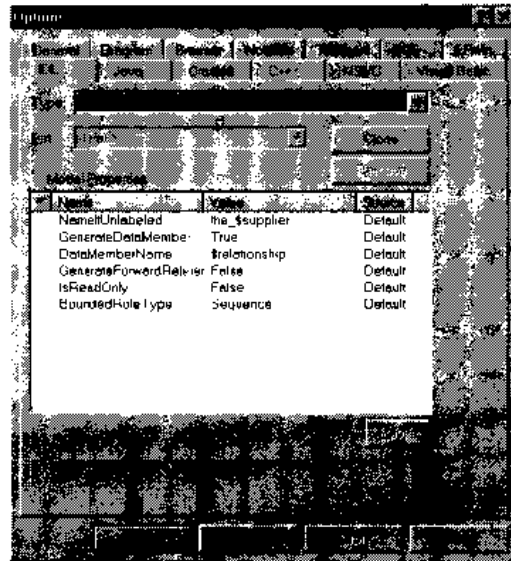
Rose 98i的组件规范窗口中可以改变一些CORBA生成属性。双击Component框图中的CORBA组件出现下列窗口：



从这个窗口中可以放上接口或改变各种代码生成选项，如文档、改变管理标识和版权

关联（作用）属性

作用属性是涉及关联的CORBA属性。利用这些属性可以控制关联代码的生成。



这些属性可以在两处设置。要设置所有关联的属性，选择Tools►Options，然后选择CORBA标签，并从下拉列表框选择Role。要对一个关联设置属性，选择关联规范窗口的CORBA标签并编辑属性。

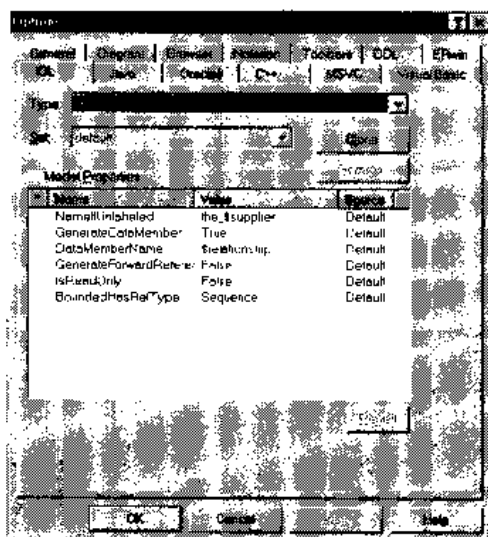
表16.7列出了CORBA关联属性及其用途和缺省值。

表16.7 CORBA/IDL关联代码生成属性

属性	用途	缺省
ArrayDimensions (98i)	BoundedRoleType关联属性设置为Array时设置异常、结构或联合使用的数组维度	Blank
CaseSpecifier (98i)	设置联合的案例语句标号	Blank
GenerateForwardReference	控制引用接口包括#include语句或前引用	#include (False)
IsReadOnly	控制生成的是否只读属性	False
Order (98i)	设置生成的属性和作用顺序	Blank
BoundedRoleType	如果关系倍数大于1，控制生成属性使用数组或顺序	Sequence
NameIfUnlabeled (98)	在没有作用名而且DataMemberName为blank或\$relationship时设置生成的属性名	the_\$supplier
GenerateDataMember (98)	控制是否对关系生成属性	True
DataMemberName (98)	设置生成的属性名	\$relationship

累积（Has关系）属性（98）

Has属性是与累积相关的CORBA属性。利用这些属性可以控制累积代码生成。这些属性只适用于Rose 98，不适用于Rose 98i。



这些属性可以在两处设置。要设置所有累积的属性，选择**Tools>Options**，然后选择**IDL**标签，并从下拉列表框选择**IDL**。要对一个累积设置属性，选择累积规范窗口的**CORBA**标签并在此编辑属性。

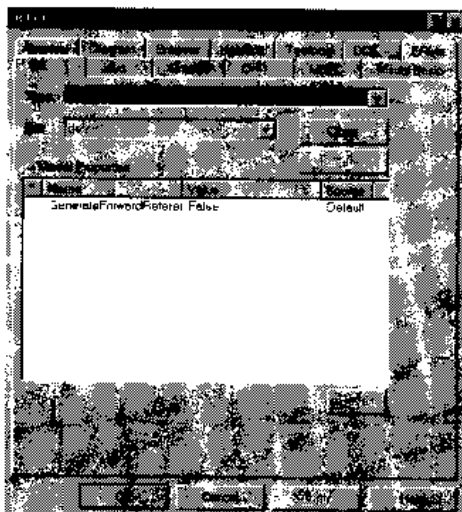
表16.8列出了CORBA累积属性及其用途和缺省值。

表16.8 CORBA/IDL累积代码生成属性

属性	用途	缺省
NameIfUnlabeled	在没有作用名而且DataMemberName为blank或\$relationship时设置生成的属性名	the_\$supplier
GenerateDataMember	控制是否对关系生成属性	True
DataMemberName	设置生成的属性名	\$relationship
GenerateForwardReference	控制引用接口包括#include语句或前引用	#include (False)
IsReadOnly	控制生成的是否只读属性	False
BoundedRoleType	如果关系倍数大于1，控制生成属性使用数组或顺序	Sequence

依赖性属性

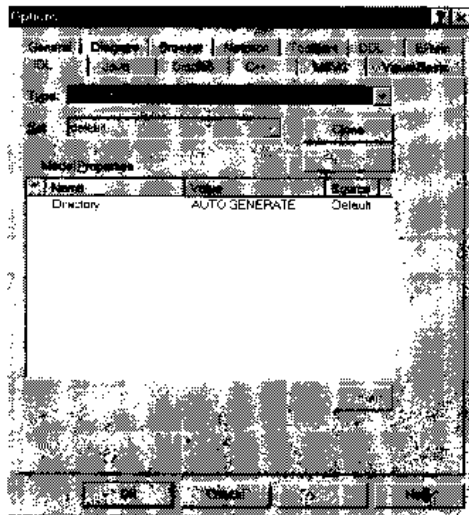
依赖性属性是控制依赖性关系如何生成的CORBA属性。



这些属性可以在两处设置。要设置所有依赖性的属性，选择Tools►Options，然后选择CORBA标签，并从下拉列表框选择Dependency。要对一个依赖性设置属性，选择依赖性规范窗口的CORBA标签并在此编辑属性。依赖性只有一个GenerateForwardReference，控制引用接口包括#include语句或前引用，缺省使用#include语句。

子系统属性 (98)

子系统属性 (98) 是适用于Rose模型中Component view包的属性。子系统属性只有一个Directory。子系统属性只适用于Rose 98。



这些属性可以在两处设置。要设置所有子系统的属性，选择Tools►Options，然后选择CORBA标签，并从下拉列表框选择Subsystem。要对一个子系统设置属性，选择子系统规范窗口的CORBA标签编辑属性。

子系统属性只有一个Directory，设置对Component view包生成的目录名。缺省时使用包名。

生成代码

下面几节介绍从各种模型元素生成的CORBA代码。Rose用各种模型元素规范窗口中输入的信息生成CORBA/IDL。

下面先介绍典型生成的代码。

类

对象模型中的一个类生成一个IDL文件。生成的文件如下：

```
Interface TheClass
{
};
```

代码中还生成大量其他信息，如配置管理语句、版权声明和包括语句。稍后将介绍完整的文件。类的所有属性、操作和关系均会体现在生成代码中。每个类生成的主要元素包括：

- Class name (类名)
- Attributes (属性)
- Operations (操作)
- Relationships (关系)
- Documentation (文档)

生成代码时，Rose用模型Component视图中建立的包结构生成相应目录。模型中的每个包生成一个目录。在每个目录中，Rose生成包中每个类的文件。如果Component视图中没有生成组件和包，则Rose用Logical视图中建立的包结构生成相应目录。

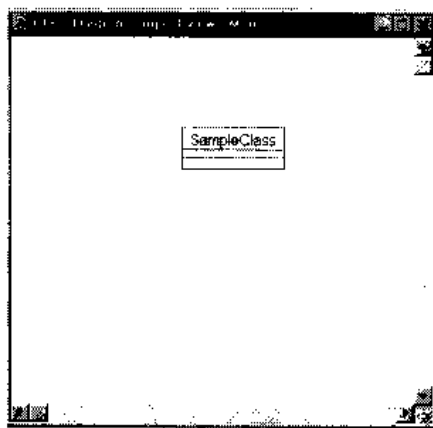
Rose模型中的大多数信息都直接用于生成代码。例如，每个类的属性、操作、关系和类名直接影响生成的代码。其他模型字段（如输入类文档）不直接影响代码，这些字段值成为生成代码的说明语句。

表16.9列出了类规范窗口中的字段及哪些字段直接影响生成的代码。

表16.9 类规范对IDL的影响

属性	影响	属性	影响
Name	模型中的名称变成类名	Type	无影响
Stereotype	无影响	Export Control	无影响
Documentation	说明语句	Cardinality	无影响
Space	无影响	Persistence	无影响
Concurrency	无影响	Abstract	无影响
Formal Arguments	无影响	Operations	在代码中生成
Attributes	在代码中生成	Relationships	在代码中生成

下面看看下图所示类生成的代码：



这个类生成下列IDL文件：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl".[this is default.]
```

```
interface SampleClass {
};

#endif
```

下面逐段地介绍这个文件。

模块段

模块段包含所生成类的基本信息，包括下列语句：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl
```

这个段包括描述所生成类和IDL文件位置的说明语句。

配置管理段

配置管理支持与配置管理软件的集成，包括下列语句：

```
/* CmIdentification
   %X% %Q% %Z% %W% */
```

配置管理段包括配置管理设置的信息。要改变配置管理设置，选择**Tools►Options**，第二行（%X% %Q% %Z% %W%）是缺省配置管理设置。在CORBA标签中，从Type下拉列表框选择**Tools►Options**打开模块规范代码生成属性。可以将CmIdentification属性值变为配置管理软件认识的字符串。

这个设置可以采用的样本值如下：

- \$date插入生成代码的日期
- \$time插入生成代码的时间
- \$module插入组件名
- \$file插入组件文件

预处理器指令段

预处理器指令段包括下列语句：

```
#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED
```

插入这些语句使文件不会重复出现。

包括段

生成代码的包括段中有组件AdditionalIncludes代码生成属性中增加的项目和组件规范窗口Includes段中增加的项目。

SampleClass的包括段如下：

```
#include "IncludedClass.idl"
```

本例中只包括一个另外的类IncludedClass。

类定义段

类定义段包含类本身的信息，包括类名、属性、操作和关系。对上述类，类定义段包括：

```
interface SampleClass {
};
```

如果用文档窗口或类规范窗口的文档区输入类文档，则这个文档成为代码中的说明语句。

不同类型CORBA/IDL生成的代码

缺省情况下，Rose对每个类生成一个接口，但还有其他生成选项，包括TypeDef、enumeration和const。生成选项可能通过CORBASpecificationType类属性设置。

下面介绍用不同CORBA/IDL规范类型选项对下图所示类生成的代码。



TypeDef生成

如果类版型设置为CORBATypeDef，则不生成类的接口，而生成TypeDef。在类规范窗口的Implementation Type属性中，输入这个TypeDef的别名定义。SampleClass类生成的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

→ typedef int SampleClass;
#endif
```

枚举生成

第二种可以生成的CORBA类型是枚举。如果选择这个选项，则Rose在生成文件中用关键字enum。上述类将版型设置为CORBAEnum时生成的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

→ enum SampleClass {
    Attribute1,
```



```

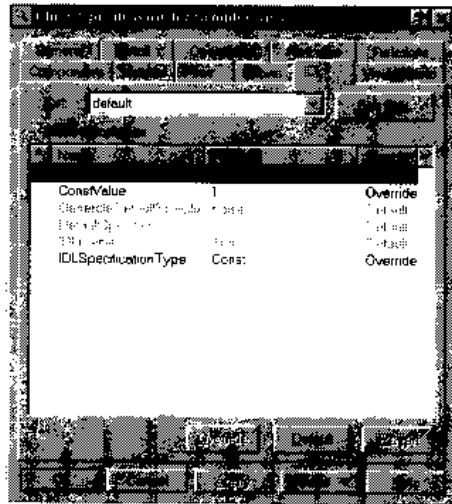
Attribute2
};

#endif

```

常量生成

第三种可以生成的CORBA类型是常量。这里Rose在生成的IDL中使用关键字const。要生成常量，将类版型设置为CORBAConstant。在Rose 98i类规范窗口中，将Implementation Type字段设置为要用的数据类型，将Constant Value字段设置为常量值。在Rose 98中，将Implementation Type类属性设置为要用的数据类型，将ConstValue属性设置为常量值。



SampleClass生成的文件如下：

```

//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

→ const int SampleClass = 4;

#endif

```

异常生成

第四种可以生成的CORBA类型是异常。如果类版型设置为CORBAException，则Rose在代码中包括关键字exception。SampleClass生成的文件如下：

```

//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification

```

```

        %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

→ exception SampleClass {
    string Attribute1;
    string Attribute2;
};

#endif

```

结构生成

另一种可以生成的CORBA类是结构。如果类版型设置为CORBA Struct, 则Rose在代码中包括关键字struct。类属性显示为生成文件中的数据类型。SampleClass生成的文件如下:

```

//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

→ struct SampleClass {
→ string Attribute1;
→ string Attribute2;
→ };

#endif

```

联合生成

最后可以将类版型设置为CORBA Union, 在CORBA中生成联合。生成的代码如下:

```

//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

union SampleClass switch(int) {
    case 1: string Attribute1;
    case 2: string Attribute2;
};

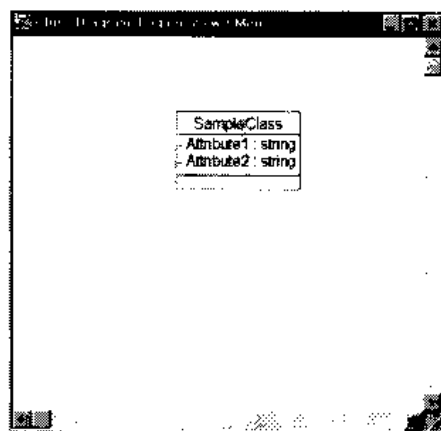
#endif

```

生成代码之前, 每个属性应有案例指定符。打开每个属性的规范窗口, 并在Case Specifier属性中输入数值。输入的值控制代码中生成的Case语句。上例中, Attribute1的案例指定符为1, Attribute2的案例指定符为2。

属性

从上例可以看出，代码中生成类的同时也生成属性。但并非所有CORBA类型都是这样。本节要介绍对每种类型生成的属性代码：interface、TypeDef、enumeration、constant、exception、structure和union。对每种类型，我们将介绍下图所示类生成的代码。



对接口生成的属性

在接口中，所有类属性都出现在生成的代码中。对每个属性，Rose包括：

- Data type（数据类型）
- Documentation（文档）

对SampleClass类生成的接口如下：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

interface SampleClass {
→       attribute string Attribute1;
→       attribute string Attribute2;
};

#endif
```

对TypeDef生成的属性

如果类版型设置为CORBATTypeDef，则生成代码中不出现属性。

对枚举生成的属性

对于枚举，Rose在生成代码中放上属性，但Rose忽略数据类型、缺省值和其他属性规

范。SampleClass类生成的枚举如下：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

enum SampleClass {
→   Attribute1,
→   Attribute2
};

#endif
```

对常量生成的属性

如果类版型设置为CORBAConstant，则生成代码中不出现属性。

对异常生成的属性

如果类版型设置为CORBAException，则类的所有属性都包括在代码中。对每个属性，代码包括：

- Data type
- Documentation

例如，下面是SampleClass的CORBASpecificationType设置为异常时生成的代码：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

exception SampleClass {
→   string Attribute1;
→   string Attribute2;
};

#endif
```

对结构生成的属性

如果类版型设置为CORBAStruct，则类的所有属性都包括在代码中。对每个属性，代码包括：

→ Documentation

例如，下面是SampleClass的CORBASpecificationType设置为结构时生成的代码：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

struct SampleClass {
→   string Attribute1;
→   string Attribute2;
   };

#endif
```

对联合生成的属性

如果类版型设置为CORBAUnion，则类的属性显示为联合中的case语句。例如，下面是SampleClass生成的代码：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

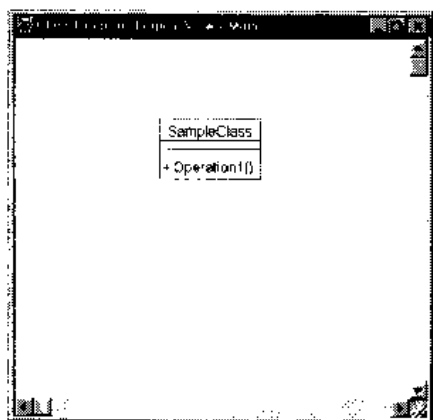
union SampleClass switch(int) {
→   case 1: string Attribute1;
→   case 2: string Attribute2;
   };

#endif
```

case语句中的值由每个属性规范窗口中的Case Specifier字段输入值设置。

操作

Rose模型中定义的操作出现在生成的IDL中。但和属性一样，操作只对某些CORBA类型适用。本节介绍下图所示类生成的操作代码：



我们将介绍CORBASpecificationType属性值改变时生成的代码。

对接口生成的操作

对于接口，类的所有操作出现在生成的代码中，包括参数、参数数据类型和返回值。对SampleClass类，生成的接口如下：

```
//Source file: c:/program files/rational/rose 98i/corba/SampleClass.idl

#ifndef __SAMPLECLASS_DEFINED
#define __SAMPLECLASS_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "IncludedClass.idl"

interface SampleClass {
    /*
       @roseuid 3738DE2901AE */
    string Operation1 ();

};

#endif
```

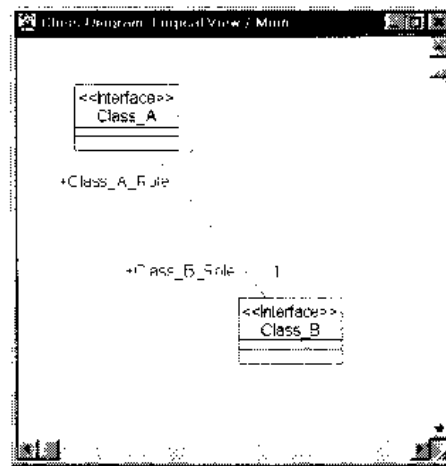
→

对其他CORBA/IDL类型生成的操作

操作只在对接口生成的IDL中显示。如果类版型设置为TypeDef、enumeration、const、exception、struct或union，则代码中不生成操作。

双向关联

要支持双向关联，Rose在代码中生成属性。关系中的每个类包含支持关联的属性。生成的属性名由关联关系中的作用名控制。要先输入作用名之后才能生成代码。



上述两个类生成的代码如下：

```
Interface Class_A
{
    attribute Class_B Class_B_Role;
};
```

和

```
Interface Class_B
{
    attribute Class_A Class_A_Role;
};
```

可以看出，Rose自动在双向关联关系两端生成属性。利用Class_B_Role属性，A类可以方便地访问B类，利用Class_A_Role 属性，B类可以方便地访问A类。

A类的完整代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ #include "Class_B.idl"

interface Class_A {
→     attribute Class_B Class_B_Role;
};

#endif
```

可以看出，A类包括B类类型的属性，B类也可包括A类类型的属性。这两个属性支持Class_A和Class_B之间的关系。

对TypeDef生成的双向关联

如果A类的版型设置为CORBATTypeDef, 则类B属性不放在A类代码中, 但A类代码中会加上#include语句如下:

```
#include "Class_B.idl"
```

对枚举生成的双向关联

如果A类的版型设置为CORBAEnum, 则代码中放上Class_B.IDL的#include语句, 但A类中不生成属性。A类代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ #include "Class_B.idl"

enum Class_A {
};

#endif
```

对常量生成的双向关联

如果A类为常量, 则它有B类的#include语句, 但没有支持关系的属性。A类代码如下:

```
//Source file: c:/program files/rational/rcse 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ #include "Class_B.idl"

const int Class_A = 4;

#endif
```

对异常生成的双向关联

如果A类为异常, 则A类代码中会生成B类类型的属性。A类代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
```



```

→ #include "Class_B.idl"

exception Class_A {
→   Class_B Class_B_Role;
};

#endif

```

对结构生成的双向关联

如果A类的版型设置为CORBAStruct，则A类代码中会生成B类类型的属性。A类代码如下：

```

//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ #include "Class_B.idl"

struct Class_A {
→   Class_B Class_B_Role;
};

#endif

```

对联合生成的双向关联

如果A类的CORBASpecificationType设置为联合，则生成的代码中有B类的#include语句，但没有在A中生成B类的属性。A类代码如下：

```

//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ #include "Class_B.idl"

union Class_A switch(int) {
    case 2: string Attribute2;
    case 1: string Attribute1;
→   case 3: Class_B Class_B_Role;
};

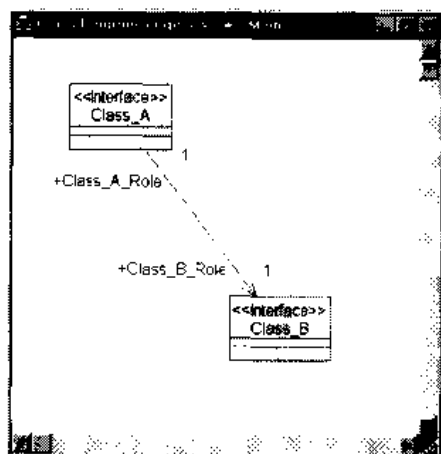
#endif

```

和A类的其他属性一样，对关联生成的属性也需要case指定符。case指定符可以用关系的CaseSpecifier作用代码生成属性设置。

单向关联

和双向关联一样，Rose生成支持单向关联的属性。但单向关联只在关系一端生成属性。



对上面的Class_A和Class_B类，生成的代码如下：

```
Interface Class_A
{
    attribute Class_B Class_B_Role;
};
```

和

```
Interface Class_B
{
};
```

可以看出，Rose只对关联一端的关系生成专用属性。具体地说，它对Client类生成属性，而不在Supplier类中生成属性。

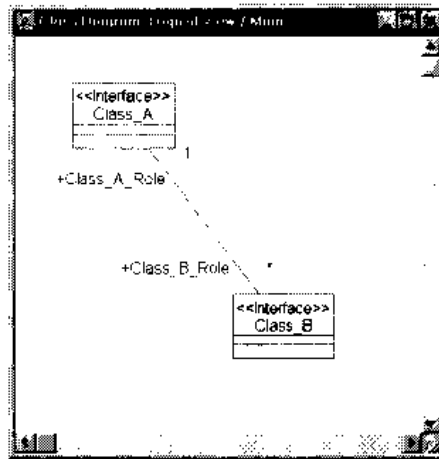
对其他CORBA类型（TypeDef、enumeration、const、exception、struct或union），Rose像双向关联中一样生成代码。单向关联的唯一不同之处是只在关系一端生成属性。

注意这里的倍增性为一对一。稍后会介绍倍增性设置对代码的影响。

倍增性为一对多的关联

在一对一关系中，Rose只是生成支持关联的相应属性。而对于一对多关系，一个类要包含其他类的设置。

下面举一个例子。



这里具有一对多关系。如上图所示，B类可以生成指向A类的属性，但A类中仅有简单指针属性是不够的，A类中的属性要用某种容器类或数组作为数据类型。IDL中可以用两种容器类：顺序和数组。缺省用顺序。

本例中Rose生成的代码如下：

```

Interface Class_A
{
    typedef sequence <Class_B> Class_B_Role_def;
    attribute Class_B_Role_def Class_B_Role;
};

```

和

```

Interface Class_B
{
    attribute Class_A Class_A_Role;
};

```

可以看出，B类只是引用A类，而A类中生成B类属性时用了容器类。

A类的完整代码如下：

```

//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

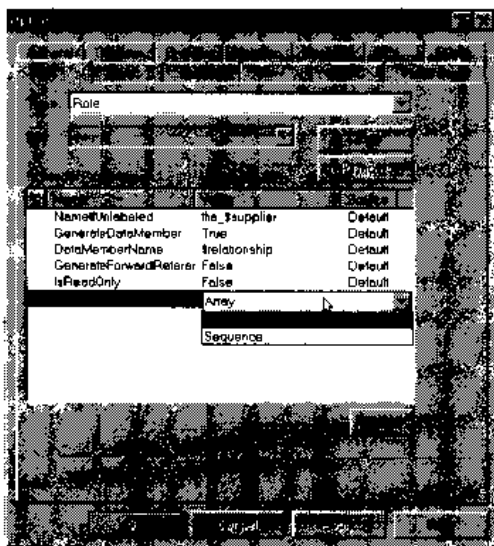
#include "Class_B.idl"

interface Class_A {
→   typedef sequence <Class_B> Class_B_Role_def;
→   attribute Class_B_Role_def Class_B_Role;
};

#endif

```

Rose缺省用顺序作为容器类。要使用数组，打开关系规范窗口，在CORBA A或CORBA B标签中，将BoundedRoleType属性变为Array。要对倍数大于1的所有关系使用数组，从菜单中选择Tools►Options，在CORBA标签的下拉列表框中选择Role，并将BoundedRoleType属性变为Array。



对TypeDef生成一对多关联

如果A类的版型设置为CORBATypeDef，则代码中加上#include语句，但A类中不生成支持与B类关联的属性。A类生成的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ #include "Class_B.idl"

typedef int Class_A;

#endif
```

对枚举生成一对多关联

和对TypeDef一样，A类为枚举时Rose不生成支持与B类关联的属性，但A类生成的代码中加上#include语句。A类生成的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
```

```
→ #include "Class_B.idl"

enum Class_A {
};

#endif
```

对常量生成一对多关联

如果A类为常量，则代码中加上#include语句，但不生成支持与B类关联的属性。A类生成的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ #include "Class_B.idl"

const int Class_A = 4;

#endif
```

对异常生成一对多关联

如果A类为异常，则A类中生成支持与B类关联的属性。在一对多关系中，Rose生成这个属性时使用容器类。缺省情况下，和接口间的一对多关系一样，Rose用顺序作为容器。要将容器类变为数组，将BoundedRoleType作用属性变为Array。

下列代码是A类的版型为CORBAException且A与B有一对多关系时A类生成的代码：

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ #include "Class_B.idl"

exception Class_A {
→ sequence <Class_B> Class_B_Role;
};

#endif
```

对结构生成一对多关联

如果A类的版型为CORBAStruct且A与B有一对多关系，则A类中生成支持与B类关联的属性。在一对多关系中，Rose生成这个属性时使用容器类。缺省情况下使用结构。要改用数

组, 改变BoundedRoleType作用属性。

A类生成的代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ #include "Class_B.idl"

struct Class_A {
→ sequence <Class_B> Class_B_Role;
};

#endif
```

对联合生成一对多关联

如果A类为联合, 则A类既生成#include语句也生成支持与B类关联的属性。A类生成的代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ #include "Class_B.idl"

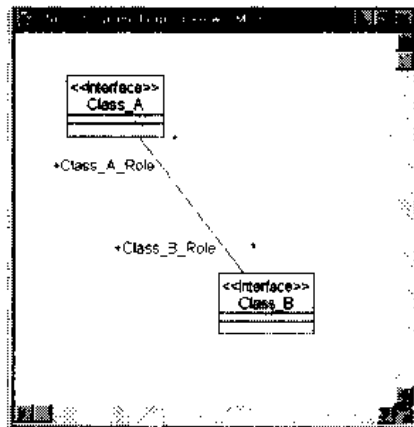
union Class_A switch(int) {
→ case 1: string Attribute1;
  case 2: string Attribute2;
  case 3: sequence <Class_B> Class_B_Role;
};

#endif
```

倍增性为多对多的关联

生成的代码与一对多关系相似, 但Rose在关系两端都生成容器类。

下面介绍下图所示关系生成的代码:



这里Rose在关系两端都生成容器类。生成的代码如下：

```
Interface Class_A
{
    typedef sequence <Class_B> Class_B_Role_def;
    attribute Class_B_Role_def Class_B_Role;
};
```

和

```
Interface Class_B
{
    typedef sequence <Class_A> Class_A_Role_def;
    attribute Class_A_Role_def Class_A_Role;
};
```

A类的完整代码与上节相同。不同之处是B类也包括一个容器类型的属性。B类生成的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Class_B.idl

#ifndef __CLASS_B_DEFINED
#define __CLASS_B_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ #include "Class_A.idl"
interface Class_B {
→     typedef sequence <Class_A> Class_A_Role_def;
→     attribute Class_A_Role_def Class_A_Role;
};

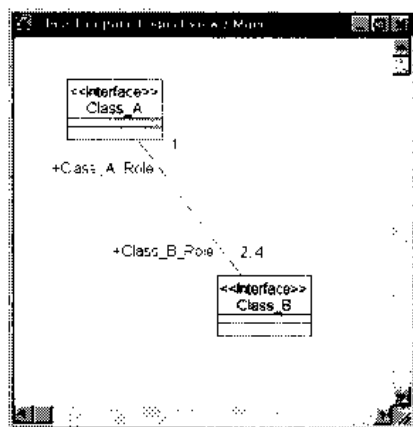
#endif
```

其他CORBA类型的多对多关联

其他CORBA类型的多对多关联生成的代码与上节相同，只是B类包含一个A类的类型属性。

约束倍增性的关联

约束倍增性的关联就是关系一端有一定取值范围的关联。例如，下例为约束倍增性的关联：



本例中，每个A类实例与二到四个B类实例相关。

约束倍增性的关联分约束关联和固定关联两种。约束关联有一定的倍数范围，如2..4。固定关联有固定倍数，如4。和一对多与多对多关系一样，Rose生成属性时使用容器类。缺省情况下，Rose用顺序，但也可以将容器变为数组。

对于上例，Rose生成如下代码：

```

Interface Class_A
{
    typedef sequence <Class_B, 4> Class_B_Role_def;
    attribute Class_B_Role_def Class_B_Role;
};
  
```

和

```

Interface Class_B
{
    attribute Class_A Class_A_Role;
};
  
```

缺省情况下，Rose用顺序作为约束关系的容器类。但也可以将容器变为数组，为此要选择关系规范窗口的CORBA A或CORBA B标签，然后将BoundedRoleType属性变为Array。要改变所有约束关系的容器类，从菜单中选择Tools►Options，在CORBA标签的下拉列表框中选择Role，并将BoundedRoleType属性变为Array。

对异常生成约束关联

如果A类为异常，则生成代码包括支持与B类关联的属性。本例中，生成代码的简化版本如下：


```
Exception Class_A
{
sequence <Class_B, 4> Class_B_Role;
};
```

和倍数大于一的其他关系一样，顺序作为约束关系的缺省容器类。要使用数组，将BoundedRoleType属性变为Array。

对结构生成约束关联

如果A类为结构，则生成代码包括支持与B类关联的属性。本例中，生成代码的简化版本如下：

```
Struct Class_A
{
sequence <Class_B, 4> Class_B_Role;
};
```

缺省时，顺序作为约束关系的容器类。要使用数组，将BoundedRoleType属性变为Array。

对联合生成约束关联

如果A类为联合，则生成代码包括支持与B类关联的属性。缺省情况下，这个属性使用的容器为顺序。这里顺序的长度为4。

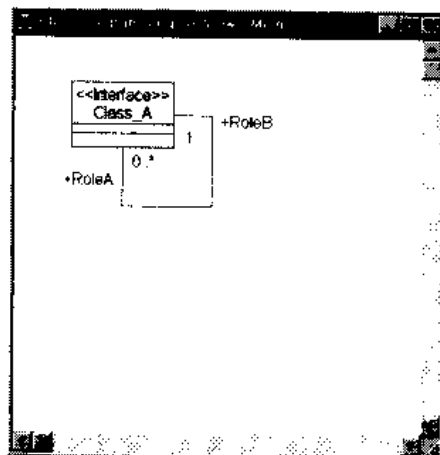
```
union Class_A switch(int) {
case 3: sequence <Class_B, 4> Class_B_Role;
};
```

对其他CORBA/IDL类型生成约束关联

如果CORBA/IDL类型不是接口、异常、结构或联合，则不生成支持关系的属性，但Class_A和Class_B中会放上#include语句。

反身关联

反身关联与两个类之间的关联相似。对于下图：



生成的代码如下:

```
Interface Class_A
{
    typedef sequence <Class_A> RoleA_def;
    attribute Class_A RoleB;
    attribute RoleA_def RoleA;
};
```

前两行支持关系的0..*端, 有一个支持倍增性的容器类。第三行支持关系中倍数为1的一端。

A类的完整代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ interface Class_A {
    typedef sequence <Class_A> RoleA_def;
→     attribute Class_A RoleB;
→     attribute RoleA_def RoleA;
};

#endif
```

对结构生成反身关联

如果A类为结构, 则在其中生成一个支持反身关系的属性。如果关系与上面的反身关联相同, 则生成的代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
→ exception Class_A {
    Class_A RoleB;
};

#endif
```

对联合生成反身关联

如果A类为联合, 则在其中生成一个支持反身关系的属性。A类的代码如下:

```
//Source file: c:/program files/rational/rose 98:/corba/Class_A.idl

#ifndef __CLASS_A_DEFINED
#define __CLASS_A_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

→ struct Class_A {
    Class_A RoleB;
};

#endif
```

对其他CORBA/IDL类型生成反身关联

由于TypeDef、enumeration或constant不生成属性，因此这些类型的反身关系不在代码中体现。

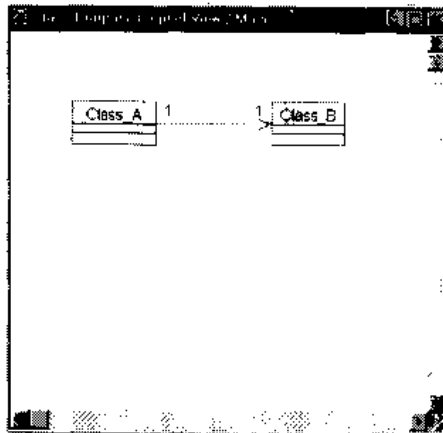
累积

生成CORBA/IDL时，关联和累积一样处理。前面介绍的所有关联问题（倍增性、单向与双向、反身）也适用于累积，任何CORBA类型（interface、TypeDef、enumeration、constant、exception、structure或union）都是如此。

关于如何生成单向累积、不同倍增性的累积和反身累积的信息，见关联的相应章节。

依赖性关系

依赖性关系不生成属性。如果Class_A和Class_B之间有依赖性关系，则Class_A或Class_B之间不生成属性。



生成的代码如下：

```
Interface Class_A
{
};
```

和

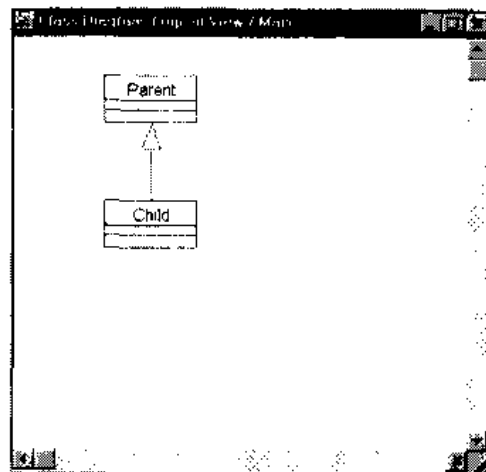
```
Interface Class_B
{
};
```

Rose只在A类中放上B类的引用，即Class_B.IDL的包括语句，B类中根本不引用A类。

由于依赖性关系不与成属性，因此不对任何CORBA类型（interface、TypeDef、enumeration、constant、exception、structure或union）生成属性。

一般化关系

UML一般化关系在IDL中成为继承关系。在Rose模型中，继承关系显示如下：



对于这类关系，Rose生成如下代码：

```
Interface Parent
{
};
```

和

```
Interface Child : Parent
{
};
```

下面要介绍生成的实际代码。在父类代码中，根本不提及子类。这样就使父类保持一般化，可以派生多个子类而不影响父类的代码。

在子类中，生成支持从父类继承的代码。子类的代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */
```

```
→      #include "Parent.idl"
→
      interface Child : Parent {
      };

      #endif
```

对TypeDef生成一般化关系

如果子类为TypeDef, 则生成代码中出现父类#include语句, 但代码中不显示继承关系。子类的IDL如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Parent.idl"

typedef Child;

#endif
```

对枚举生成一般化关系

枚举也是这样。尽管出现父类#include语句, 但代码中不显示继承关系。子类的IDL如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Parent.idl"

enum Child {
};

#endif
```

对常量生成一般化关系

和TypeDef与枚举中一样, 常量的代码中不直接实现一般化关系。但会出现引用父类的#include语句。生成的代码如下:

```
//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED
```

```

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Parent.idl"

const Child = ;

#endif

```

对异常生成一般化关系

异常不支持继承，因此和其他情况一样，会出现引用父类的#include语句，但代码本身不体现一般化关系。这时的IDL如下：

```

//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Parent.idl"

exception Child {
};

#endif

```

对结构生成一般化关系

和其他CORBA类型一样，结构不支持继承，因此出现父类#include语句，但代码中不显示继承关系。子类的IDL如下：

```

//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Parent.idl"

struct Child {
};

#endif

```

对联合生成一般化关系

联合的一般化也和除接口以外的其他CORBA类型相似。由于联合不支持一般化，因此不在生成代码中出现一般化，但会出现引用父类的#include语句。代码如下：

```
//Source file: c:/program files/rational/rose 98i/corba/Child.idl

#ifndef __CHILD_DEFINED
#define __CHILD_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Parent.idl"

union Child switch() {
};

#endif
```

生成CORBA/IDL举例

前面几章用各种语言生成了ATM例子的代码。本例要生成ATM例子中Account类的IDL。

本书选配光盘中包括了ATM例子中Account类的IDL。

练习

第3章至第10章构造了订单输入系统的模型。这里要对订单输入系统的一些类生成IDL代码，包括Order类、OrderItem类、TransactionMgr类和OrderMgr类。要生成IDL，需要完成下列步骤。本书选配光盘中包括本练习的代码。

练习步骤

设置CORBA版型

1. 打开Add Order Class框图。
2. 打开Order类的类标准规范窗口。
3. 将Order类的版型设置为Interface。
4. 对OrderItem、TransactionMgr和OrderMgr类重复第2和第3步。

设置作用名

1. 对框图上的每个关系，打开关系规范窗口。在关系两端生成作用名。

生成组件

1. 第9章用C++语言生成组件。现在要设置CORBA组件。第一步要删除不再需要的旧组件。
2. 打开Main Component框图。
3. 选择框图中所有组件并按Ctrl+D将其删除。

4. 用Component工具栏按钮，生成四个新组件：Order、OrderItem、TransactionMgr和OrderMgr。
5. 打开Order组件的规范窗口。
6. 将组件语言设置为CORBA。
7. 在浏览器中，将Order类从Logical视图的Order组件中拖走。
8. 重复第5至第7步将OrderItem、TransactionMgr和OrderMgr类映射到相应组件，并将所有组件语言设置为CORBA。

设置生成代码的目录

1. 选择Tools►CORBA►Project Specification。
2. 选择Directories字段的New工具栏按钮。设置生成代码的目录。

设置CORBA数据类型

1. 打开Order类OrderNumber属性的规范窗口。
2. 选择规范窗口Type字段旁边的“...”按钮。
3. 选择层次树中CORBA Types旁边的加号。
4. 选择类清单中的Long。
5. 对Order、OrderItem、OrderMgr和TransactionMgr类中的所有属性重复第1到第4步，将每个类属性映射到相应CORBA类型。
6. 打开Order类Setinfo操作的规范窗口。
7. 选择规范窗口Return字段旁边的“...”按钮。
8. 选择层次树中CORBA Types旁边的加号。
9. 选择类清单中的boolean。
10. 双击变元表中的Integer OrderNum变元。
11. 在这个规范窗口中，选择层次树中CORBA Types旁边的加号并选择类清单中的Long。
12. 对Order、OrderItem、OrderMgr和TransactionMgr类中的所有操作和变元重复第6到第11步。

生成CORBA/IDL

1. 打开Add New Order Class框图。
2. 选择Order、OrderItem、OrderMgr和TransactionMgr类。
3. 选择Order、OrderItem、TransactionMgr和OrderMgr。提示时将组件映射到项目规范窗口中加进的目录。本书选配光盘中包括了生成的代码。

小结

本章介绍了各个Rose模型元素如何用CORBA/IDL实现。利用类、包、属性、操作、关联、累加和其他Rose模型元素的代码生成属性，可以很好地控制生成的代码。

生成代码的步骤如下：

1. 选择CORBA/IDL代码生成属性。
2. 选择Class或Component框图中要生成的类或组件。
3. 选择Tools►CORBA►Generate CORBA。

完成这些步骤后，即可得到从模型生成的CORBA/IDL文件。

第17章 DDL代码生成

- 设置类的持续性
- 设置DDL代码生成属性
- 从Rose模型生成DDL

如果模型中的类想生成数据库表格，Rose可以对表格生成Structured Query Language (SQL) 语句。类中的每个属性成为数据库表格中的字段。通过代码生成属性可以设置每个字段的数据类型、长度和其他细节。本章介绍如何对Rational Rose模型生成Database Definition Language (数据库定义语言) DDL代码。

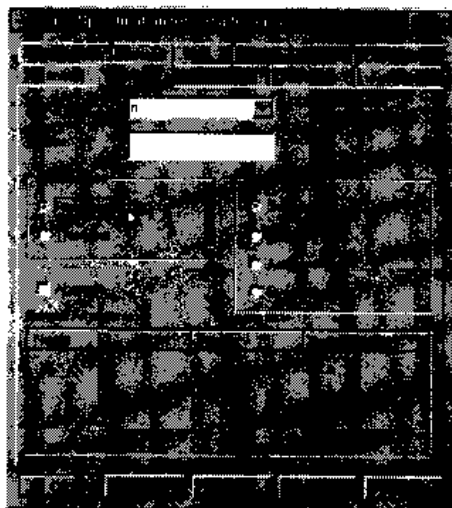
从模型生成DDL的步骤如下：

1. 设置类的持续性。
2. 设置DDL代码生成属性。
3. 选择Class框图中要生成的类。
4. 选择Tools►DDL►Generate Code。

缺省情况下，Rose不对所有类生成DDL。许多类（如控制类和边界类）变成数据库表格是没有意义的，只有实体类可以变成数据库表格。

第一步要标出哪些类要生成DDL。Rose中所有标为持续的类都可以生成DDL。要将类标为持续，打开类规范窗口，并在Detail标签中选择Persistent单选钮。

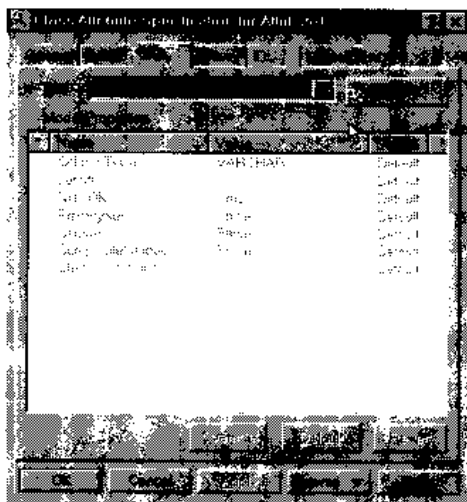
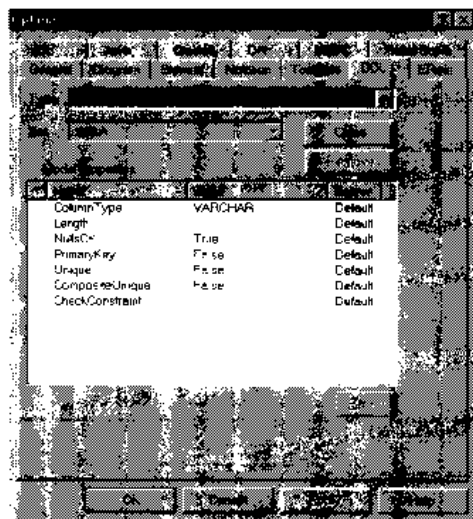
将相应类标为持续后，可以设置其DDL生成属性并由此生成DDL。本章介绍属性和项目的各种DDL代码生成属性。由于操作和其他模型元素不在DDL中体现，因此这些元素没有DDL生成属性。



DDL代码生成属性

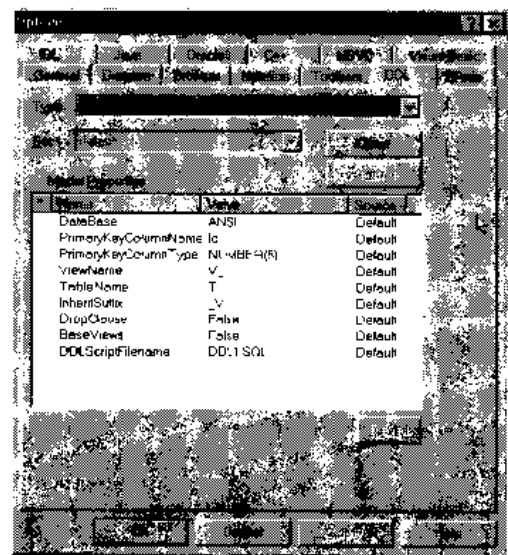
和其他语言一样，从Rose模型生成的DDL也是由一组属性控制的。在Rose中，属性和项目都有DDL属性。要浏览和设置这些属性，选择Tools►Options，然后选择DDL标签。

用这个窗口进行任何改变都将设置所有属性和整个项目的缺省值。要设置单个属性的DDL生成属性，打开属性规范窗口并选择DDL标签，这里进行的任何改变只影响一个属性。



项目属性

项目属性是影响整个项目的DDL代码生成属性。



利用这些属性，可以设置使用的缺省数据库，缺省主关键字名，视图前缀，表格前缀和其他整个数据库的设置。表17.1列出各个项目属性及其用途和缺省值。

表17.1 项目DDL生成属性

属性	用途	缺省
Directory (98i)	设置生成DDL的缺省目录	当前目录
DataBase	设置使用的缺省数据库（ANSI, Oracle, SQL Server, Sybase, Watcom）	ANSI
PrimaryKeyColumnName	如果不设置类中的某个属性为主关键字，则Rose会生成主关键字。主关键字名为类名加上这个属性的值	Id
PrimaryKeyColumnType	如果不设置类中的某个属性为主关键字，则Rose会生成主关键字。这个属性控制所生成主关键字的数据类型	NUMBER (5)
ViewName	设置所有生成视图的前缀	V_
TableName	设置所有生成表格的前缀	T_
InheritSuffix	设置对继承映射生成的视图后缀	_V
DropClause	控制是否先运行DROP TABLE语句再运行CREATE TABLE语句	False
BaseViews	控制是否对表格生成视图	False
DDLScriptFilename	设置生成的DDL脚本文件名	DDL1.SQL

属性属性

第二组DDL属性是属性的属性。这些属性可以控制每个字段的缺省数据类型、每个字段的缺省长度和是否允许NULL值。

这些属性可以在两处设置。要设置所有属性的属性，选择Tools►Options，然后选择DDL标签，并从下拉列表框选择Attribute。要为一个属性设置属性，选择属性规范窗口的DDL标签，并在此编辑属性。

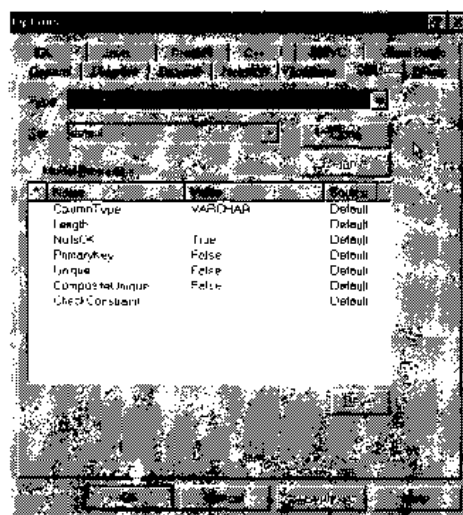


表17.2列出了DDL属性的属性及其用途和缺省值。

表17.2 DDL属性代码生成属性

属性	用途	缺省
ColumnType	设置每个字段的缺省数据类型	VARCHAR
Length	设置每个字段的缺省长度	Blank
NullsOK	控制字段是否允许NULL值	True
PrimaryKey	控制字段是否主关键字或主关键字的一部分	False
Unique	控制每个记录是否需要唯一值	False
CompositeUnique	控制字段是否表格接合关键字的一部分	False
CheckConstraint	设置字段的限制检查	Blank

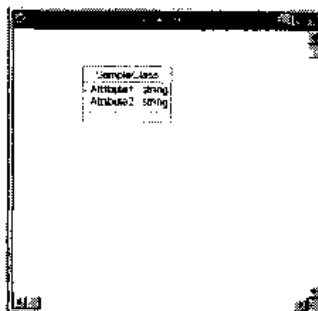
生成代码

在最简单的情况下，Rose对标为持续的每个类生成一个数据库表格。但如果持续类之间有关系，则生成的DDL会更加复杂。Rose自动生成外部关键字和生成支持关系的视图。下面几节介绍对简单类和类间不同关系生成的DDL。

首先介绍对单个类生成的DDL。

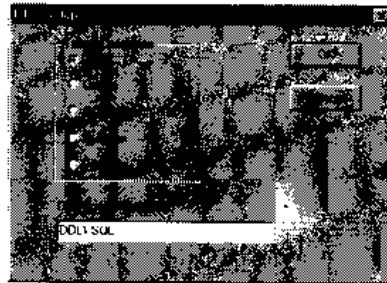
类和属性

如果类标为持续，则可以对其生成SQL脚本，以便生成数据库表格。类中每个属性变成表格中的字段。如果没有显式标为主关键字的属性，则Rose会生成主关键字，用表名和缺省前缀Id。下面看看下图所示类的DDL：



```
CREATE TABLE T_SampleClass(  
    Attribute1 VARCHAR,  
    Attribute2 VARCHAR,  
    SampleClassId NUMBER(5),  
    PRIMARY KEY(SampleClassId))
```

这里没有改变DDL生成属性，而是采用缺省值。上述SQL是对该类生成的ANSI SQL。生成DDL时，Rose询问是否使用ANSI、Oracle、SQL Server、Sybase或Watcom



上例中如果使用Oracle，则生成的DDL如下：

```
CREATE TABLE T_SampleClass(  
    Attribute1 VARCHAR(),  
    Attribute2 VARCHAR(),  
    SampleClassId NUMBER(5),  
    PRIMARY KEY(SampleClassId));
```

使用SQL Server时，生成的DDL如下：

```
CREATE TABLE T_SampleClass(  
    Attribute1 VARCHAR(),  
    Attribute2 VARCHAR(),  
    SampleClassId NUMBER(5),  
    PRIMARY KEY(SampleClassId))  
go
```

使用Sybase时，生成的DDL如下：

```
CREATE TABLE T_SampleClass(  
    Attribute1 VARCHAR(),  
    Attribute2 VARCHAR(),  
    SampleClassId NUMBER(5),  
    PRIMARY KEY(SampleClassId))  
go
```

最后，使用Watcom时，生成的DDL如下：

```
CREATE TABLE T_SampleClass(  
    Attribute1 VARCHAR(),  
    Attribute2 VARCHAR(),  
    SampleClassId NUMBER(5),  
    PRIMARY KEY(SampleClassId));
```

可以看出,不同数据库生成的DDL差不多,但Rose用所选数据库要求的语法。从Rose生成SQL语句后,就可以对数据库执行这个SQL语句。

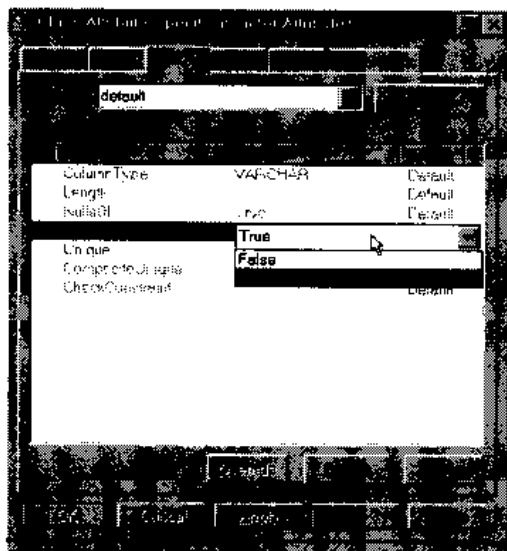
改变表格名

缺省情况下, Rose用T_加上类名作为生成的表格名。要改变缺省表格前缀, 选择Tools►Options, 然后选择DDL标签, 并从下拉列表框选择Project, 然后改变TableName属性值。

设置主关键字

如果没有显示标为主关键字的属性, 则Rose会生成主关键字, 用表名和缺省前缀Id。上例中, SampleClass类的主关键字变成SampleClassId。要改变使用的后缀, 选择Tools►Options, 然后选择DDL标签, 并从下拉列表框选择Project, 然后改变PrimaryKeyColumnName属性值。

如果要将类的一个属性变为主关键字, 打开该属性的规范窗口, 选择DDL标签, 并将PrimaryKey属性变为True。



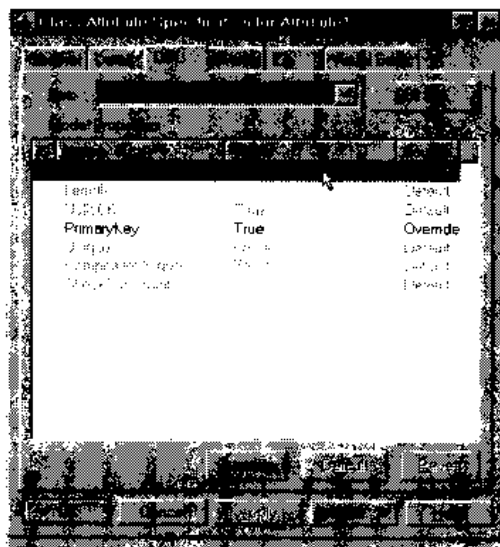
要生成复合主关键字, 将两个或几个属性的PrimaryKey属性变为True。例如, 如果SampleClass类中的两个属性都作为主关键字, 则生成复合关键字。

```
CREATE TABLE T_SampleClass(
    Attribute1 VARCHAR(),
    Attribute2 VARCHAR(),
    PRIMARY KEY(Attribute1,Attribute2))
```

改变字段数据类型或长度

每个字段的缺省数据类型为VARCHAR, 但可以改变缺省数据类型或改变一个字段的的数据类型。要改变所有字段的缺省数据类型, 选择Tools►Options, 然后选择DDL标签, 并将ColumnType属性的属性变为缺省数据类型。

要改变一个字段的数据类型，打开属性规范窗口，并在DDL标签中改变ColumnType属性值。



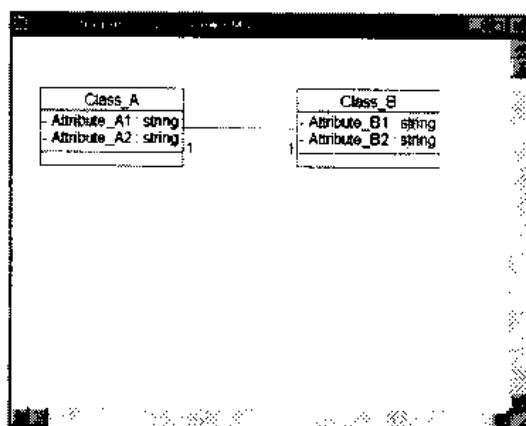
控制字段是否允许NULL值

要控制字段是否允许NULL值，可以改变属性的NullsOK属性。缺省情况下，所有字段都允许NULL值。如果将一个属性的NullsOK属性变为False，则Rose在生成的代码中加上NOT NULL从句。例如，将Attribute1属性的NullsOK属性变为False如下：

```
CREATE TABLE T_SampleClass(
    Attribute1 VARCHAR() NOT NULL,
    Attribute2 VARCHAR(),
    PRIMARY KEY(Attribute1))
```

双向关联

如果类之间为双向关联，则Rose通过增加外部关键字在生成的DDL中包括这个关系。两个表都有外部关键字，引用另一表的主关键字。例如，看看下图所示关系生成的DDL：



上述两个类生成的DDL如下：


```
CREATE TABLE T_Class_A(  
    Attribute_A1 VARCHAR(),  
    Attribute_A2 VARCHAR(),  
→    Class_BId NUMBER(5) REFERENCES T_Class_B(Class_BId),  
    Class_AId NUMBER(5),  
    PRIMARY KEY(Class_AId))
```

```
CREATE TABLE T_Class_B(  
    Attribute_B1 VARCHAR(),  
    Attribute_B2 VARCHAR(),  
→    Class_AId NUMBER(5) REFERENCES T_Class_A(Class_AId),  
    Class_BId NUMBER(5),  
    PRIMARY KEY(Class_BId))
```

可以看出, T_Class_A和T_Class_B都有外部关键字, 指向对方。

改变外部关键字名

缺省情况下, Rose用主关键字名作为其他表中的外部关键字名。例如, T_Class_A的关键字为Class_AId, 因此, T_Class_B的外部关键字名也为Class_AId。

要改变外部关键字名, 在两类的关系中加上关联名。Rose即用这个关联名作为外部关键字名。本例中, 如果Class_A和Class_B之间的关联为AssociationName, 则生成的DDL如下:

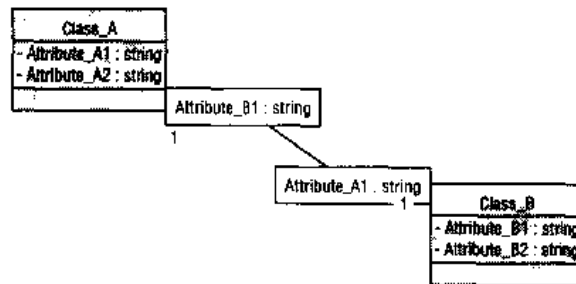
```
CREATE TABLE T_Class_A(  
    Attribute_A1 VARCHAR(),  
    Attribute_A2 VARCHAR(),  
→    AssociationName NUMBER(5) REFERENCES T_Class_B(Class_BId),  
    Class_AId NUMBER(5),  
    PRIMARY KEY(Class_AId))
```

```
CREATE TABLE T_Class_B(  
    Attribute_B1 VARCHAR(),  
    Attribute_B2 VARCHAR(),  
→    AssociationName NUMBER(5) REFERENCES T_Class_A(Class_AId),  
    Class_BId NUMBER(5),  
    PRIMARY KEY(Class_BId))
```

可以看出, Rose用这个关联名作为外部关键字名。

改变索引

要生成表格索引, 需要有关系限定符。例如, 假设要在A类中生成Attribute_A1的索引, B类中生成Attribute_B1的索引, 可以加进限定符如下:



在生成的DDL中，就有了每个表格的索引。T_Class_A对Attribute_A1字段索引而T_Class_B对Attribute_B1字段索引。生成的DDL如下：

```

CREATE TABLE T_Class_A(
    Attribute_A1 VARCHAR(),
    Attribute_A2 VARCHAR(),
    Class_Bid NUMBER(5) REFERENCES T_Class_B(Class_BId),
    Class_AId NUMBER(5),
    PRIMARY KEY(Class_AId))

→ CREATE INDEX Class_A_1 ON T_Class_A(Attribute_A1)

CREATE TABLE T_Class_B(
    Attribute_B1 VARCHAR(),
    Attribute_B2 VARCHAR(),
    Class_AId NUMBER(5) REFERENCES T_Class_A(Class_AId),
    Class_Bid NUMBER(5),
    PRIMARY KEY(Class_BId))

→ CREATE INDEX Class_B_1 ON T_Class_B(Attribute_B1)
  
```

关联倍增性

生成DDL时，Rose只检查关系的移动性，而不考虑关联倍增性。关联倍增性不影响生成的DDL。因此一对一、一对多和多对多关系的生成方法是相同的。

单向关联

单向关联生成的DDL与双向关联相似。但对于单向关联，外部关键字只放在关系一端。下面看看单向关联生成的DDL。

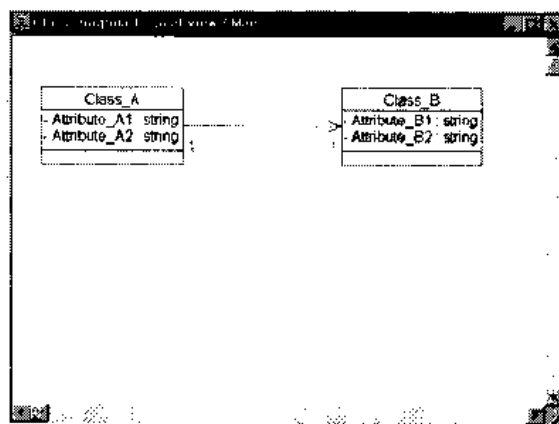
本例中，对T_Class_A表生成外部关键字，而不对T_Class_B表生成外部关键字。生成的DDL如下：

```

CREATE TABLE T_Class_A(
    Attribute_A1 VARCHAR(),
    Attribute_A2 VARCHAR(),
    Class_Bid NUMBER(5) REFERENCES T_Class_B(Class_BId),
    Class_AId NUMBER(5),
    PRIMARY KEY(Class_AId))

→
  
```

```
CREATE TABLE T_Class_B(
    Attribute_B1 VARCHAR(),
    Attribute_B2 VARCHAR(),
    Class_Bid NUMBER(5),
    PRIMARY KEY(Class_Bid))
```

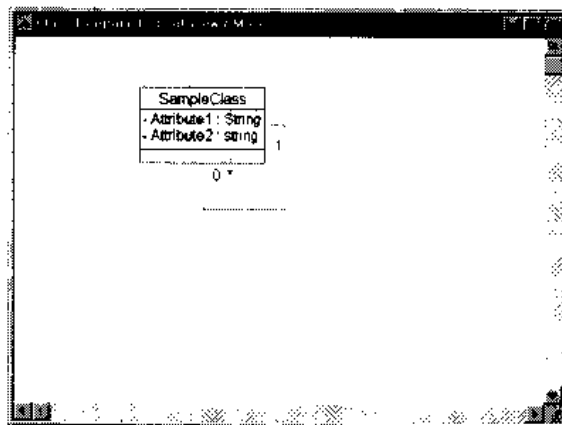


关联倍增性

和双向关联一样，Rose对单向关系生成DDL时忽略关联倍增性，一对一、一对多和多对多关系的生成方法是相同的。

反身关联

对于反身关联，Rose在类中放上引用主关键字的外部关键字。和普通关联一样，Rose生成DDL时忽略关联倍增性。



上类生成的DDL如下：

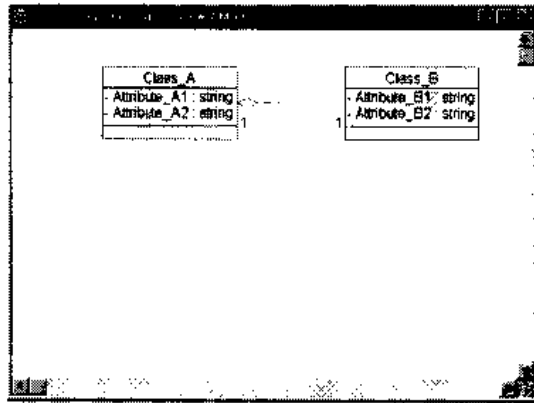
```
CREATE TABLE T_SampleClass(
    Attribute1 VARCHAR() NOT NULL,
    Attribute2 VARCHAR(),
    Attribute1 VARCHAR() NOT NULL,
    FOREIGN KEY (Attribute1) REFERENCES T_SampleClass,
    PRIMARY KEY(Attribute1))
```

可以看出，T_SampleClass类中放上引用主关键字Attribute1的外部关键字。

累积

累积生成的DDL与所用数据库有关。如果用ANSI、SQL Server或Sybase生成DDL，则累积的生成与关联完全一样。

如果使用Oracle或Watcom，则Rose在生成的DDL中加上ON DELETE CASCADE从句。



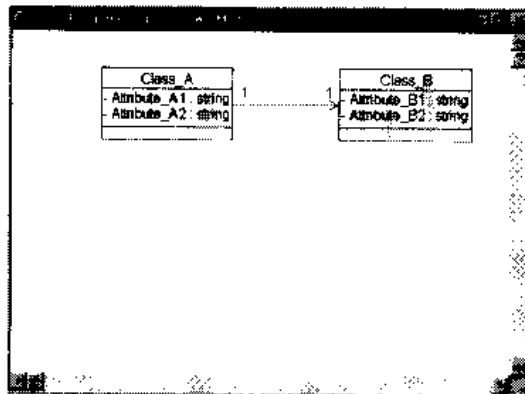
上例中生成的Oracle DDL如下:

```
CREATE TABLE T_Class_A(
    Attribute_A1 VARCHAR(),
    Attribute_A2 VARCHAR(),
    Class_Bid NUMBER(5) REFERENCES T_Class_B(Class_Bid) ON DELETE CASCADE,
    Class_Aid NUMBER(5),
    PRIMARY KEY(Class_Aid));

CREATE TABLE T_Class_B(
    Attribute_B1 VARCHAR(),
    Attribute_B2 VARCHAR(),
    Class_Aid NUMBER(5) REFERENCES T_Class_A(Class_Aid),
    Class_Bid NUMBER(5),
    PRIMARY KEY(Class_Bid));
```

依赖性关系

如果两个类之间有依赖性关系，则Rose生成的DDL与两个类之间没有任何关系时一样。每个表格单独生成，不加进支持依赖性关系的外部关键字。



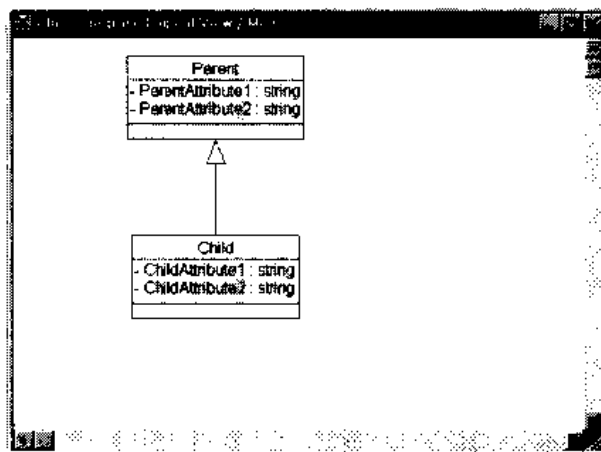
上述关系生成的DDL如下：

```
CREATE TABLE T_Class_A(
    Attribute_A1 VARCHAR(),
    Attribute_A2 VARCHAR(),
    Class_AId NUMBER(5),
    PRIMARY KEY(Class_AId))

CREATE TABLE T_Class_B(
    Attribute_B1 VARCHAR(),
    Attribute_B2 VARCHAR(),
    Class_BId NUMBER(5),
    PRIMARY KEY(Class_BId))
```

一般化关系

由于关系型数据库不直接支持继承，因此Rose生成视图以支持模型中的一般化关系。在父子关系中，Rose首先生成父和子的表格，并生成支持继承关系的视图。让我们看看下图所示关系生成的DDL：



本例中，生成两个表格，一个用于父类，一个用于子类。此外，还生成支持继承关系的视图。

```
CREATE TABLE T_Parent(
    ParentAttribute1 VARCHAR(),
    ParentAttribute2 VARCHAR(),
    PRIMARY KEY(ParentAttribute1))

CREATE TABLE T_Child(
    ChildAttribute1 VARCHAR(),
    ChildAttribute2 VARCHAR(),
    ParentAttribute1 VARCHAR(),
    PRIMARY KEY(ParentAttribute1),
    FOREIGN KEY (ParentAttribute1) REFERENCES T_Parent
)
```

```
CREATE VIEW Child V(  
    ChildAttribute1,  
    ChildAttribute2,  
    ParentAttribute1,  
    ParentAttribute2)  
AS SELECT  
    T_Child.ChildAttribute1,  
    T_Child.ChildAttribute2,  
    T_Parent.ParentAttribute1,  
    T_Parent.ParentAttribute2  
FROM T_Child,T_Parent
```

对继承关系生成DDL时，不能对父类使用Rose生成的主关键字，而要设置父类某个属性的PrimaryKey DDL属性为True。

生成DDL举例

前面几章用各种语言生成了ATM例子的代码。本例要生成ATM例子中Account类的DDL。

下面是对Account类生成的ANSI SQL。可以看出，类中每个属性成为表中的字段。Rose还对表格生成主关键字AccountId。

```
CREATE TABLE T_Account(  
    Account Number VARCHAR(),  
    PIN VARCHAR(),  
    Balance VARCHAR(),  
    AccountId NUMBER(5),  
    PRIMARY KEY(AccountId))
```

练习

本练习对订单输入系统的Order和OrderItem类生成DDL。要生成DDL，需要完成下列步骤。

练习步骤

将类标为持续

1. 打开类框图。
2. 选择Order类。
3. 打开类规范窗口并选择Detail标签。
4. 复选Persistent框并关闭类规范窗口。
5. 对OrderItem类重复第2到第4步。

设置DDL属性

1. 打开Order类OrderNumber属性的规范窗口。

2. 选择DDL标签。
3. 将ColumnType属性变为NUMBER。
4. 将Length属性变为3。
5. 将PrimaryKey属性变为True并关闭规范窗口。
6. 重复第1到第5步将OrderItem类的ItemID属性变为主关键字，类型为NUMBER，长度为5。

生成IDL

1. 选择Class框图的Order和OrderItem类。
2. 选择Tools►DDL►Generate Code。选择数据库类型ANSI。生成的代码如下：

```
CREATE TABLE T_OrderItem(  
    ItemID NUMBER(5),  
    ItemDescription VARCHAR(),  
    PRIMARY KEY(ItemID))  
  
CREATE TABLE T_Order(  
    OrderNumber NUMBER(3),  
    CustomerName VARCHAR(),  
    OrderDate VARCHAR(),  
    OrderFillDate VARCHAR(),  
    ItemID NUMBER(5),  
    FOREIGN KEY (ItemID) REFERENCES T_OrderItem,  
    PRIMARY KEY(OrderNumber))
```

小结

本章介绍如何从Rose模型生成DDL。在Rose中可以将类标为持续以生成DDL。生成DDL的步骤如下：

1. 设置类的持续性。
2. 设置DDL代码生成属性。
3. 选择Class框图中要生成的类。
4. 选择Tools►DDL►Generate Code。

第18章 Oracle8结构生成

- 在Rose中用Data Type Creation Wizard生成表格、视图、存储过程、触发器和其他Oracle8元素
- 对项目、类、属性和作用设置Oracle8属性
- 从Rose模型生成Oracle8结构

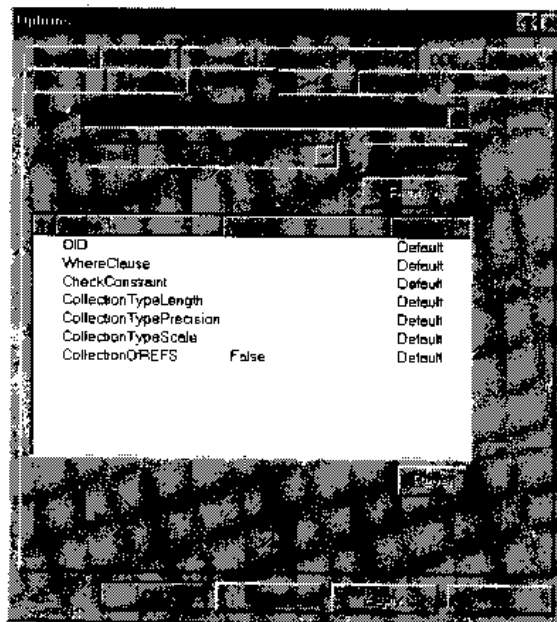
Rose通过集成Oracle8，可以生成表格、视图、存储过程、触发器和其他Oracle8元素。Oracle8集成是Rational Rose和Oracle之间向导驱动的易用链接。

Rose企业版提供Oracle8，具有正向和逆向转出工程代码功能，可以生成新数据库或检查现有数据库。后者在重建工程中特别有用，可以检查现有代码和数据库。

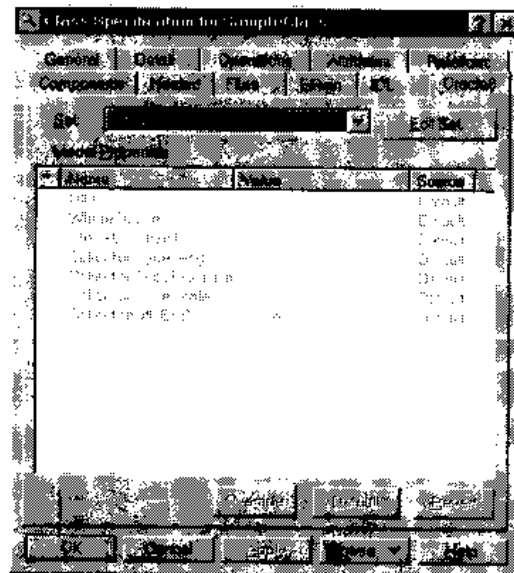
本章介绍Rose正向工程特性，包括Oracle8的代码生成属性和Oracle8中如何实现各种Rose元素。

Oracle8代码生成属性

和其他语言一样，从Rose模型生成Oracle8数据库时生成的DDL也是用一组属性控制。Rose中有针对项目、类、属性、操作、模块规范和作用的Oracle8属性。要浏览和设置这些属性，选择Tools►Options，然后选择Oracle8标签。

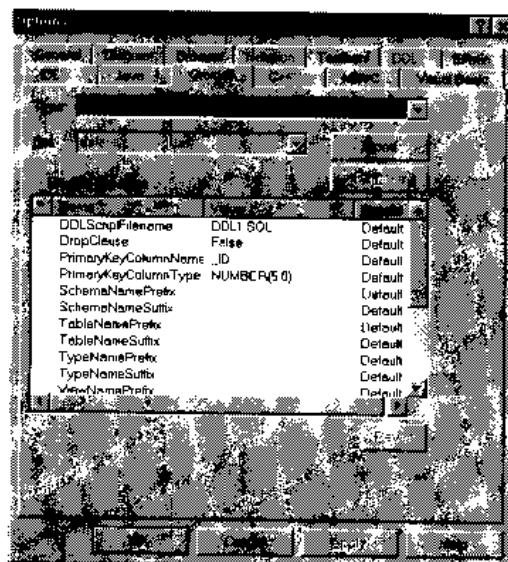


这个窗口中所作的改变会设置所有对象的缺省值。要设置单个属性、类或作用的Oracle8代码生成属性，打开其规范窗口并选择Oracle8标签。



项目属性

项目属性是适用于整个项目而不是某个模型元素（如类和关系）的代码生成属性。



本节的选项包括生成代码时使用的缺省文件名和主关键字列、结构、表等的后缀。表18.1列出各个项目属性及其用途和缺省值。

表18.1 项目代码生成属性

属性	用途	缺省值
DDLScriptFileName	生成代码时使用的缺省文件名	DDL1.SQL
DropClause	控制是否对每个Oracle实体运行DROP语句	False
PrimaryKeyColumnName	设置生成主关键字时使用的后缀	_ID
PrimaryKeyColumnType	设置主关键字的缺省数据类型	NUMBER(5,0)
SchemaNamePrefix	设置生成结构的组件名开头所要的前缀	Blank
SchemaNameSuffix	设置生成结构的组件名后面所要的后缀	Blank

(续表)

属性	用途	缺省值
TypeNamePrefix	设置生成对象的类名开头所要的前缀	Blank
TypeNameSuffix	设置生成对象的类名后面所要的后缀	Blank
TableNamePrefix	设置生成表格的类名开头所要的前缀	Blank
TableNameSuffix	设置生成表格的类名后面所要的后缀	Blank
ViewNamePrefix	设置生成视图的类名开头所要的前缀	Blank
ViewNameSuffix	设置生成视图的类名后面所要的后缀	Blank
VARRAYNamePrefix	设置生成VARRAY的类名开头所要的前缀	Blank
VARRAYNameSuffix	设置生成VARRAY的类名后面所要的后缀	Blank
NestedTableNamePrefix	设置生成嵌套表的类名开头所要的前缀	Blank
NestedTableNameSuffix	设置生成嵌套表的类名后面所要的后缀	Blank
ObjectTableNamePrefix	设置生成Object表的类名开头所要的前缀	Blank
ObjectTableNameSuffix	设置生成Object表的类名后面所要的后缀	Blank

类属性

类属性是适用于类的Oracle8代码生成属性。这些属性可以设置集合类型的where从句。

这些属性可以在两处设置。要设置所有类的属性，选择Tools►Options，然后选择Oracle8标签，并从下拉列表框选择Class。要为一个类设置属性，选择类规范窗口的Oracle8标签编辑属性。

表18.2列出了Oracle8类属性及其用途和缺省值。

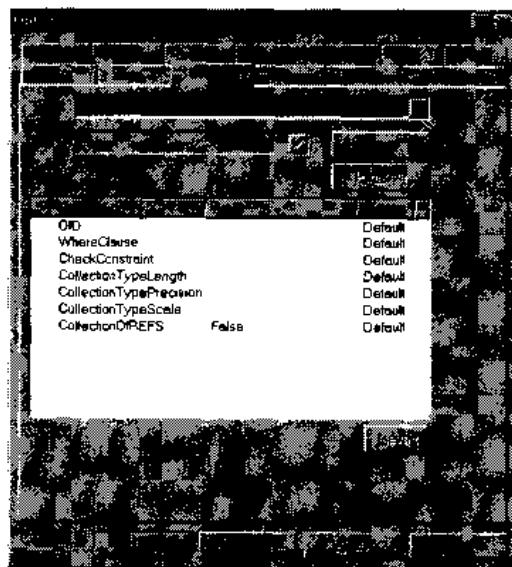
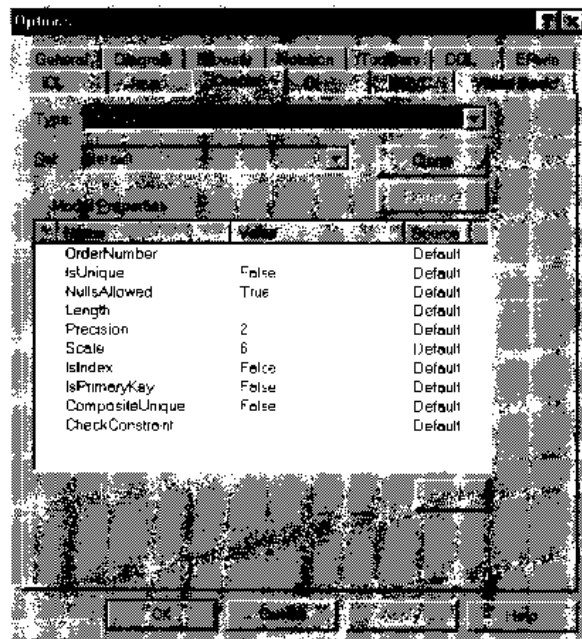


表18.2 Oracle8类代码生成属性

属性	用途	缺省值
OID	设置对象ID	Blank
WhereClause	设置过滤条件	Blank
CheckConstraint	设置检查限制	Blank
CollectionTypeLength	设置标量VARRAY中的数值长度	Blank
CollectionTypePrecision	设置标量VARRAY中的数值精度	Blank
CollectionTypeScale	设置标量VARRAY大小	Blank
CollectionOfREFS	表示类型是否其他对象的REF集合	False

属性属性

属性属性是与属性相关的Oracle8属性。例如，用属性属性可以确定属性是否唯一，设置属性为主关键字或设置限制。



这些属性可以在两处设置。要设置所有属性的属性，选择Tools►Options，然后选择Oracle8标签，并从下拉列表框选择Attribute。要为一个属性设置属性，选择属性规范窗口的Oracle8标签编辑属性。

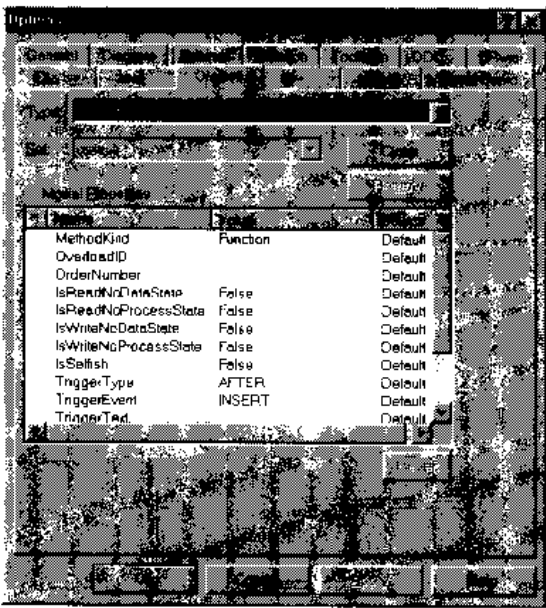
表18.3列出了Oracle8属性属性及其用途和缺省值。

表18.3 Oracle8属性代码生成属性

属性	用途	缺省值
OrderNumber	设置属性的列顺序	Blank
IsUnique	确定属性是否唯一	False
NullsAllowed	确定该字段是否允许NULL值	True
Length	设置字段长度	Blank
Precision	设置NUMBER字段的精度	2
Scale	NUMBER类型长度	6
IsIndex	确定属性是否索引的一部分	False
IsPrimaryKey	确定属性是否主关键字	False
CompositeUnique	确定属性是否复合关键字的一部分	False
CheckConstraint	设置检查限制	Blank

操作属性

操作属性是与特定操作相关的Oracle8代码生成属性。这些属性可以确定生成的方法类型。



这些属性可以在两处设置。要设置所有操作的属性，选择Tools►Options，然后选择Oracle8标签，并从下拉列表框选择Operation。要为一个操作设置属性，选择操作规范窗口的Oracle8标签编辑属性。

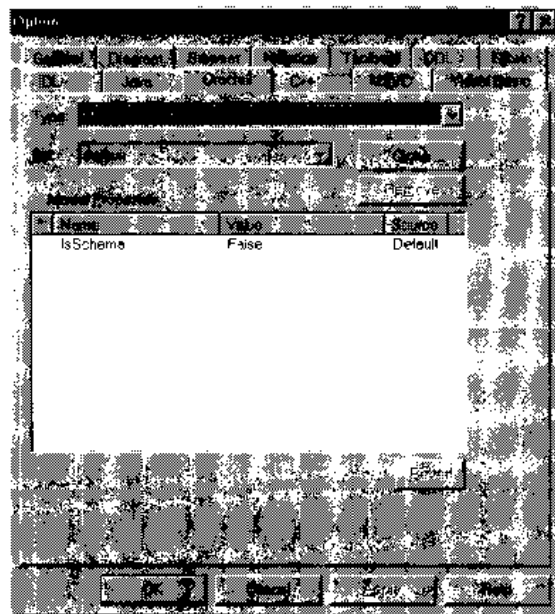
表18.4 列出了Oracle8操作属性及其用途和缺省值。

表18.4 Oracle8操作代码生成属性

属性	用途	缺省值
MethodKind	确定操作为Map Method、Order Method、Function、Procedure、Operator、Constructor、Destructor、Trigger或Calculated Column	Function
OrderNumber	如果列为计算视图列，指定视图中的顺序号	Blank
TriggerType	设置生成的触发器类型（BEFORE或AFTER）	AFTER
TriggerEvent	设置启动触发器的事件（INSERT，DELETE，UPDATE）	INSERT
TriggerText	设置触发器的过程文本	Blank
TriggerForEach	控制触发器对每行或每个语句触发	ROW

模块规范属性

模块规范属性是与Rose中生成的结构相关的属性。这些属性可以确定是否生成结构。

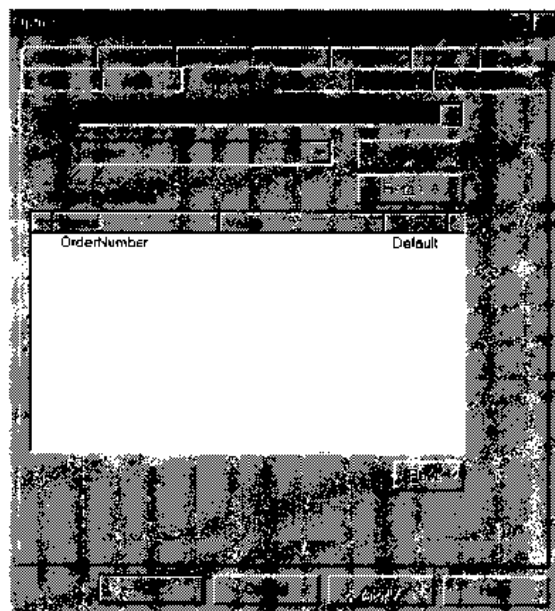


这些属性可以在两处设置。要设置所有模块规范的属性，选择Tools►Options，然后选择Oracle8标签，并从下拉列表框选择Module Specification。要为一个模块规范设置属性，选择模块规范窗口的Oracle8标签并编辑属性。

Oracle8中唯一的模块规范属性是IsSchema，控制组件是否表示Oracle8结构。这个属性的缺省值为False。

作用属性

作用属性是影响关系的Oracle8代码生成属性。有一个作用属性，可以设置关系所生成属性的顺序号。



这些属性可以在两处设置。要设置所有作用的属性，选择Tools►Options，然后选择Oracle8标签，并从下拉列表框选择Role。要对一个作用设置属性，选择作用规范窗口的Oracle8标签并编辑属性。

Oracle8中只有一个作用属性，即OrderNumber，可以设置关系所生成属性的顺序号，这个字段缺省为blank。

生成Oracle8对象

生成Oracle8对象不同于生成其他语言的对象。在Oracle8中，要生成结构、关系型表、Object类型和视图。在Rose中，可以用数据类型生成向导（Data Type Creation Wizard）生成这些Oracle8结构。虽然Rose中可以直接生成对象，但相当困难，而利用向导，则自动生成所有的Oracle8代码生成属性。如果在Rose中直接生成对象，则要手工设置这些属性。下面几节介绍如何用Data Type Creation Wizard生成Oracle8结构。

Object Type

Object Type是包含信息（属性）和行为（方法）的Oracle8结构。信息和作用于信息的行为可以组成类，而这些项目则可以组成Object Type。

在Rose中，Object Type显示为类，版型为ObjectType。类的每个属性实现为所生成Object Type的属性，而操作实现为方法。下面几节将详细介绍属性和操作。

要生成Object Type，选择Tools►Oracle8►Data Type Creation Wizard。在出现的窗口中，选择ObjectType，如图18.1。

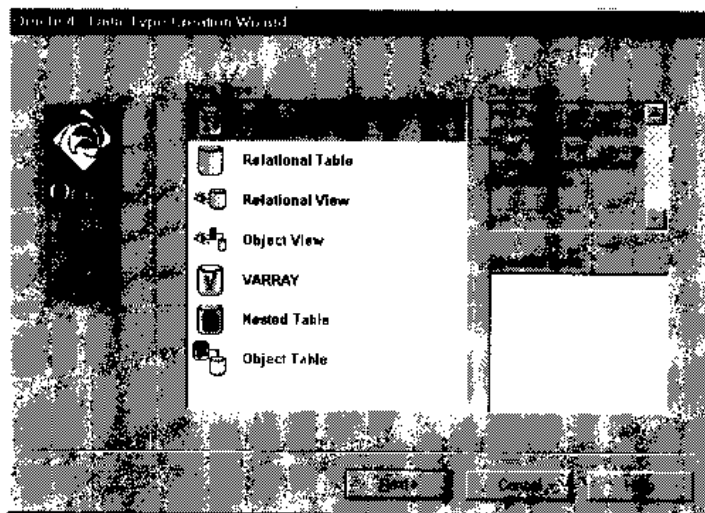


图18.1 生成Object Type

选择Object Type后，单击Next按钮。在Data Type Description屏幕中，输入新的Object类型名，Object类型的结构（组件），包含该类型的包和显示该类型的Class框图。如果喜欢，也可以在窗口底部输入文档。Data Type Description窗口如图18.2。

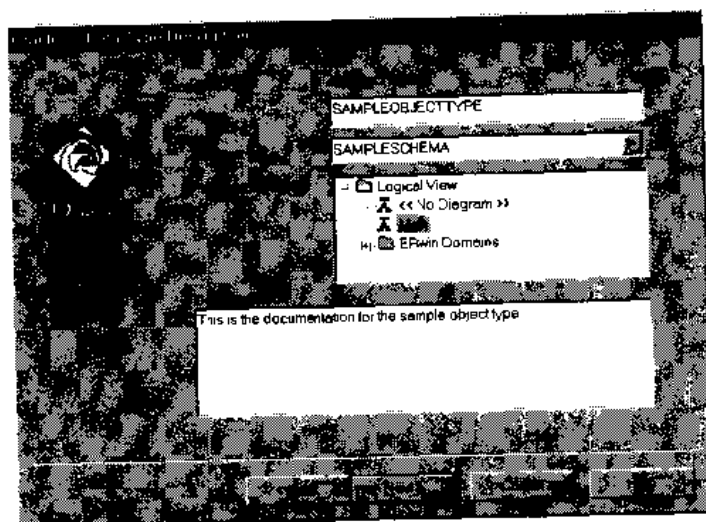


图18.2 Data Type Description屏幕

单击Next按钮打开Define Attribute窗口，如图18.3。可以选择现有属性，也可以从Object类型生成新属性。

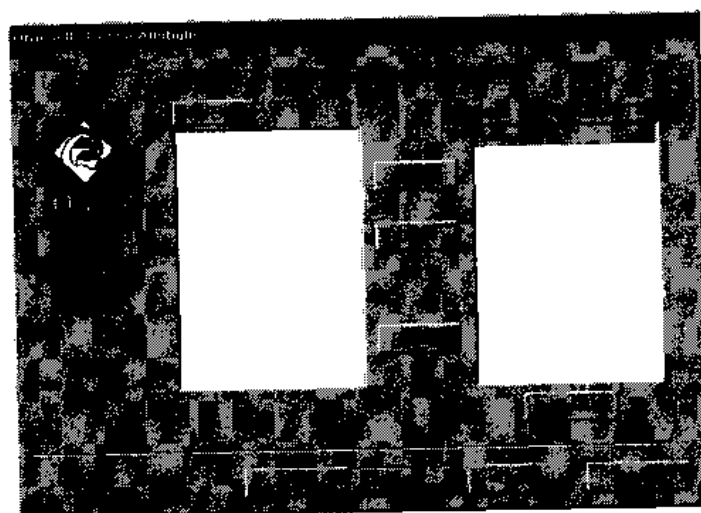


图18.3 Define Attribute窗口

要选择现有属性，选择Map From按钮。如果启动向导之前选择了一些Object类型或表格，则它们已经出现在Map From框中。从框中选择项目或加进新属性。要加进新属性，选择Create按钮。出现Create/Edit Attribute窗口，如图18.4。

在这个窗口中，输入新属性的名称、类型、精度、长度和其他细节。完成之后，单击Add Attribute按钮。加进所有属性后，单击Close按钮。

下一屏幕为Define Operations屏幕，如图18.5。

要生成操作，首先输入操作名并选择操作类型。在Return Type下拉列表框中，选择从操作返回的信息类型。

如果操作包含参数，输入参数名、方向和类型。然后单击Add按钮加进参数。

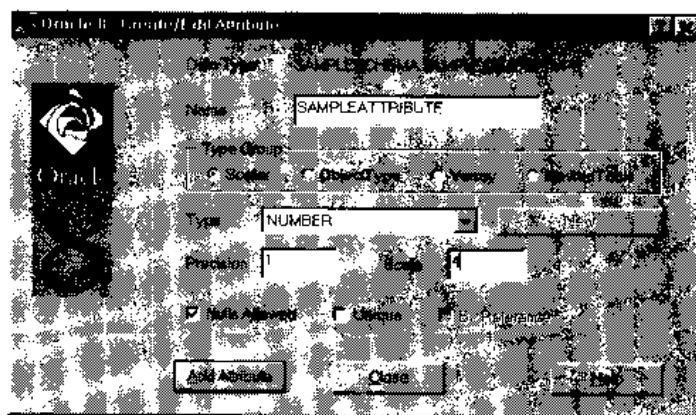


图18.4 Create/Edit Attribute窗口

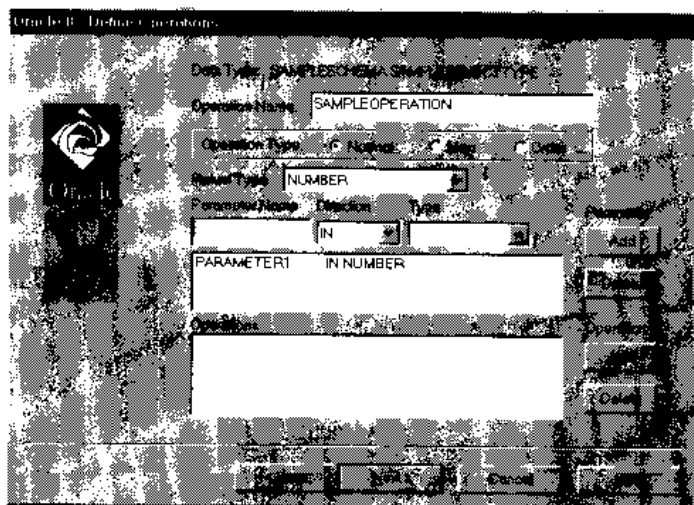


图18.5 Define Operations屏幕

加进所有参数后，单击Add按钮加进操作。用同样的方法加进其他操作。完成之后单击Next按钮。

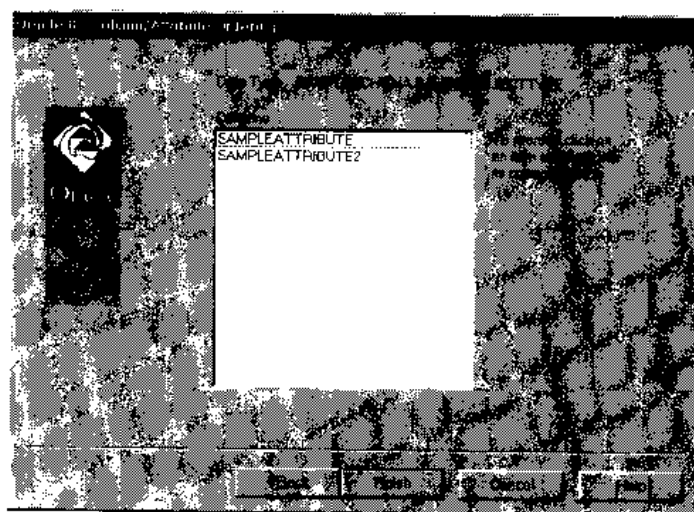


图18.6 Ordering窗口

然后可以改变属性顺序。在图18.6所示的Ordering窗口中，列出了这个Object类型设置的所有属性。要改变属性顺序，将属性拖动到新位置。顺序正确之后，单击Finish按钮。

注意Ordering窗口是Oracle8的另一向导。生成数据类型后，可以选择Tools>Oracle8>Ordering Wizard回到Ordering窗口。

Rose生成版型为ObjectType的类，如图18.7。类中包括用向导定义的所有属性和操作。

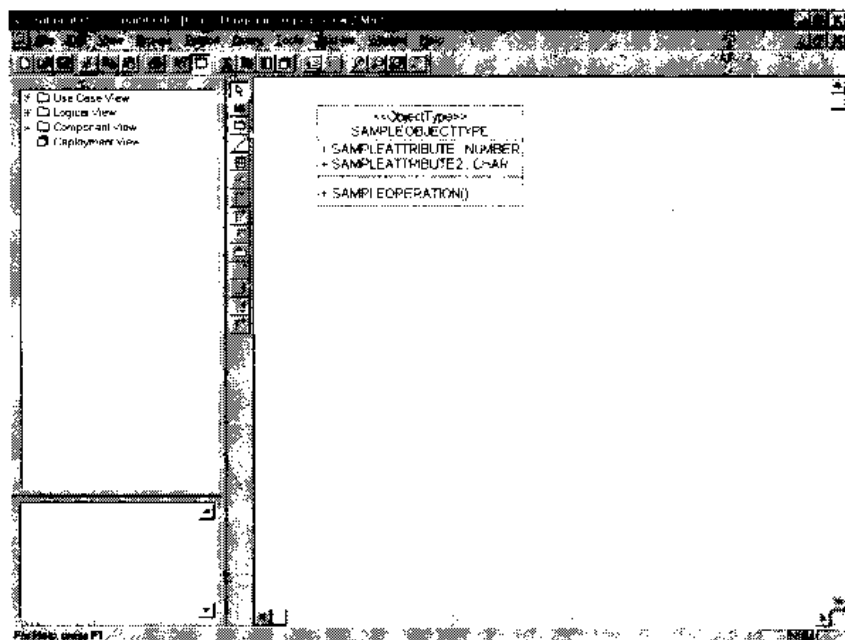


图18.7 SampleObjectType类

关系型表格

关系型表格与Object类型相似，但不包含方法。此外，关系型表格可以有索引和关键字。

Rose中关系型表格显示为版型为Relational Table的类。类的每个属性实现为所生成关系型表格中的属性。

要生成关系型表格，选择Tools>Oracle8>Data Type Creation Wizard。在出现的窗口中，选择Relational Table，如图18.8。

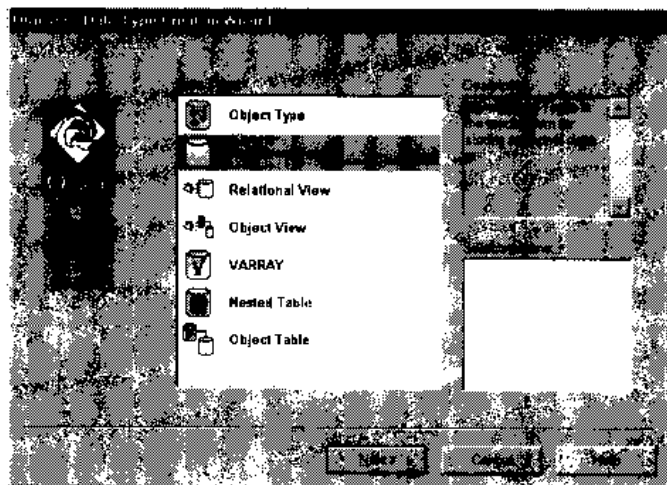


图18.8 生成关系型表格

选择Relational Table之后，单击Next按钮。在Data Type Description屏幕中，输入新关系型表格名，关系型表格结构（组件），表格所在包和显示表格的Class框图。如果喜欢，也可以在窗口底部输入文档。Data Type Description屏幕如图18.9。

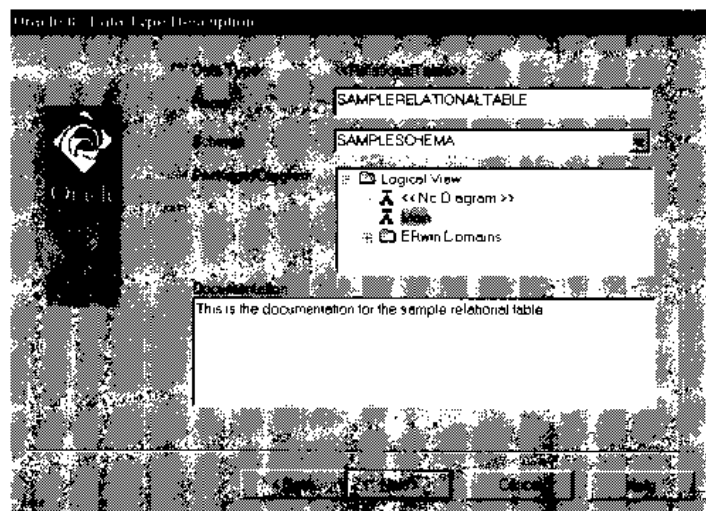


图18.9 Data Type Description屏幕

在图18.10所示的Define Column窗口中，可以选择现有属性或对关系型表格生成新列（属性）。

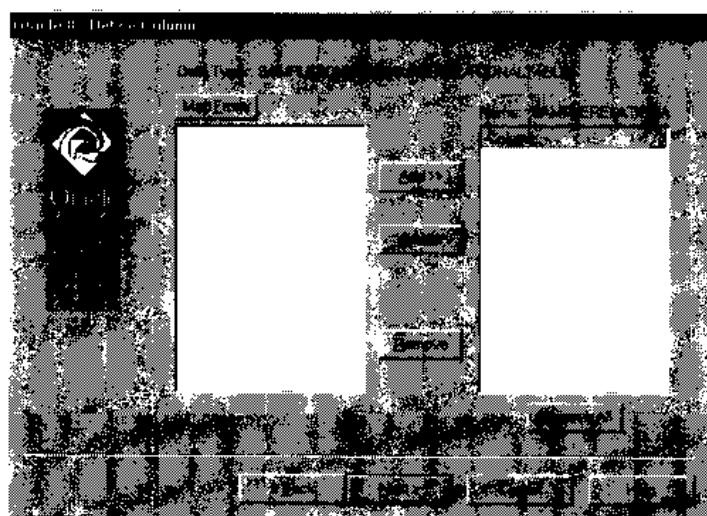


图18.10 Define Column窗口

要选择现有属性，选择Map From按钮。如果启动向导之前选择了一些关系型表格，则它们已经出现在Map From框中。从框中选择项目或加进新列。要加进新列，选择Create按钮。出现Create/Edit Column窗口，如图18.11。

在这个窗口中，输入新属性的名称、类型、精度、长度和其他细节。完成之后，单击Add Column按钮。加进所有列后，单击Close按钮。

下一屏幕为Indices屏幕，如图18.12。

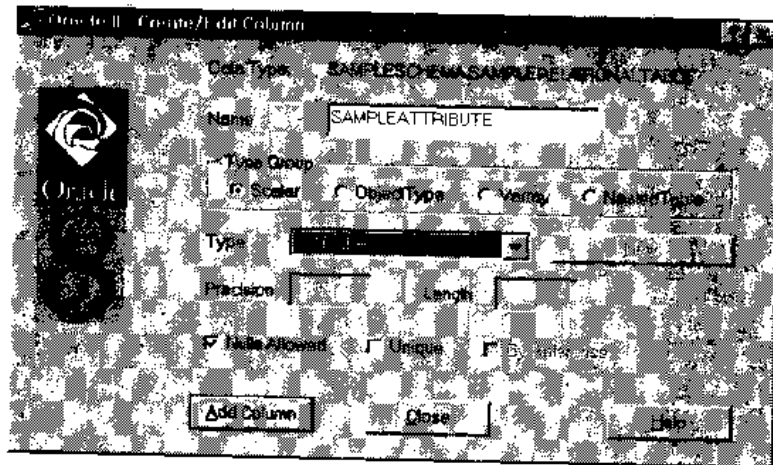


图18.11 Create/Edit Column窗口

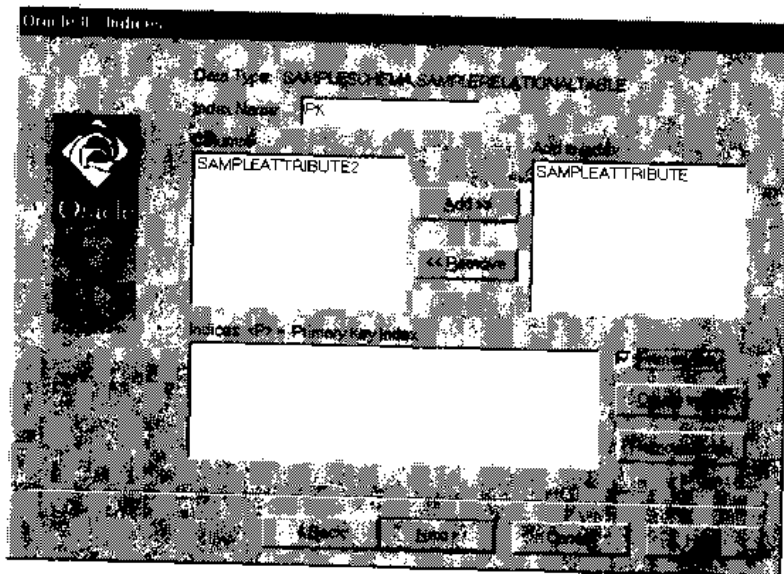


图18.12 Indices屏幕

要生成索引，首先输入索引名。在Columns列表框中，选择索引要包括的列并按Add按钮。如果要想该索引成为主索引，选择Primary Key框。按Create Index按钮生成新索引，它会出现于屏幕底部的Indices列表框中。输入索引名生成另一个索引，如果已经生成了表格的所有索引，按Next按钮。

出现的下一个屏幕是Foreign Keys窗口，如图18.13。可以对表格生成一个或几个外部关键字。首先，在FK (Foreign Key) Name下拉列表框中输入一个外部关键字名。在Tables列表框选择表格，其主关键字成为当前表格的外部关键字。从列表框中选择表格时，其主关键字出现在Columns列表框中。找到相应列后，在Columns列表框中选择并按Add Foreign Key按钮。新的外部关键字出现在屏幕底部的Foreign Keys列表框中。要加入另一个外部关键字，在FK Name下拉列表框中输入另一外部关键字名，并重复上述步骤。加进所有外部关键字后，单击Next按钮。

属性是哪些。Rose在关系上放上与外部关键字同名的关联名，并设置关联版型<<FK>>。图18.16显示了这个例子。

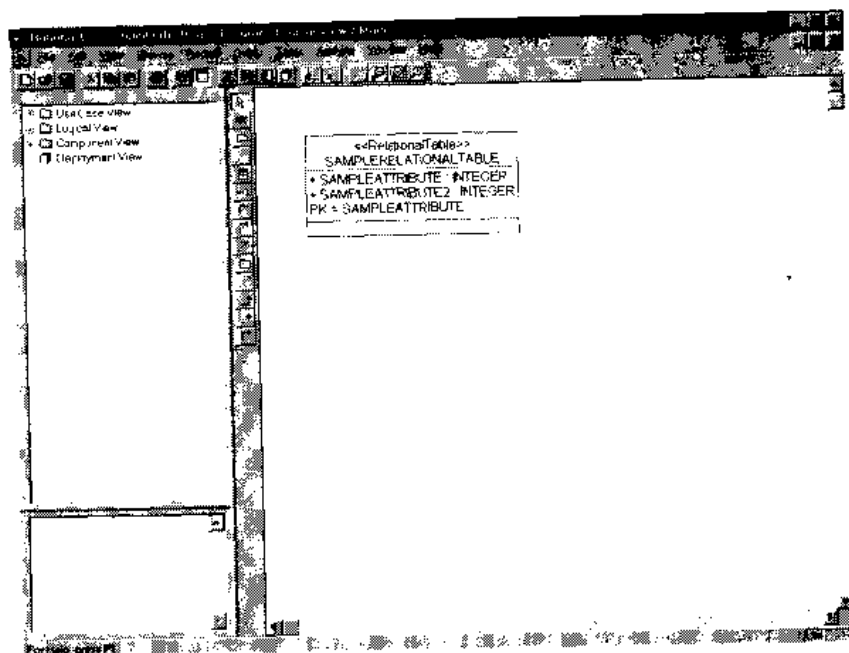


图18.15 Sample RelationalTable类

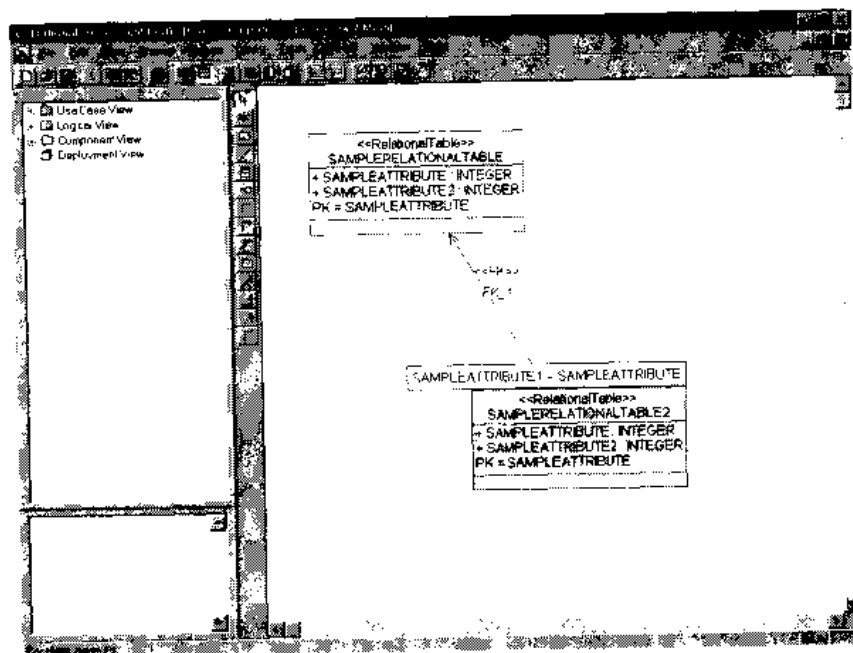


图18.16 Class框图上的外部关键字

关系型视图

关系型视图是个虚拟表格，其信息来自多个表格。Rose中的关系型视图显示为类，版型为RelationalView。这个类的每个属性实现为所生成关系型视图的属性。

要生成Relational视图，选择Tools>Oracle8>Data Type Creation Wizard。在出现的窗口中，选择Relational View，如图18.17。

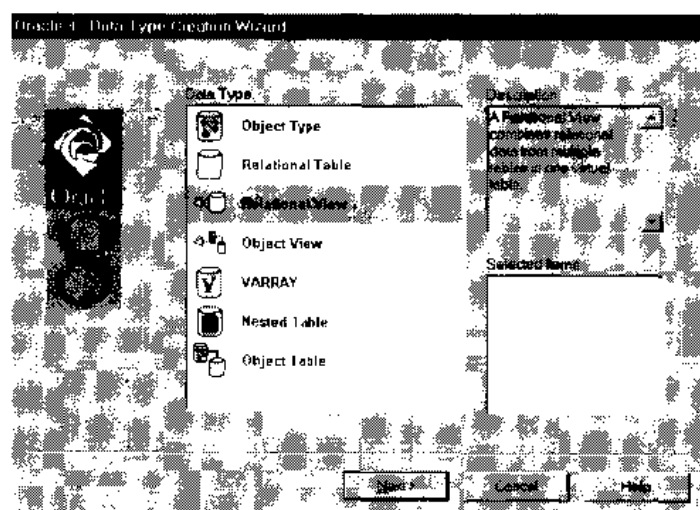


图18.17 生成Relational视图

选择Relational View后，单击Next按钮，在Data Type Description屏幕中，输入新的关系型视图名，关系型视图的结构（组件），包含该类型的包和显示该类型的Class框图。如果喜欢，也可以在窗口底部输入文档。Data Type Description窗口如图18.18。

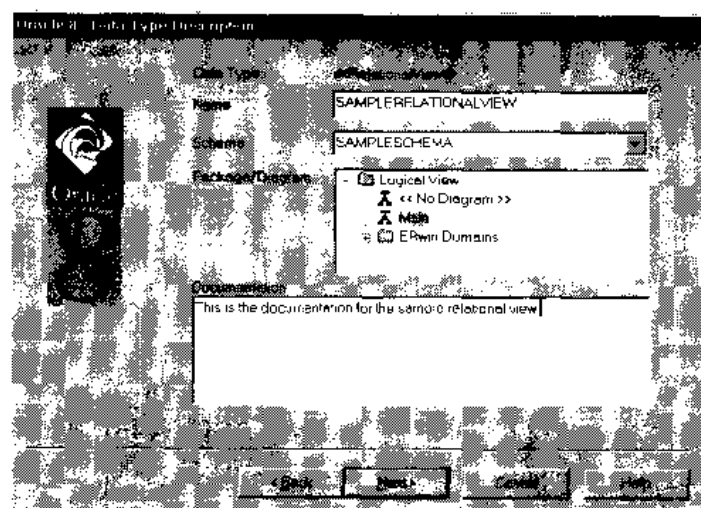


图18.18 Data Type Description屏幕

在图18.19所示的Define Column窗口中，可以选择现有属性或对关系型表格生成新列（属性）。

要选择现有列，选择Map From按钮。如果选择关系型表格或视图之后再启动向导，则其已经在Map From框中显示。选择框中的项目，或增加新列。选择Map From框中的表格可将其所有列加进视图中。要删除视图中的列，选择该列并单击Remove按钮。

要对视图输入Where从句，单击Where按钮。在Where Clause窗口中输入Where从句，如图18.20。

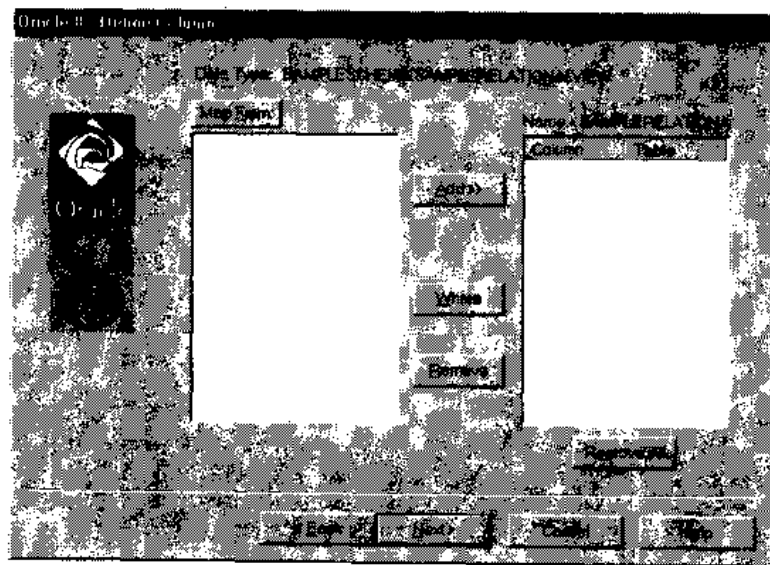


图18.19 Define Column窗口

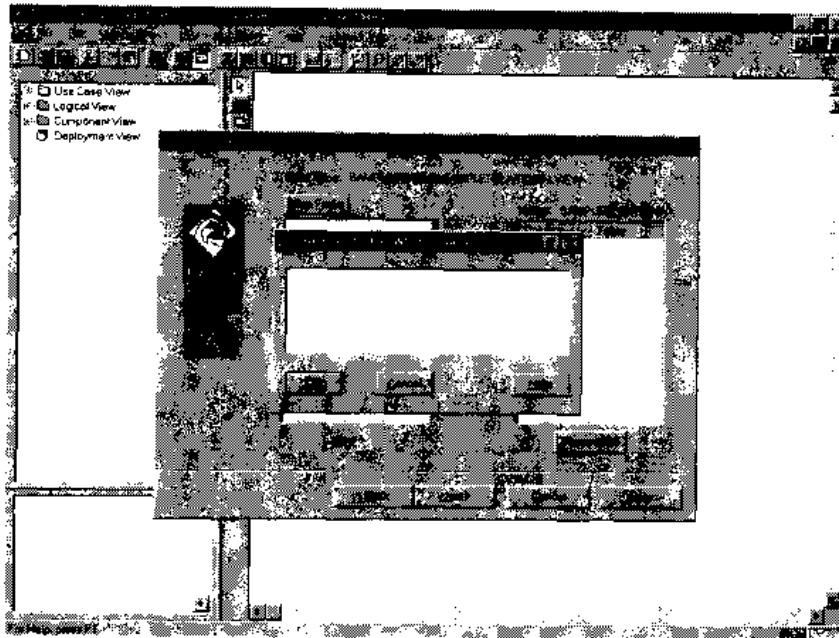


图18.20 定义Where从句窗口

然后可以改变列顺序。在图18.21所示的Ordering窗口中，列出了这个关系型视图设置的所有列。要改变列顺序，将列拖动到新位置。顺序正确之后，单击Finish按钮。

注意Ordering窗口是Oracle8的另一向导。生成数据类型后，可以选择Tools►Oracle8►Ordering Wizard回到Ordering窗口。

Rose生成版型为RelationalView的类，如图18.22，类中包括用向导定义的所有列。

对象视图

对象视图是个虚拟对象，其信息来自多个表格。在Rose中，对象视图显示为版型为Objectview的类。类的每个属性实现为所生成对象视图的属性。

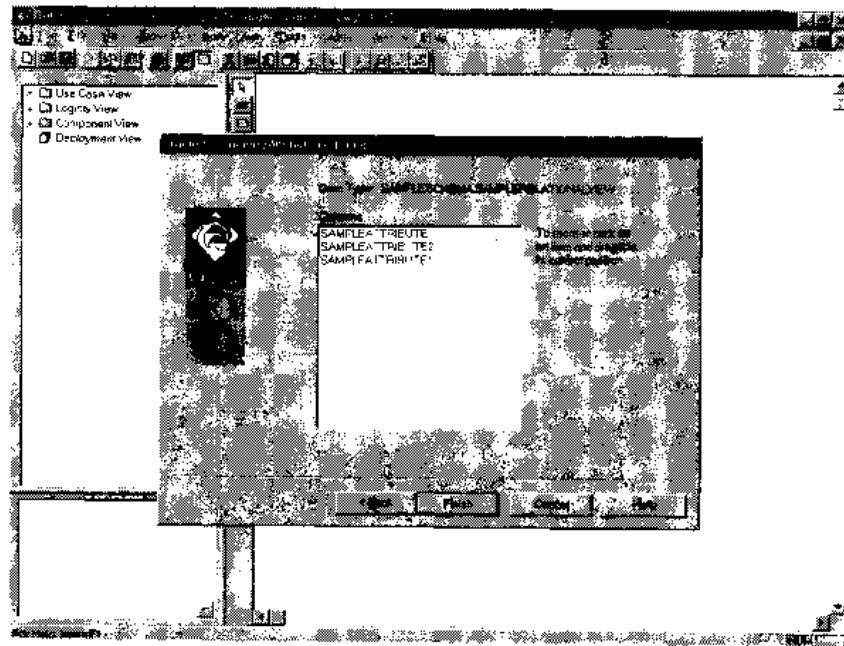


图18.21 Ordering窗口

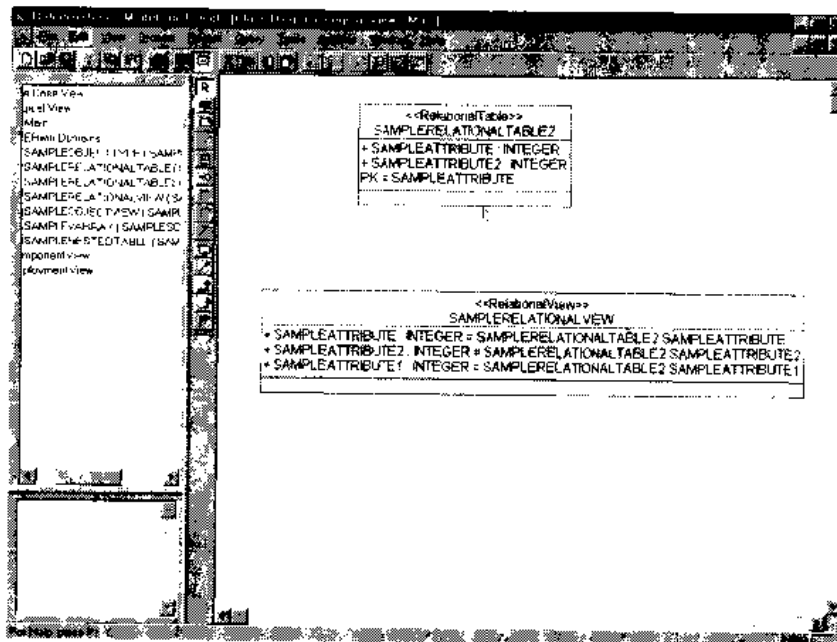


图18.22 SampleRelationalVable类

要生成Object视图，选择Tools>Oracle8>Data Type Creation Wizard。在出现的窗口中，选择Object View，如图18.23。

选择Object View后，单击Next按钮。在Data Type Description屏幕中，输入新的对象视图名，对象视图的结构（组件），包含该类型的包和显示该类型的Class框图。如果喜欢，也可以在窗口底部输入文档。Data Type Description窗口如图18.24。

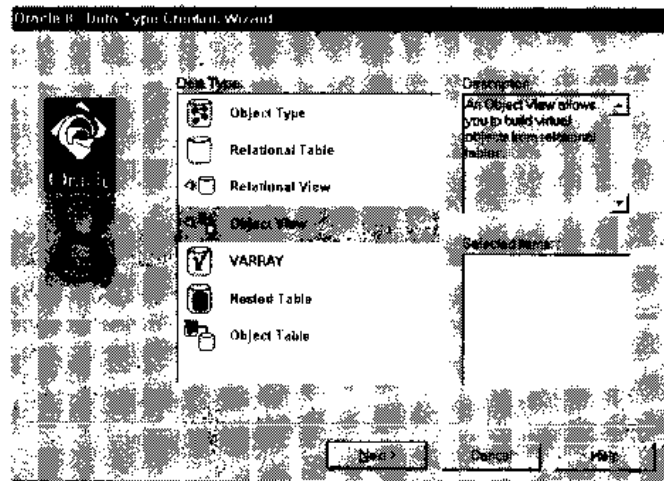


图18.23 生成Object视图

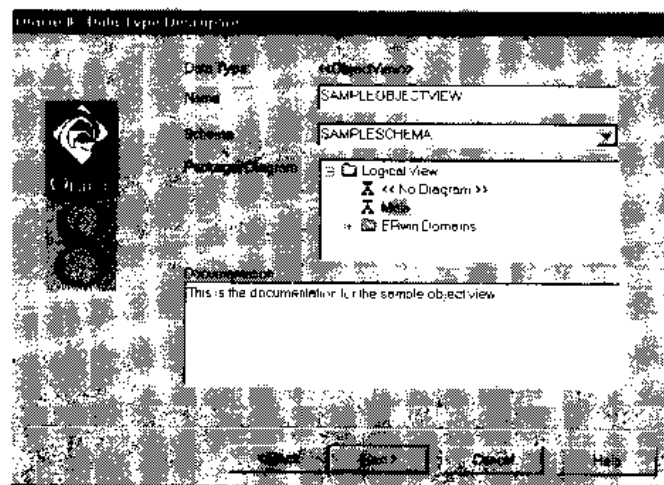


图18.24 Data Type Description屏幕

然后出现图18.25所示的Type Selection窗口。选择现有对象类型或单击New ObjectType按钮生成新的对象类型。



图18.25 Object Type Selection窗口

所选对象类型的属性出现在Object View Map窗口，如图18.26。单击Map To按钮显示一列表格和视图，属性可以从中映射。选择所要表格式视图并单击OK。选择对象视图中的属性和Map To列表框中的相应属性，然后单击Map。映射所有属性后，单击Where按钮输入视图的Where从句。在Where Clause窗口输入Where从句，如图18.27。映射所有属性和输入所有Where从句后（如需要），单击Next按钮继续。

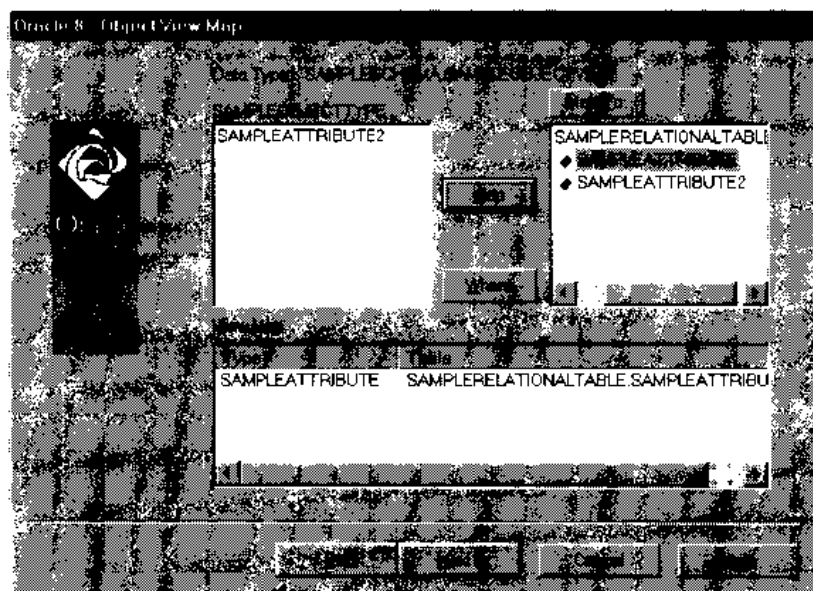


图18.26 Object View Map窗口

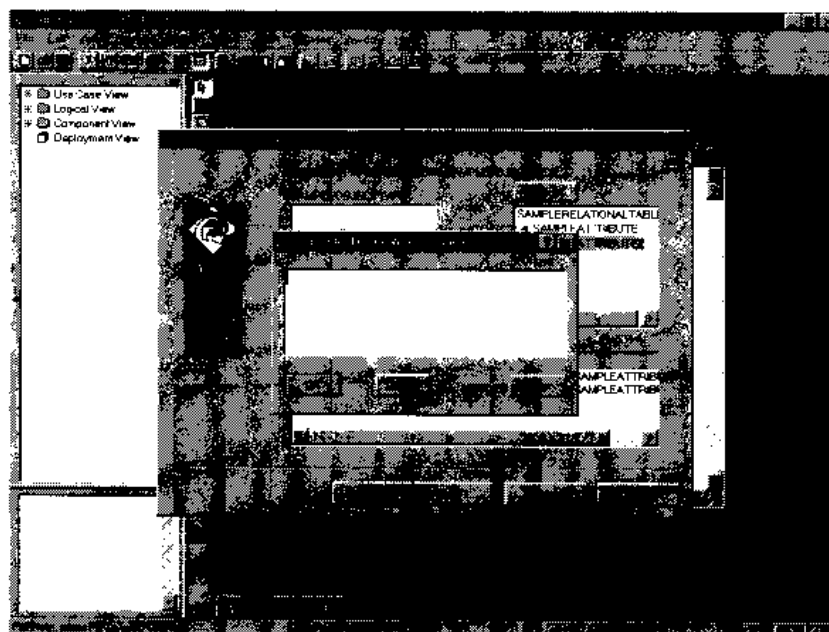


图18.27 输入Where从句

在下一个窗口，生成视图的对象标识符，类似于主关键字。从Attributes列表框中输入要放在对象标识符中的属性。单击Add按钮将这些属性加进对象标识符中。完成之后单击Finish按钮。

Rose生成版型为ObjectView的类，如图18.28。Rose从对象视图中向对象视图中的每个表格生成依赖性。

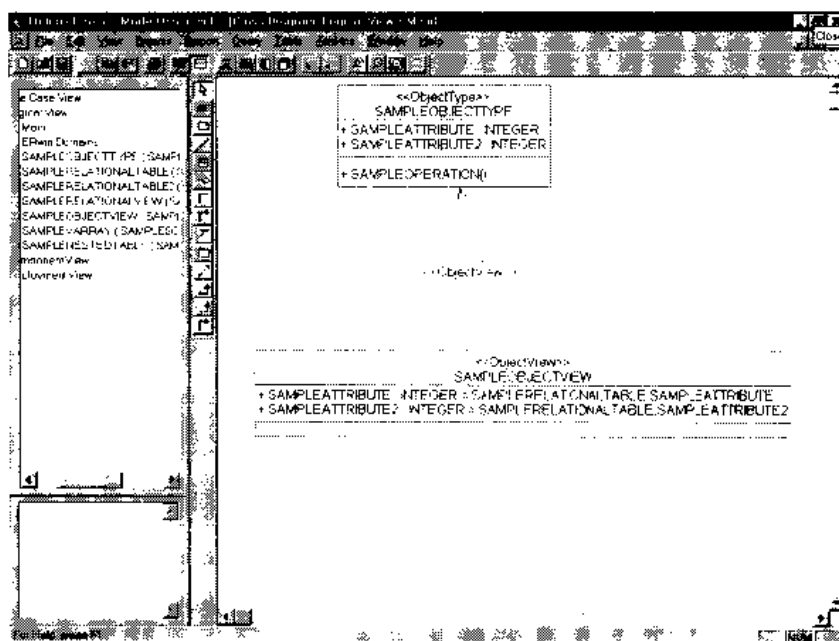


图18.28 SampleObjectView类

VARRAY

VARRAY是Oracle8中的容器类。是对象类型或标量（如数字、字符和日期）的数组。VARRAY显示为类，版型为VARRAY。

要生成VARRAY，选择Tools>Oracle8>Data Type Creation Wizard。在出现的窗口中，选择VARRAY，如图18.29。

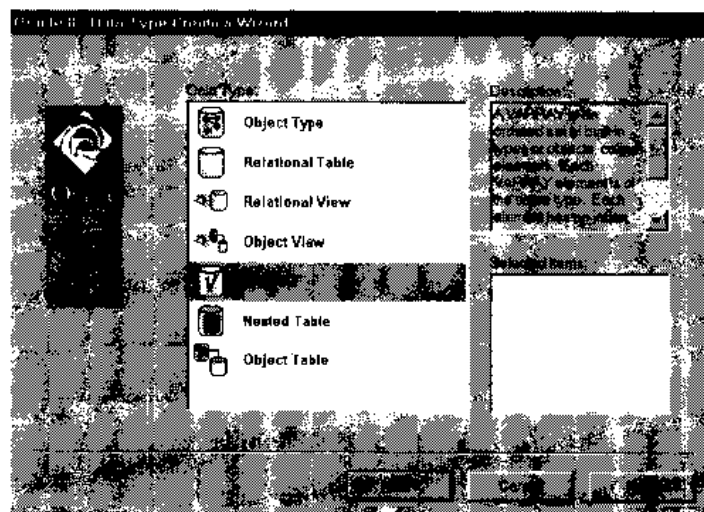


图18.29 生成VARRAY

选择VARRAY后，单击Next按钮。在Data Type Description屏幕中，输入新的VARRAY名，VARRAY的结构（组件），包含该类型的包和显示该类型的Class框图。如果喜欢，

也可以在窗口底部输入文档。Data Type Description窗口如图18.30。完成之后，单击Next按钮继续。

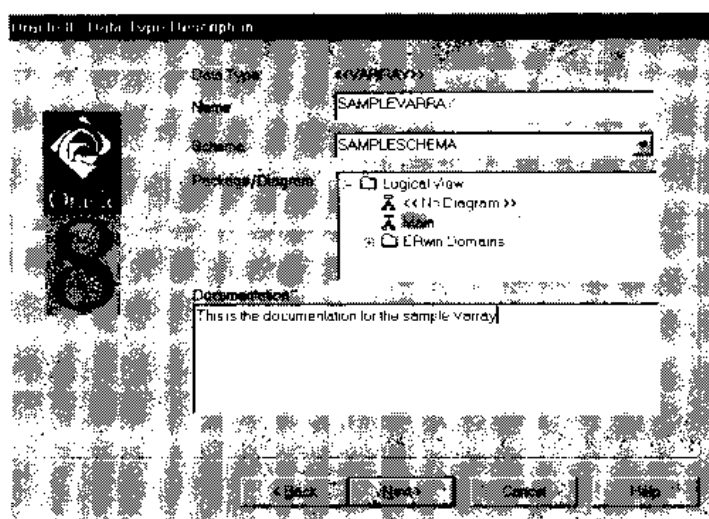


图18.30 Data Type Description屏幕

出现Type Selection窗口，如图18.31。为VARRAY选择类型，对象类型和标量。如果需要，输入精度、长度或比例。最后，输入VARRAY的基数性（或长度）。完成之后单击Finish按钮。

Rose生成版型为VARRAY的类，如图18.32。Rose生成从VARRAY到指定对象类型的依赖性。

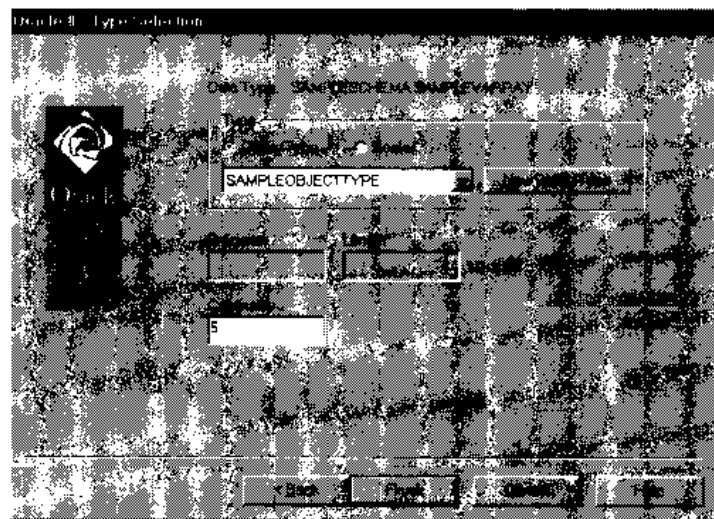


图18.31 Type Selection窗口

嵌套表

嵌套表是表现为表格列的对象，可以是任何对象类型或标量，如数字、字符和日期。嵌套表显示为类，版型为NestedTable。

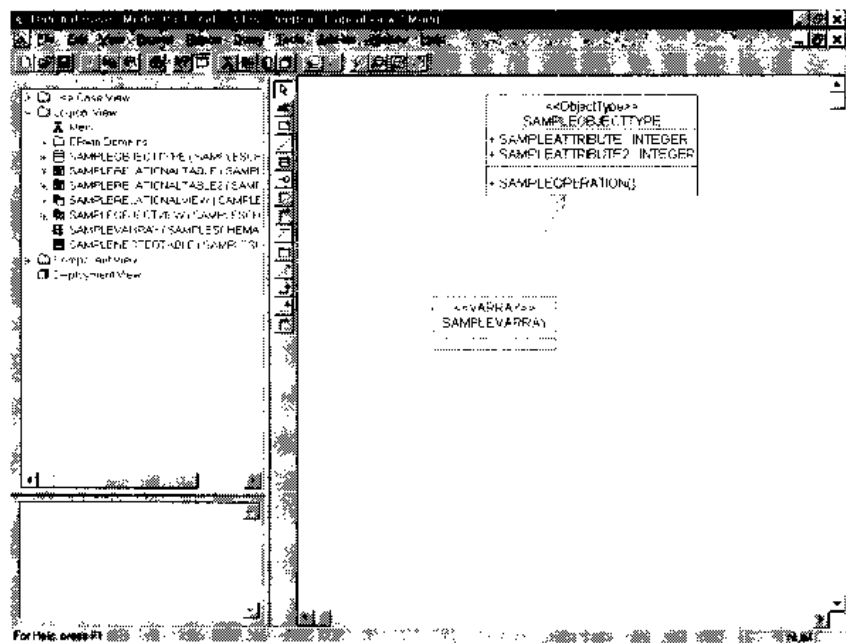


图18.32 版型为VARRAY的类

要生成Nested表，选择Tools>Oracle8>Data Type Creation Wizard。在出现的窗口中，选择NestedTable，如图18.33。

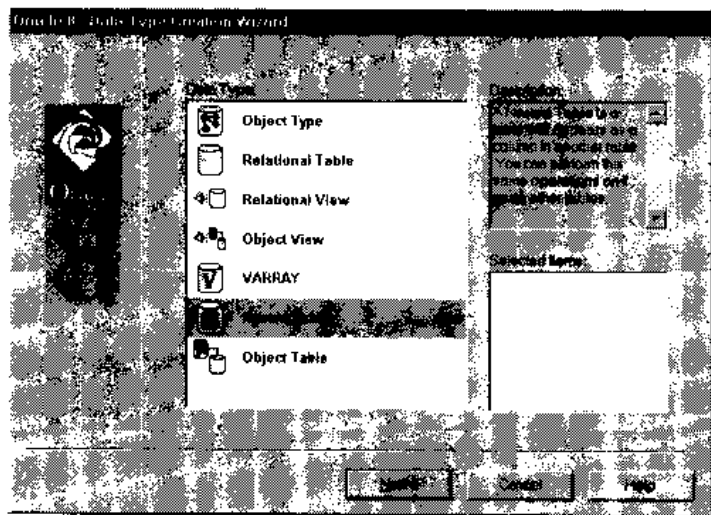


图18.33 生成Nested表

选择NestedTable后，单击Next按钮。在Data Type Description屏幕中，输入新的嵌套表名，嵌套表的结构（组件），包含该类型的包和显示该类型的Class框图。如果喜欢，也可以在窗口底部输入文档。Data Type Description窗口如图18.34。完成之后，单击Next按钮继续。

出现Type Selection窗口，如图18.35，选择嵌套表类型，可以是对象类型或标量。如果需要，输入精度、长度或比例。完成之后单击Finish按钮。

Rose生成版本为NestedTable的类，如图18.36。Rose生成从嵌套表到指定对象类型的依赖性关系。

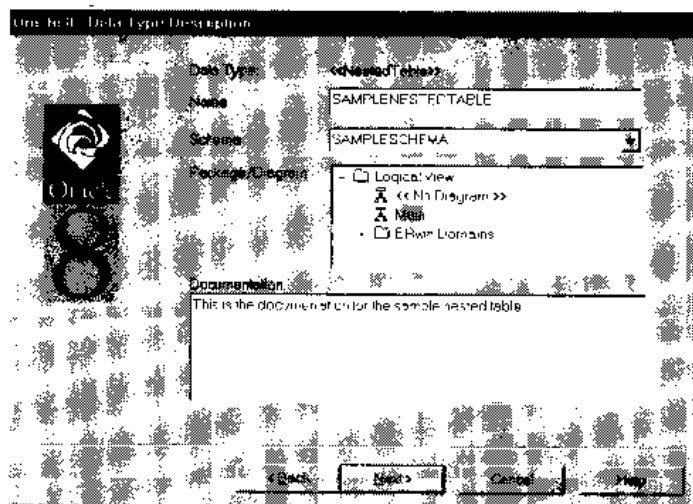


图18.34 Data Type Description屏幕

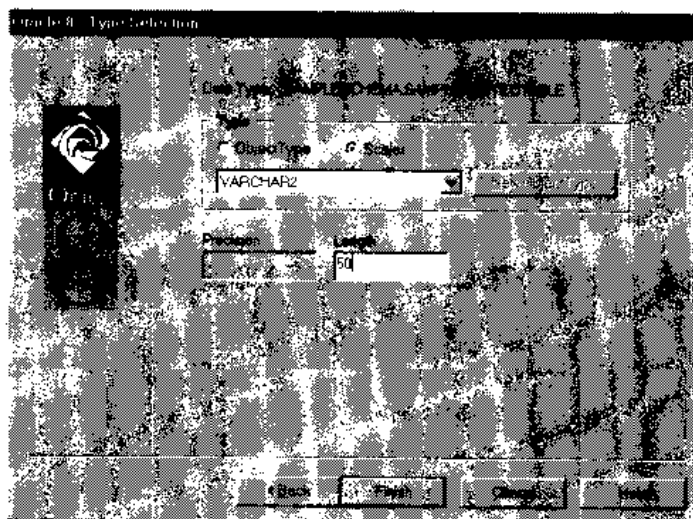


图18.35 Type Selection窗口

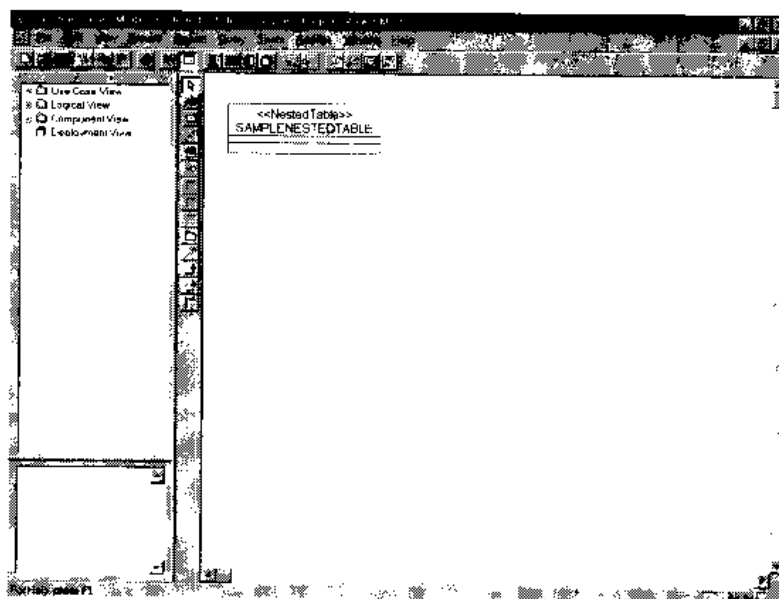


图18.36 SampleNestedTable类

对象表

对象表是对象类型的集合，可以是结构中指定的任何对象类型。对象表显示为类，版型为Object Table。

要生成Object表，选择Tools►Oracle8►Data Type Creation Wizard。在出现的窗口中，选择Object Table，如图18.37。

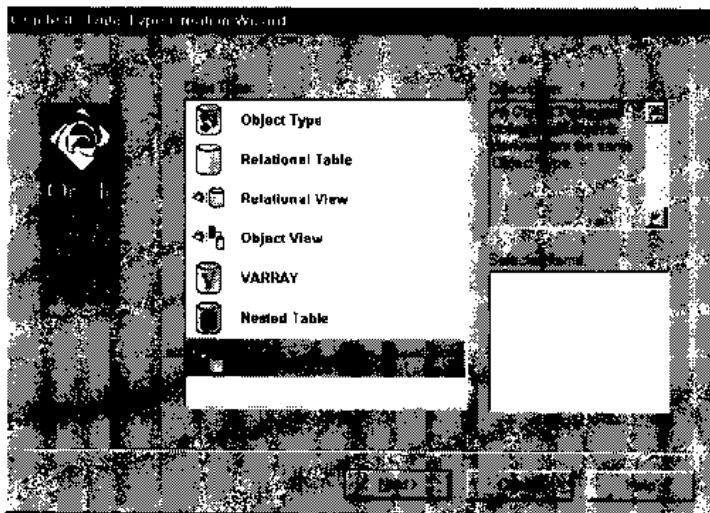


图18.37 生成Object表

选择ObjectTable后，单击Next按钮。在Data Type Description屏幕中，输入新的对象表名，对象表的结构（组件），包含该类型的包和显示该类型的Class框图。如果喜欢，也可以在窗口底部输入文档。Data Type Description窗口如图18.38。完成之后，单击Next按钮继续。

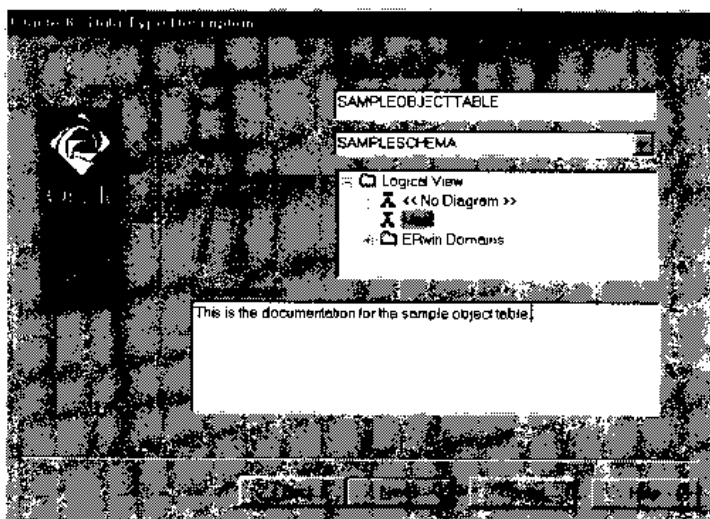


图18.38 Data Type Description屏幕

出现Type Selection窗口，如图18.39。选择对象表类型，完成之后单击Finish按钮。

Rose生成版型为ObjectTable的类，如图18.40。Rose生成从对象表到指定对象类型的依赖性关系。

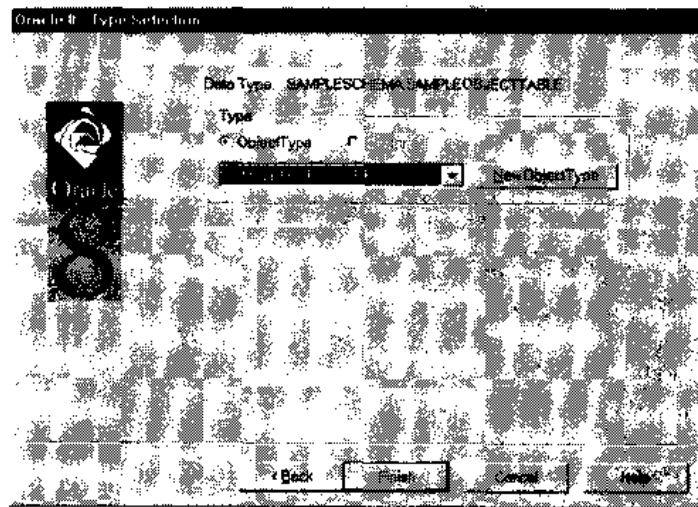


图18.39 Type Selection窗口

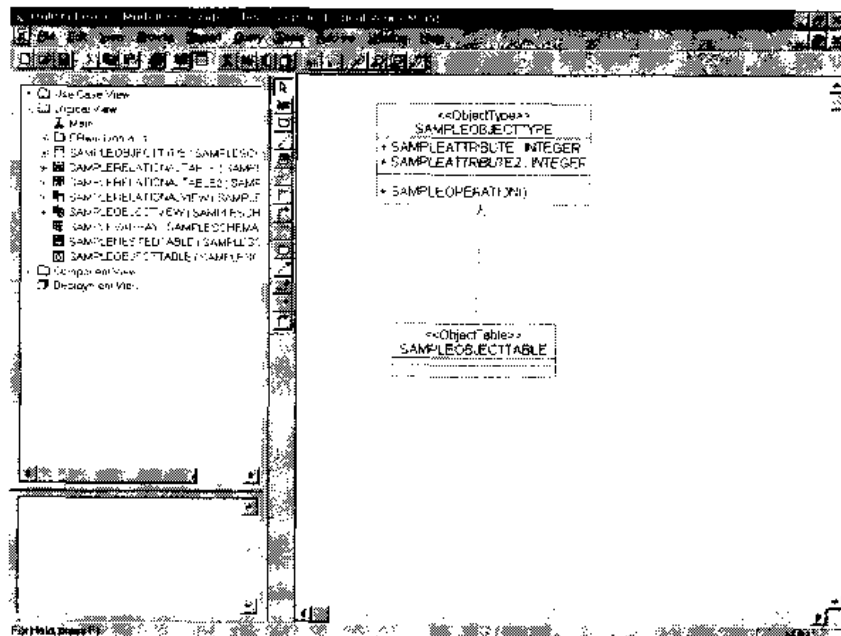


图18.40 SampleObjectTable类

生成的代码

生成Oracle8数据类型后，即可生成建立数据库结构所需的语句。生成语句之前，还可以进行语法检查，看看是否有什么错误。为此，选择要检查的对象，然后选择Tools►Oracle8►Syntax Checker。

要生成建立表、视图和对象的语句，选择要生成的对象，然后选择Tools►Oracle8►Schema Generation，出现Schema Generation窗口，如图18.41。

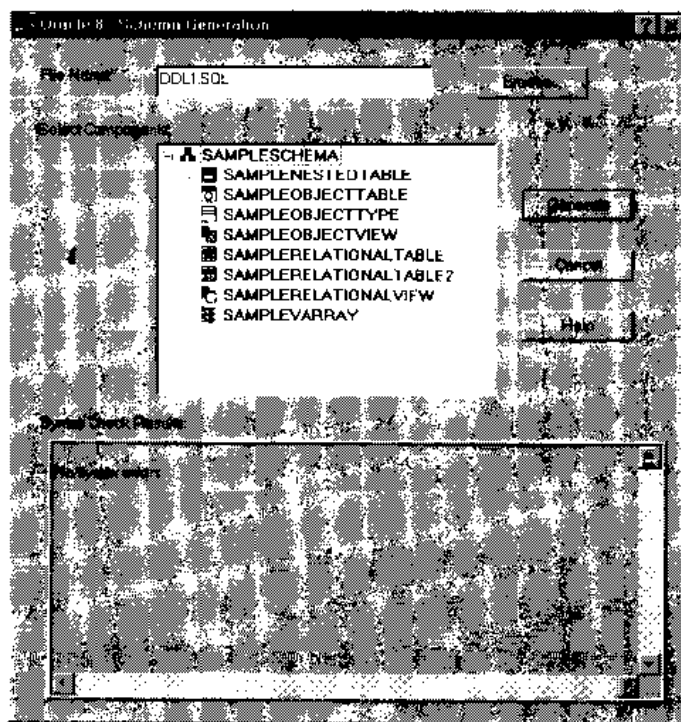


图18.41 Schema Generation窗口

这里可以指定语句的文件名，选择要生成的结构和对象。要生成的对象显示为黑体。要取消对象，单击即可使对象名不显示黑体。单击Generate即可生成语句，出现DDL Execution窗口，上半部分显示执行期间遇到的错误，下半部分显示要执行的语句，这时可以按Execute执行语句，条件是安装了Oracle8客户机软件并能连接Oracle8数据库。

小结

本章介绍各种Rose模型元素如何在Oracle8中实现。利用Data Type Creation Wizard向导，可以生成对象类型、关系型表、关系型视图、对象视图、VARRAY、嵌套表和对象表。然后可以生成在Oracle8数据库中建立这些对象所需的DDL，并在需要时执行DDL，建立这些对象。

下一章介绍用Rational Rose进行逆向转出工程代码的概述。后面几章介绍如何从C++、Java、Visual Basic和PowerBuilder代码生成Rose模型，以及如何逆向转出工程的Oracle8结构。

第19章 用Rational Rose逆向转出工程代码简介

- 逆向转出工程代码以显示现有系统的组织和结构

逆向转出工程代码就是利用源代码中的信息生成或更新Rose模型。通过集成C++、Java、Visual Basic和许多其他语言，Rose 98和98i支持将代码逆向转出工程代码为UML模型。信息技术项目的一个挑战就是保持对象模型与代码的一致。随着要求的改变，人们可能直接改变代码，而不是先改变模型再从个这个模型生成代码。逆向转出工程代码可以帮我们使模型与代码保持同步。

下面几章介绍从各种语言代码逆向转出工程代码为Rose模型的细节。在逆向转出工程代码过程中，Rose从代码读取组件、包、类、关系、属性和操作。将这些信息读进Rose模型中后，就可以进行所需的改变，然后通过Rose的正向工程特性重新生成代码。

可用的选项取决于所用的Rose版本。

- Rose Modeler不包括逆向转出工程代码功能。
- Rose Professional包括一种语言的逆向转出工程代码功能。
- Rose Enterprise包括C++、Visual C++（用于98i）、Visual Basic和Java逆向转出工程代码和Oracle8结构逆向转出工程代码功能。
- Rose Add-ins包括其他语言的逆向转出工程代码功能，如PowerBuilder或Forte等。

逆向转出工程代码生成的模型元素

逆向转出工程代码过程中，Rose收集下列元素的信息：

- Classes（类）
- Attributes（属性）
- Operations（操作）
- Relationships（关系）
- Packages（包）
- Components（组件）

利用这个信息，Rose可以生成或更新对象模型。根据逆向转出工程代码所用语言，可以生成新Rose模型或更新当前Rose模型。

下面首先介绍类、属性和操作。如果源代码文件包含类，则逆向转出工程代码过程生成Rose模型中的相应类。类的每个属性和操作出现为Rose模型中新类的属性和操作。除了属性和操作名外，Rose还取得可见性、数据类型、缺省值等信息。

例如，下列Java类进行逆向转出工程代码时，Rose生成图19.1所示的模型：

```
// Source file: SampleClass.java

public class SampleClass {
    public double PublicAttribute;
```

```

private int PrivateAttribute = 1;
protected int ProtectedAttribute;

SampleClass() {
}

public int PublicOperation(Double Parameter1) {
}

public int PrivateOperation() {
}

protected Double ProtectedOperation(int Parameter1, Double
Parameter2) {
}
}

```

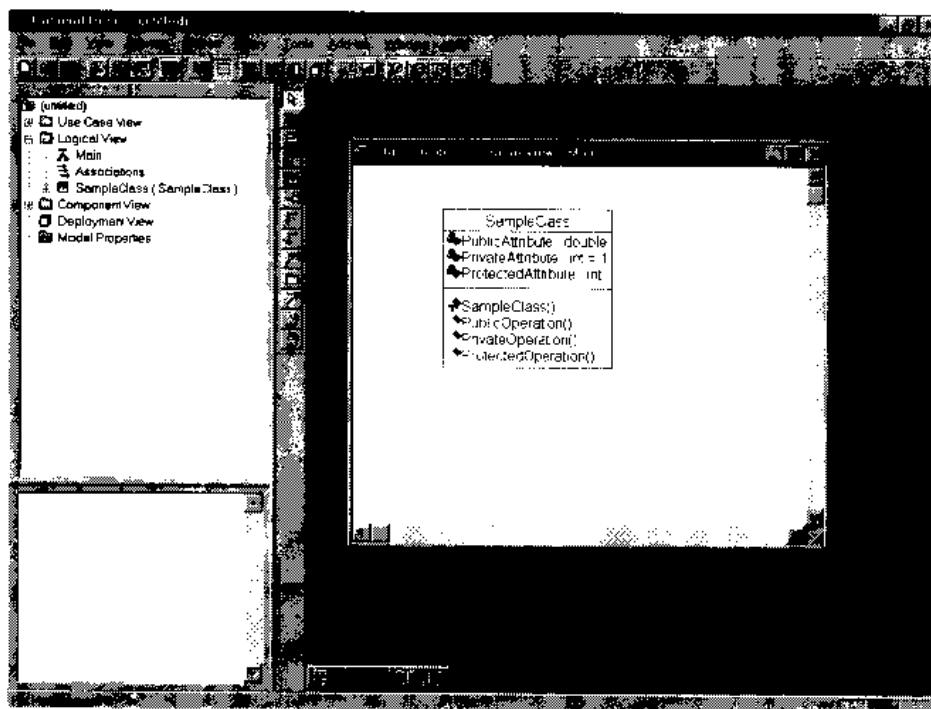


图19.1 Java类进行逆向转出工程代码

如果以前用Rose生成类，然后在代码中进行改变，则逆向转出工程代码过程中可以在模型中体现这些变化。例如，如果删除代码中的一个操作，则逆向转出工程代码过程中可以在模型中删除这个操作。如果直接将属性和操作加进代码中，逆向转出工程代码过程中可以在模型中增加这些属性和操作。

除了类以外，Rose还收集代码中的关系信息。如果一个类包含的属性数据类型为另一个类，则Rose生成这两个类之间的关系。例如，对于下面两个Java类，Rose在其间生成关联关系，如图19.2。

```

// Source file: Class_A.java

package javaclasses;

```

```
import java.jvclasses.Class_B;

public class Class_A {
    public Class_B m_Class_B;

    Class_A() {
    }
}

// Source file: Class_B.java

package javaclasses;

public class Class_B {

    Class_B() {
    }
}
```

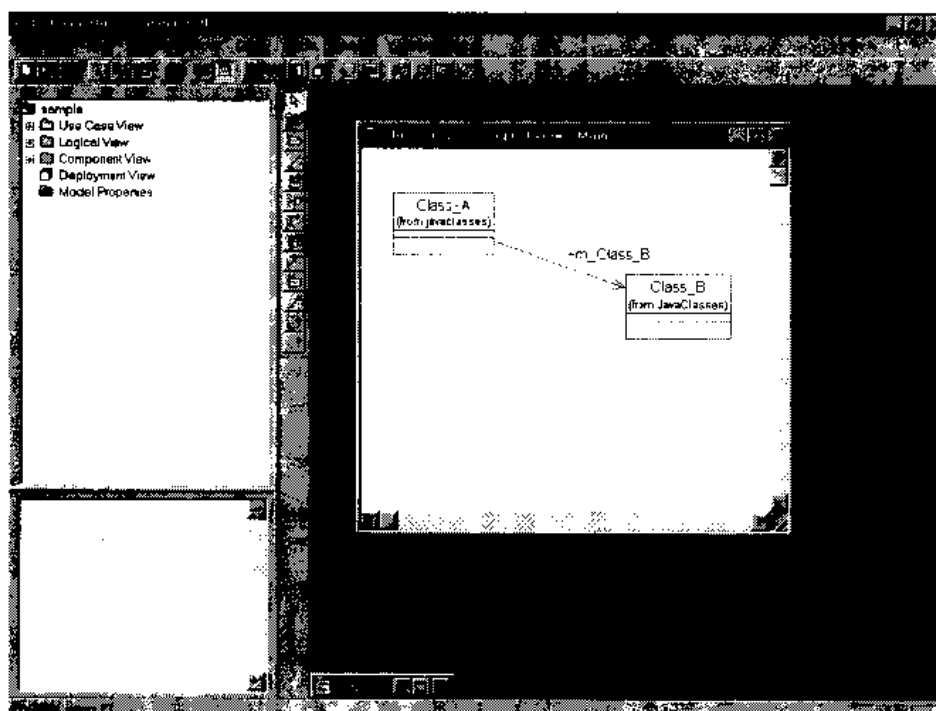


图19.2 关联关系的逆向转出工程代码

Rose模型中还生成继承关系。Rose生成一般化关系以支持代码中的继承。如果模型中有基础类包，如JDK或PowerBuilder系统类型，则Rose在逆向转出工程代码类与基础类之间生成一般化关系。例如，图19.3是逆向转出工程代码两个PowerBuilder窗口的结果，一个继承另一个，两者都继承PowerBuilder基础类Window。

逆向转出工程代码过程之后，代码中的组件也会在Rose中表示。每种语言处理组件的方法稍有不同。下面几章将介绍组件的逆向转出工程代码。

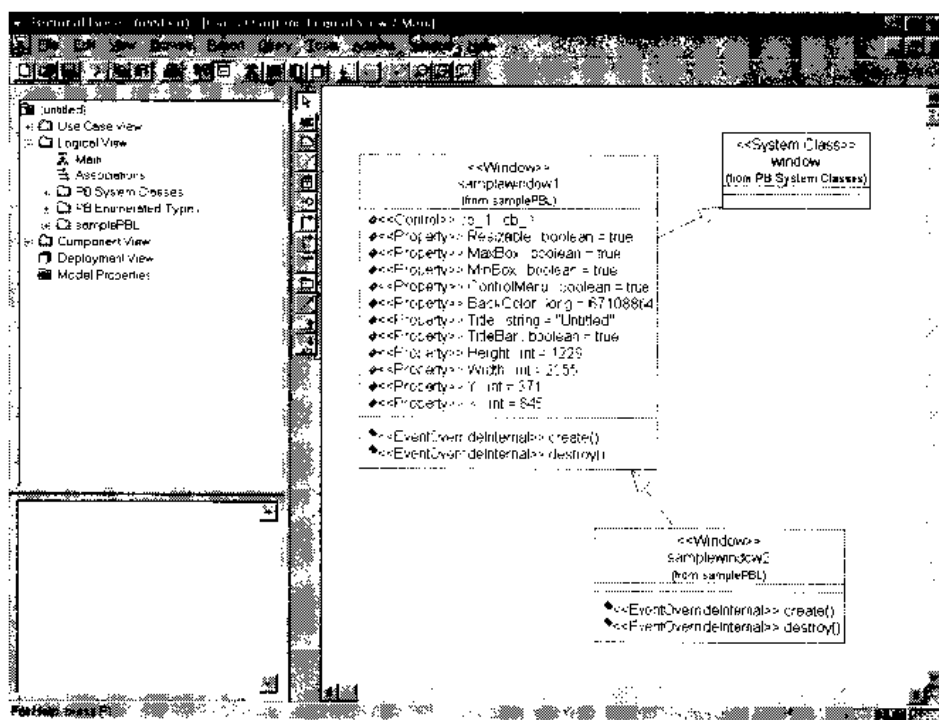


图19.3 一般化关系逆向转出工程代码

双向工程

用Rose生成代码时，生成代码中放上一个标识号。例如，代码中可能有如下语句：

```
@roseuid 36730C530302
```

这些数字和字母的字符串可以标识代码中的类、操作和其他模型元素，使Rose模型中的代码同步。

除了ID号以外，Rose在逆向转出工程代码过程中还在代码中生成保护区。保护区中的代码在双向工程期间受到保护。

例如，假设Rose生成的部分C++代码如下：

```
void SampleClass::DoSomething ()
{
    /// begin SampleClass::DoSomething%36EAB3DB03AC.body preserve=yes
    // - Code for the operation goes here -
    /// end SampleClass::DoSomething%36EAB3DB03AC.body
}
```

开发人员编写这个类的代码时，在//begin和//end语句之间编码DoSomething操作，放在保护区。对这个类进行逆向转出工程代码，作出改变，然后重新生成，DoSomething操作的代码保护不变。

小结

本章介绍了高层逆向转出工程代码过程。通过代码逆向转出工程代码，可以看出现有系统的结构和组织，还可以很详细地浏览类的细节。

Rose对C++、Java和Visual Basic等语言提供逆向转出工程代码功能，插入件还对PowerBuilder、Forte和其他面向对象语言提供逆向转出工程代码功能。关于Rose插入件的完整清单，见Rational的Web站点www.Rational.com。

下面几章介绍C++、Java、Visual Basic和PowerBuilder语言逆向转出工程代码功能，以及Oracle8结构的逆向转出工程代码。

第20章 C++与Visual C++逆向转出工程代码

- 启动C++ Analyzer应用程序并生成新项目
- 设置项目标题、目录表和扩展表
- 选择逆向转出工程代码的基础项目和文件
- 分析文件、设置输出选项并输出到Rose
- 用逆向转出工程代码向导进行Visual C++逆向转出工程代码
- 映射C++结构与Rose模型

目前软件界的C++代码数量巨大。尽管Rose支持从头生成模型的做法，但许多项目都要对现有代码进行逆向转出工程代码。要支持这些项目，Rose提供了C++逆向转出工程代码特性。

用逆向转出工程代码特性将现有代码看成Rose模型之后，可以进行必要的改变并用Rose的正向工程特性重新生成代码。这种双向工程功能可以使模型和源代码保持同步。

Rose 98i中的特性之一就是与Microsoft Visual C++ 6紧密集成。Rose中可以将Visual C++逆向转出工程代码为模型。

C++逆向转出工程代码步骤

本节介绍C++代码进行逆向转出工程代码所需的步骤。首先启动C++ Analyzer应用程序，它在Rose包中，提供C++逆向转出工程代码特性。

和C++代码生成一样，C++逆向转出工程代码也是非常容易定制的。C++ Analyzer应用程序有一系列选项，可以确定逆向转出工程代码的目标元素和如何在生成模型中显示各个C++结构。

第一步：启动C++ Analyzer应用程序

前面曾介绍过，C++逆向转出工程代码是通过Rose之外的应用程序进行的。利用这个应用程序可以选择逆向转出工程代码的文件、设置逆向转出工程代码属性和生成Rose模型。

首先选择Tools>C++>Reverse Engineering，启动C++ Analyzer应用程序，如图20.1。

第二步：生成新项目

启动C++ Analyzer应用程序后，下一步要生成新项目。项目就是Analyzer文件，包含一系列准备逆向转出工程代码的文件以及所有逆向转出工程代码属性。

选择File>New，即可在Analyzer中生成新项目，如图20.2。

第三步：设置项目标题

项目标题就是项目用途的简要说明。如果输入项目标题，而且后面要用这个项目作为另一项目的基础项目，则基础表中会出现这个标题。基础表将在下面介绍。

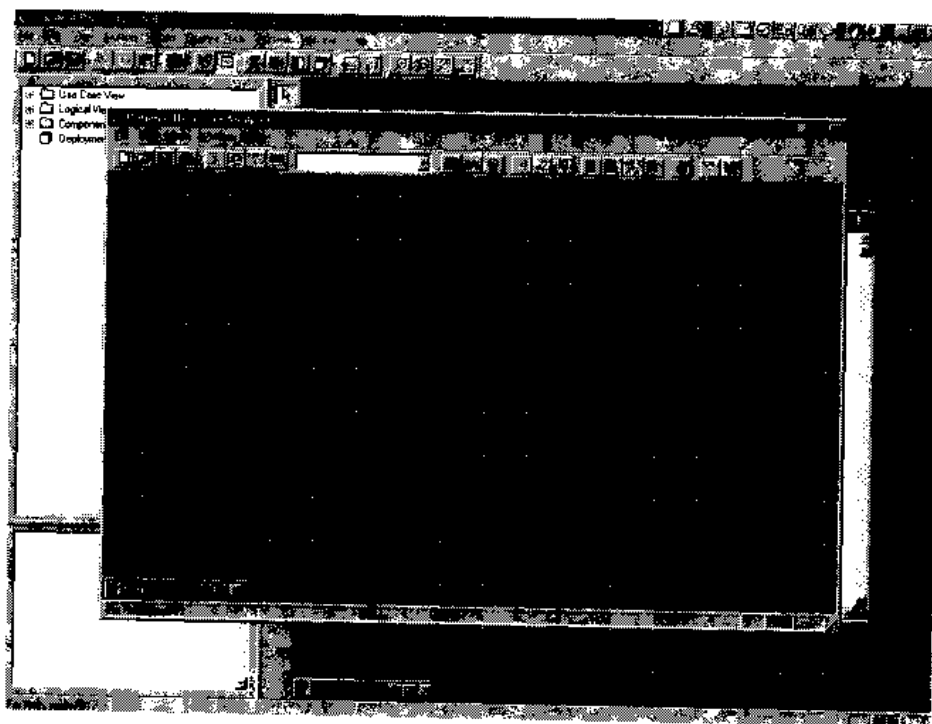


图20.1 启动C++ Analyzer应用程序

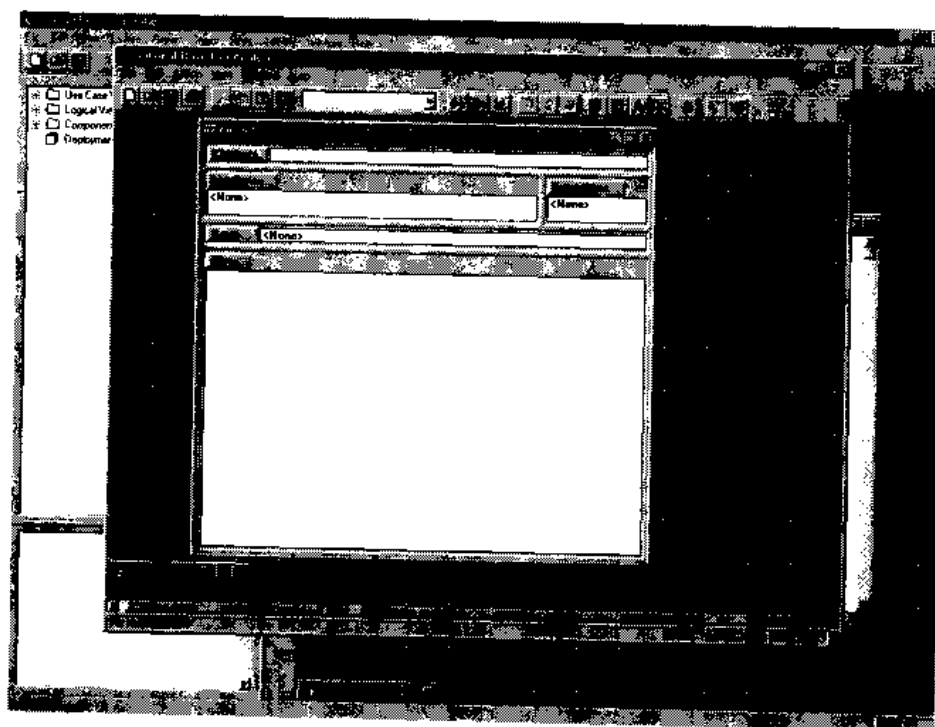


图20.2 生成新项目

标题是可选的，要设置项目标题，选择Project窗口的Caption按钮，然后可以用图20.3所示的Caption窗口输入项目标题。

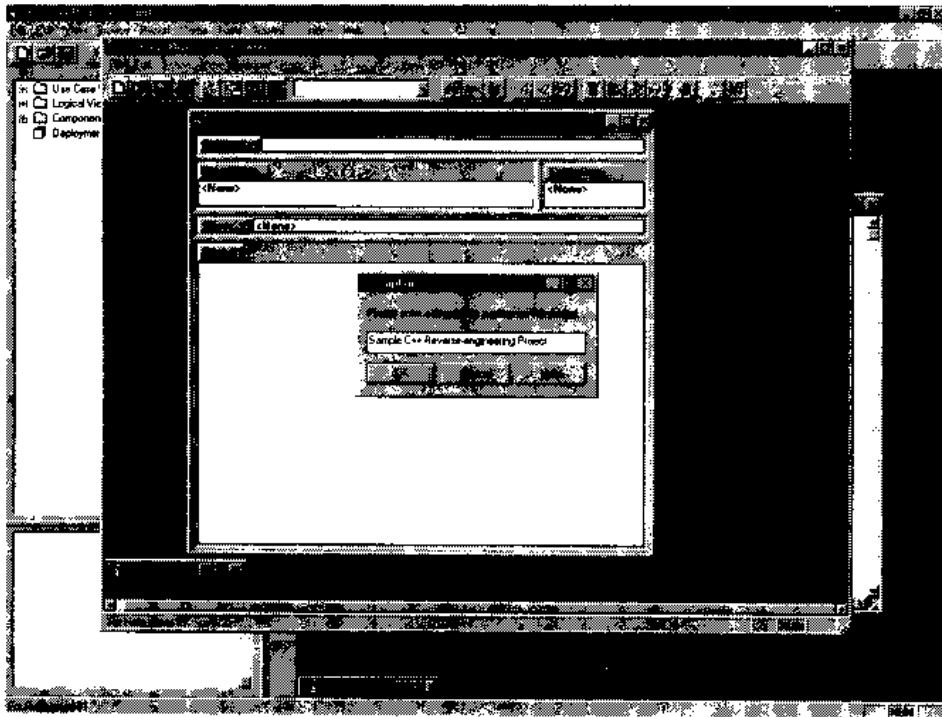


图20.3 输入项目标题

第四步：设置目录表

这一步是可选的。第七步选择文件时，会自动建立目录表。但可以在选择文件之前或之后修改目录表。

目录表指定要逆向转出工程代码的C++文件所在目录。利用项目窗口的Directories按钮（如图20.4），可以选择要加进表中的目录。

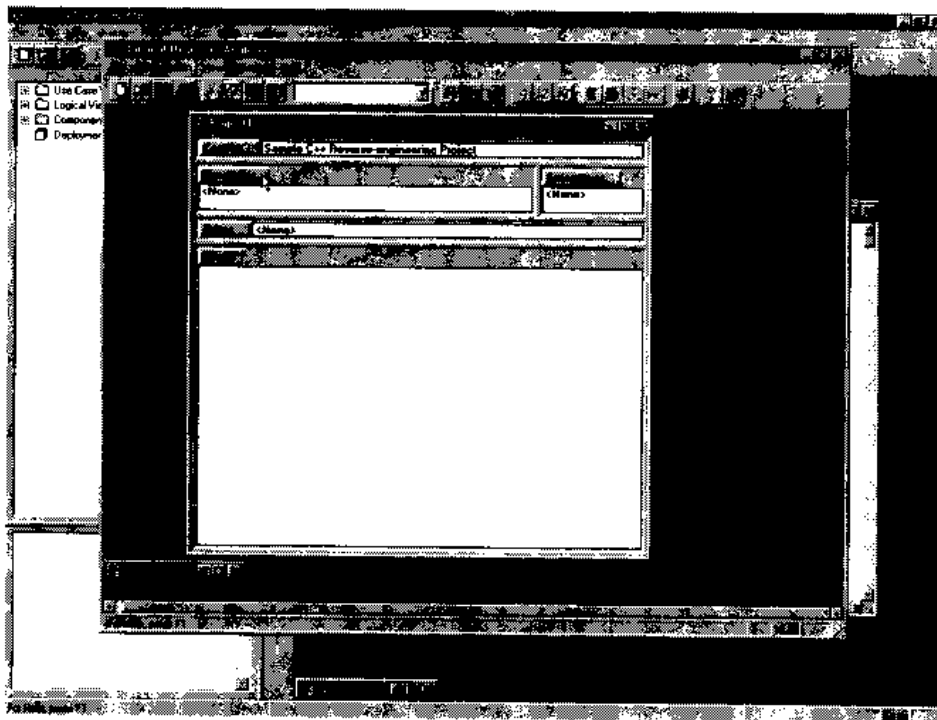


图20.4 设置目录表

选择这个按钮后，出现图20.5所示的Project Directory List窗口。

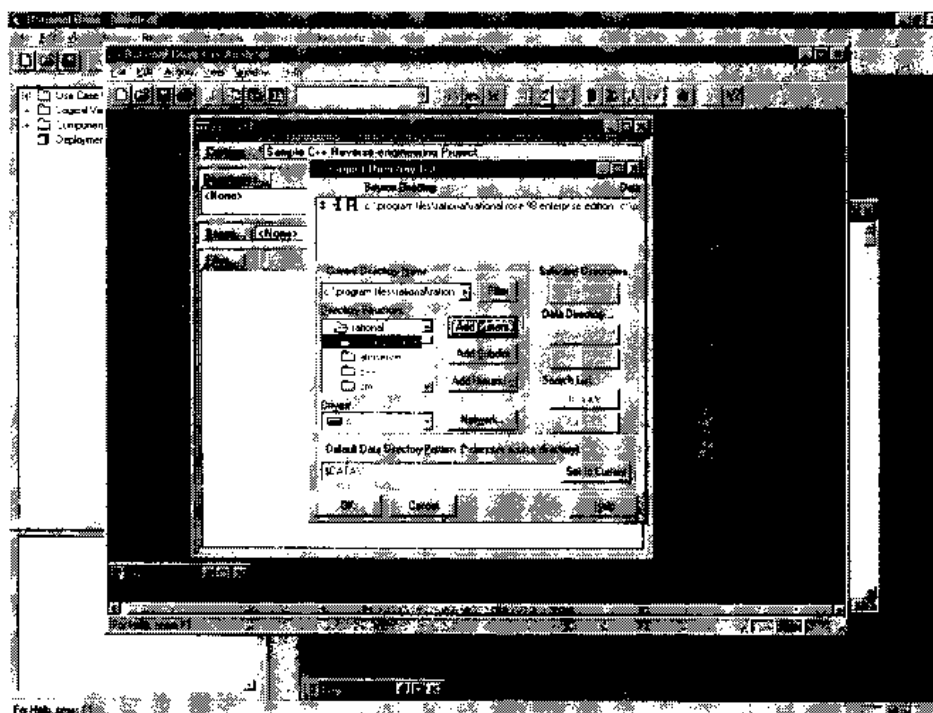


图20.5 将目录加进目录表

用Directory Structure列表框中的树视图选择要加的目录。按Add Current按钮增加目录，按Add Subdirs按钮增加目录及其直接子目录，按Add Hierarchy按钮加进目录及所有子目录和子目录下的整个树结构。

每个目录旁边显示三个图标。第一个可以在表中向上层或下层移动目录。要移动目录，用这个图标拖动目录向上或向下。移动图标如下：



第二个图标控制解析#include语句时是否搜索这个目录，缺省为搜索。如果要搜索，则出现下列图标：



否则出现下列图标：



第三个图标控制是否对这个目录中的项目生成代码。逆向转出工程代码完成后，可以对目录中的类进行改变，然后重新生成这些类的代码。但是，有时不希望重新生成代码（例如对第三方库和其他无权修改的代码）。利用代码生成图标可以确定今后能否修改代码。如果要重新生成代码，则出现下列图标：



否则出现下列图标:



第五步：设置扩展表

这一步是可选的。第七步选择文件时，清单中会自动填上这些文件的扩展名。但也可以先建立这个表。

扩展表是要逆向转出工程代码的源文件的文件扩展名表以及要逆向转出工程代码的源文件中包括的头文件扩展名表。Extension List窗口如图20.6所示。



图20.6 设置扩展表

加进表中的扩展名成为第七步选择逆向转出工程代码文件时的缺省过滤器。例如，如果在扩展表中增加.CPP，则.CPP成为选择文件时的缺省过滤器。

第六步：选择基础项目

基础项目就是包含所用基础类信息的C++ Analyzer项目文件。例如，C++ Analyzer项目可能包含你单位生成的C++基础类信息。公司的每个项目用这些基础类作为基础。包含这些基础类的路径名、文件和其他信息的C++ Analyzer项目成为当前项目的基础项目。

基础项目的好处是复用和一致性，可以在各个项目中复用组件，从而保证项目的一致性。基础项目窗口如图20.7。

逆向转出工程代码时，基础项目是可选的。要增加基础项目，用Directories和Filename字段找到路径和文件名，并单击Add按钮将其加进清单中。在项目窗口中，会出现基础项目及其标题。

利用Directory Structure树视图,选择逆向转出工程代码文件所在目录。缺省情况下,该项目中的所有.H、.HH、.HPP、.HXX、.CPP、.CXX、.CC和.C文件均会在Files Not in List (filtered)列表框中列出。如果扩展表中已有数值,则使用这些扩展名,而不用.H、.HH、.HPP、.HXX、.CPP、.CXX、.CC和.C文件。

要选择文件,首先在Files Not In List (filtered)列表框中选择,单击Add Selected按钮将其移到Files In List (filtered)列表框图中,或单击Add All按钮将所有文件移到清单中。

重复这个过程,选择目录和文件,直到Files In List (Filtered)列表框中包括要逆向转出工程代码的所有文件。然后按OK按钮,所选文件即会在项目窗口中列出。

第八步:分析文件

分析步骤从第七步选择的所有源代码文件中取出设计信息。如果遇到错误,则写入日志窗口。

要分析文件,首先在Files列表框图中选择文件。要选择所有文件,按Ctrl+A。选择Action>Analyze或按F3键即可开始分析。开始文件分析和检查时,日志中记录其进程和错误,并在Files列表框中显示每个文件的错误数。要浏览文件的错误,检查日志或双击Files列表框中的文件,即可显示这个文件的错误,如图20.9。

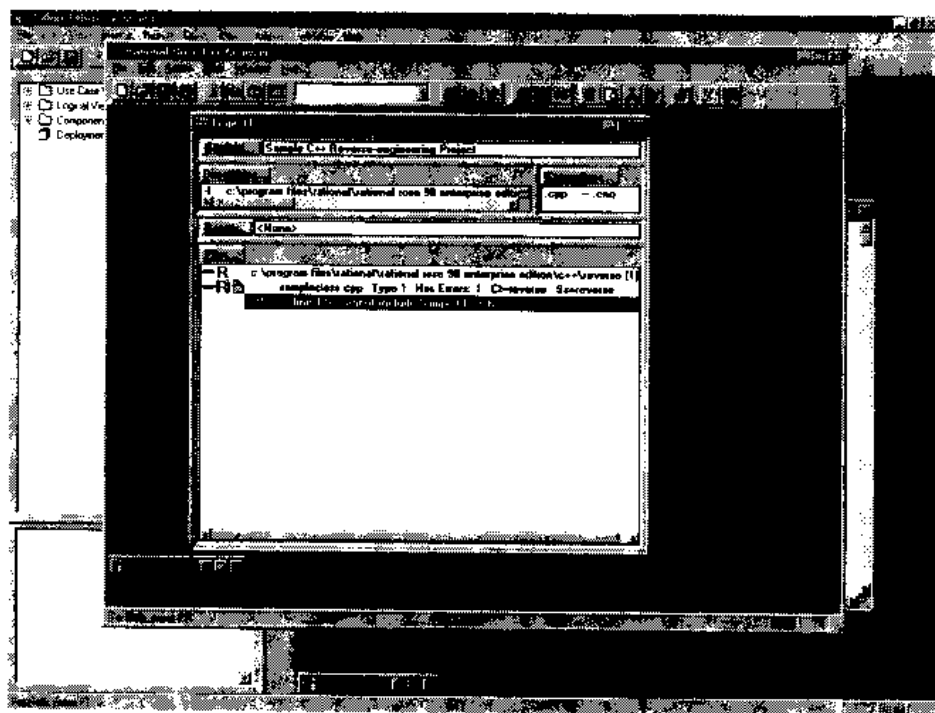


图20.9 浏览文件分析中的错误

第九步:设置输出选项

分析代码后,可以用这些信息生成Rose模型。Rose模型生成代码时有许多选项,C++ Analyzer中也有许多逆向转出工程代码选项。要浏览和设置这些选项,选择Edit>Export Options或按Ctrl+R,出现输出选项窗口,如图20.10。

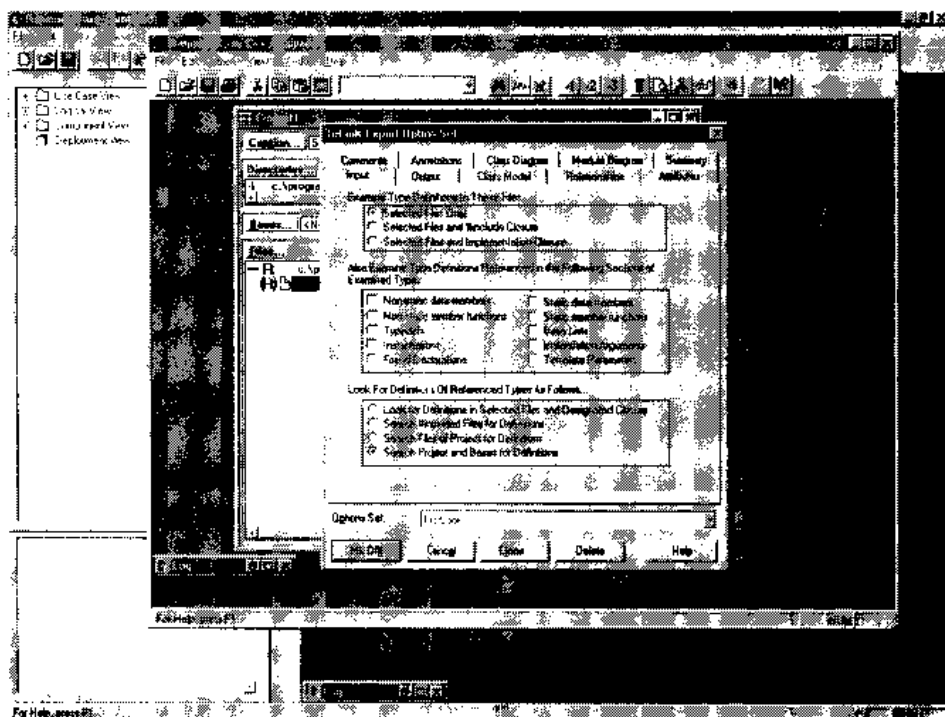


图20.10 设置输出选项

介绍这里提供的属性之前，我们先要介绍C++ Analyzer所带的三个基本选项组。在输出选项屏幕底部的Options下拉列表框中，可以选择FirstLook、RoundTrip或DetailedAnalysis。如果选择FirstLook，则设置这个窗口标签中的缺省属性以提供源代码的高级视图。如果选择DetailedAnalysis，则进行更深入的检查。最后，如果选择RoundTrip，则属性设置成支持双向工程（逆向转出工程代码现有代码为Rose模型，再将正向工程模型改变为代码）。

在Input标签中，可以设置确定检查内容的选项。第一组单选项Examine Type Definitions In These Files设置检查的文件。选项包括：

Selected Files Only 只检查Files清单中的文件。

Selected Files And #include Closure 检查清单中和#include指令中的文件（以及#include指令文件的#include指令中的文件）。

Selected Files And Implementation Closure 检查清单中和#include指令中的文件，以及与#include指令中的文件相关的实现文件。和上一选项一样，这个选项也是递归的（包括#include指令文件的#include指令中的文件）。

下一组选项Also Examine Type Definitions Referenced in the Following Sections of Examined Types确定逆向转出工程代码过程是否包括TypeDef、静态数据成员和其他C++结构。

最后，可以用Look For Definitions Of Referenced Types As Follows复选框设置类定义的搜索条件。

在Output标签中，可以控制Rose输出设置，如模型文件名，用UML、OMT还是Booch图注，以及是否生成Class框图。Output标签如图20.11。

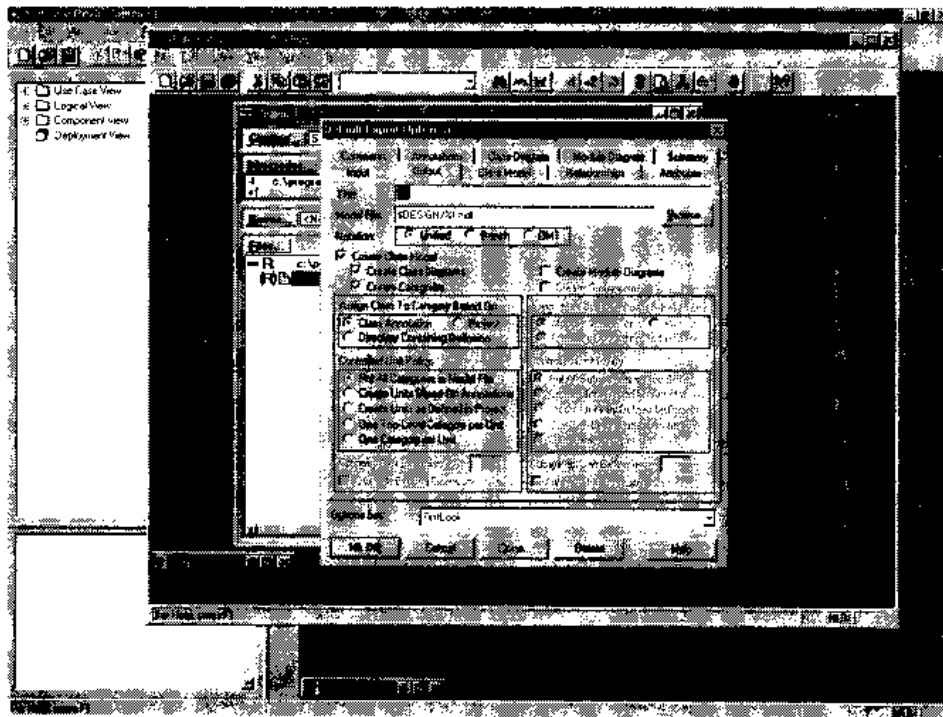


图20.11 设置输出选项

这个窗口的其他选项包括：

Title 设置生成框图的标题，用下列符号：

%c 项目标题名

%d 包含项目的简单目录名

%e 包含项目的完整目录名

%f 从所选File List项目生成的简单名称

如果选择单个文件，则产生的名称是所选文件的简单名称。如果选择多个文件，所选文件都属同一类别，则产生的名称是所选类别的名称，否则产生的名称是项目名。

%p 换成项目名

Create Class Model 允许在逆向转出工程代码的模型中生成类，可用上述符号。

Create Class Diagrams 确定Rose模型中是否生成Class框图。

Create Categories 确定是否生成逻辑视图包。

Create Module Diagrams 控制是否生成Component框图。

Create Subsystems 确定是否生成Component视图包。

Assign Class To Category Based On 确定类如何组成包。

Controlled Unit Policy 确定逻辑视图包是否设置为Rose中的控制单元。

Assign Module To Subsystem Based On 确定组件如何组成包。

Controlled Unit Policy 确定Component视图包是否设置为Rose中的控制单元。

在Class Model标签中，可以设置控制生成类、模板、模板实例、TypeDef和枚举的属性。这些微调选项可以在生成模型之前设置。Class Model标签如图20.12

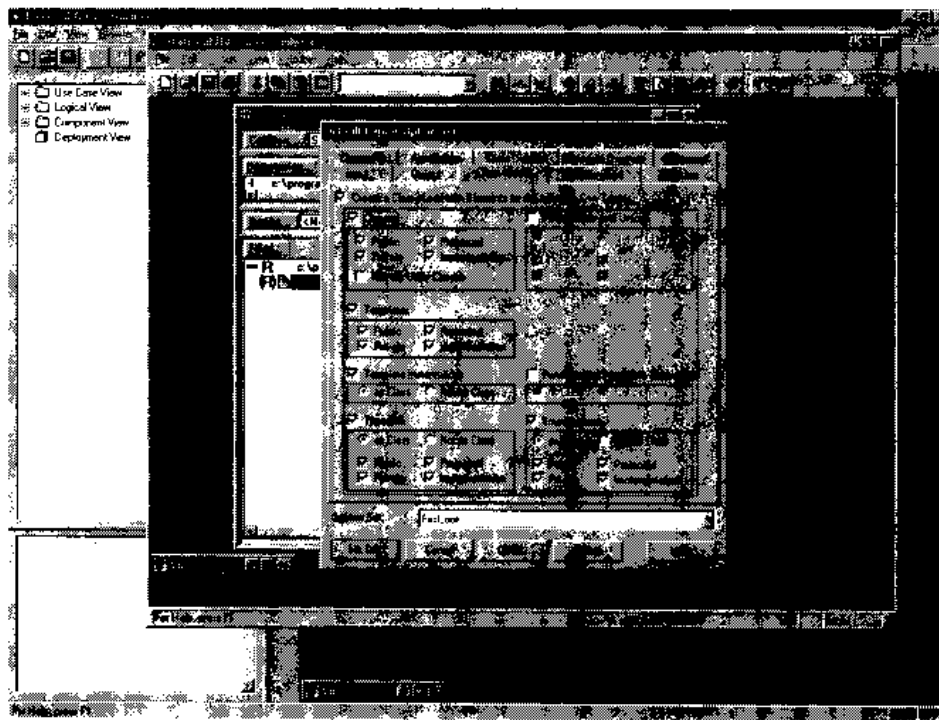


图20.12 设置类选项

这个标签的选项如下：

Create a Class Model With Elements For The Following C++ Types 控制模型中是否生成类、TypeDef和其他C++结构。

Classes 控制模型中是否将类生成为类。

Templates 控制模型中是否将模板生成为类。

Template Instantiations 控制模型中是否将模板实例生成为类。

Typedefs 控制模型中是否将TypeDefs生成为类。

POD Structs And Unions 控制模型中是否将结构和联合生成为类。

Fundamental Types (int, char等) 控制模型中是否将基础类型生成为类。

Enumerations 控制模型中是否将枚举生成为类。

Relationships 标签控制Rose模型中如何生成关系。图20.13是Relationships标签。

Relationships标签的选项如下：

Create 确定数据成员是否建模为关系。

Has Relationships or Associations 控制是否生成关联或累积。

Non-Static Data Member 控制生成模型中是否表示非静态C++数据成员。

Static Data Member 控制生成模型中是否表示静态C++数据成员。

Create Uses Relationships To Each Type Referenced In The Declaration of 控制模型中何时对各种结构生成使用 (use) 关系。

Create Inherits Relationships To Each 控制模型中生成的一般化关系。

Create Instantiates Relationships From Each Instantiation To Its Template 控制生成模型是否生成实例关系。

Create Uses Relationships From Each Class Declared “friend” 控制可见性为“朋友”的类之间是否建立关系。

Attributes标签控制Rose模型中如何生成属性和操作。这个标签如图20.14。

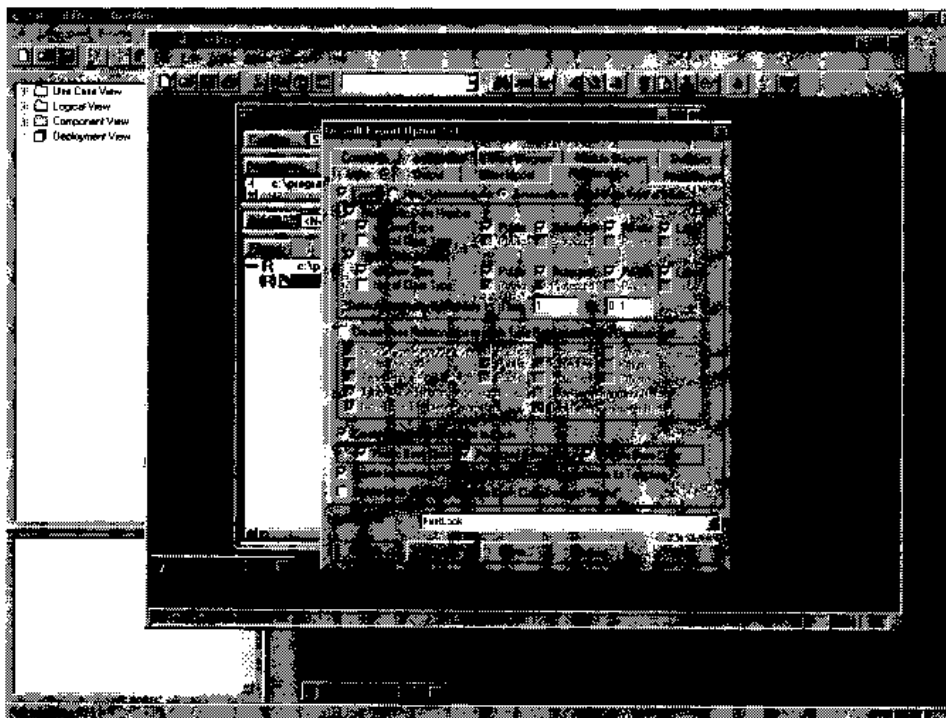


图20.13 设置关系选项

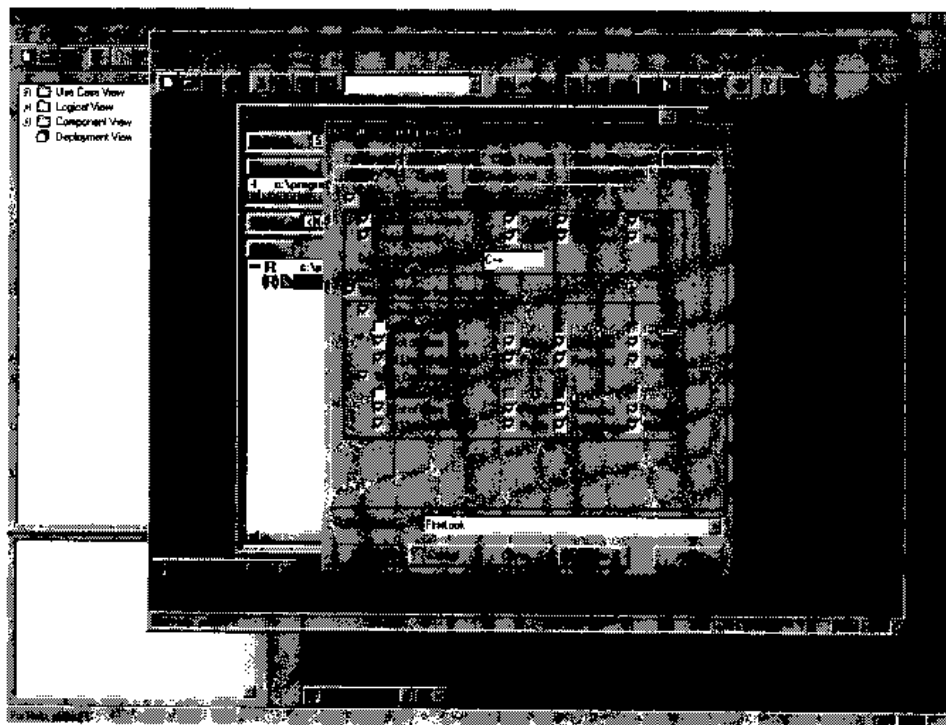


图20.14 设置属性选项

这个标签中可以设置的属性如下：

Create Operation Specifications For 控制Rose模型中生成操作的C++结构。

Create Attribute Specifications For 控制Rose模型中生成属性的C++结构。

Comments标签如图20.15，可以指定说明语句在代码中的位置。利用这个信息，C++ Analyzer可以将说明语句放到Rose文档窗口11。在Rose模型中，可以用文档窗口浏览不同模型元素的说明语句。



图20.15 设置说明选项

Annotations标签控制不同模型元素的Rose文档窗口是否放上图注注释信息。图注是Rose放在生成代码中的说明语句和标识符，用于支持双向工程。利用Annotations标签（如图20.16）可以确定是否将操作、关系、数据成员和其他代码元素的图注放进Rose模型以及放在哪里。

Class Diagrams标签控制是否生成Class框图，这些框图中显示什么内容。Class Diagrams标签如图20.17。

Class Diagrams标签中可用的选项如下：

Create Class Diagrams 控制是否生成Class框图。

Draw Categories 控制Class框图中是否放上包。

Diagram Name 设置生成的Class框图名。

Draw Model Elements Derived From The Following Constructs 控制Class框图中显示哪些元素（类、TypeDfe、枚举等）。

Draw Relationships Derived From Type References In The Following Constructs 控制Class框图中是否显示从成员变量和其他结构派生的关系。

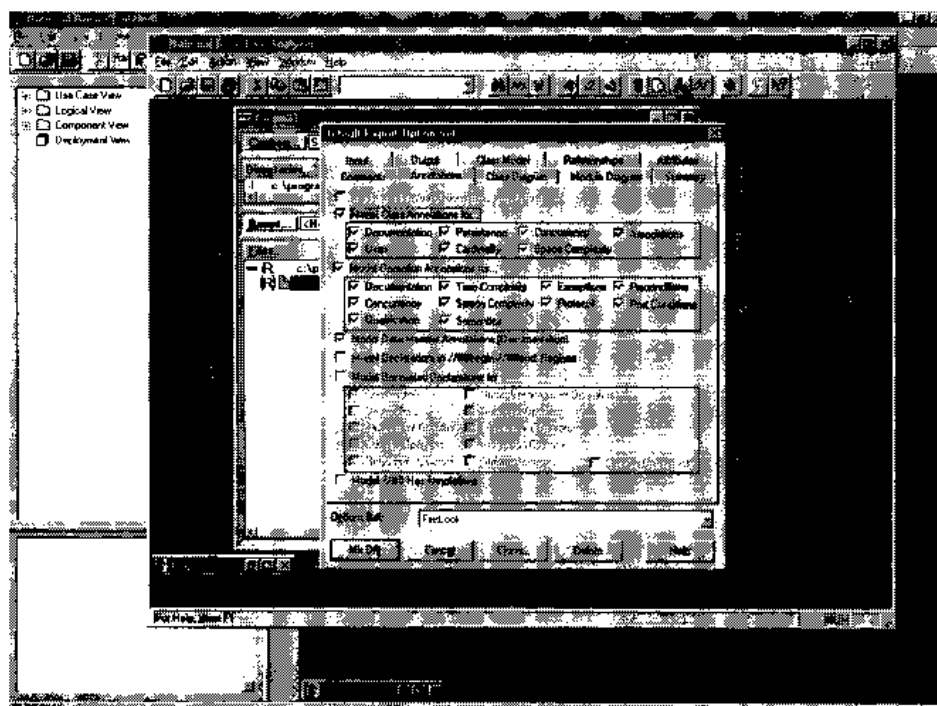


图20.16 设置图注选项

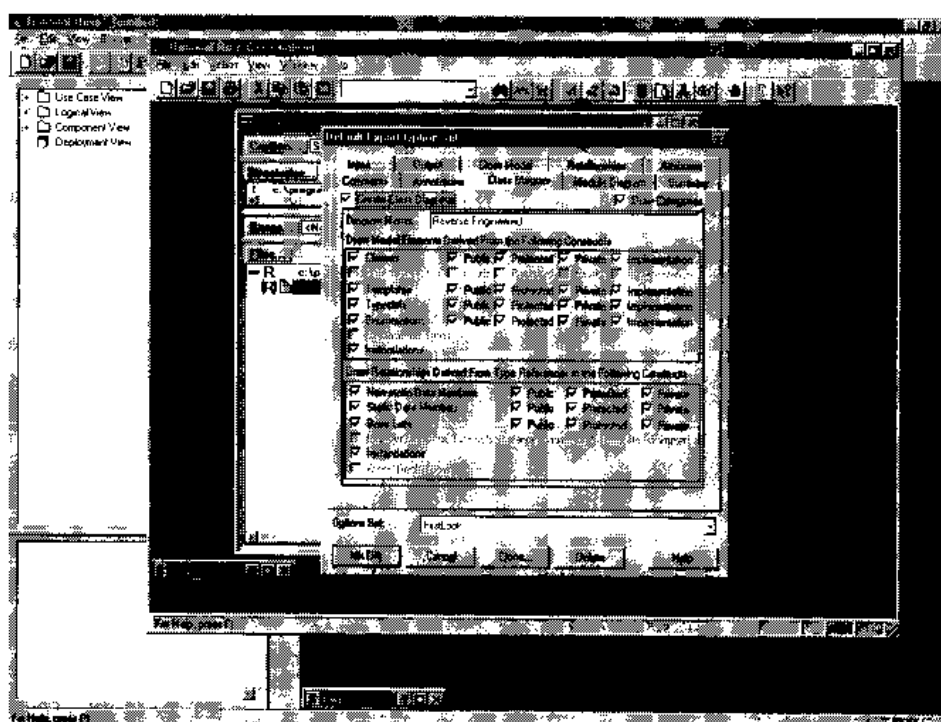


图20.17 设置Class框图选项

Module Diagram标签控制与逆向转出工程代码过程中生成的Component框图相关的选项。Component标签如图20.18。

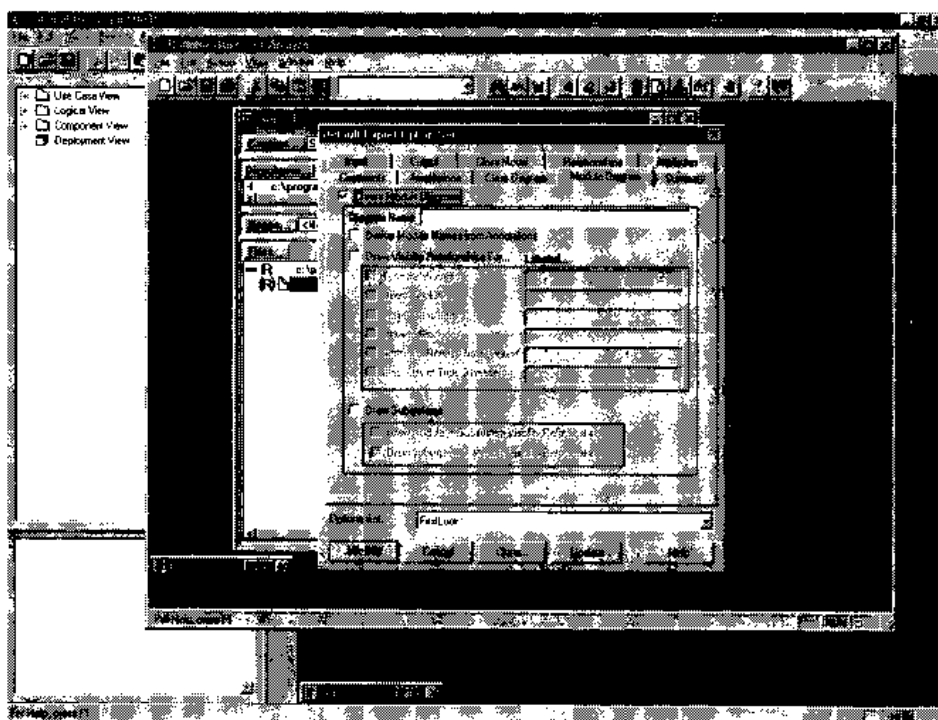


图20.18 设置Component框图选项

这个标签中的选项包括：

Create Module Diagrams 控制是否生成Component框图。

Diagram Name 设置生成的Component框图名。

Derive Module Names From Annotations 控制从Rose生成源代码时加进的图注中是否包含组件名，组件名是否从文件名派生。

Draw Visibility Relationships For 控制Component框图中是否标明#include指令和其他C++结构的关系。如果需要，本节还可以输入结构使用的标号。

Draw Subsystems 控制框图中是否出现Component视图包。

最后，Summary标签显示所有其他标签中的选项值，如图20.19。

第十步：输出到Rose

最后一步是把分析的文件输出到Rose模型。为此，选择Action►Export to Rose或按F8打开图20.20所示的Export To Rose对话框。

这个对话框可以设置生成的Rose模型名，项目标题和使用的选项设置，还可以在Summary Of Options区列出选项概述。要生成模型，单击这个窗口的OK按钮。如果指定文件名的模型已经存在，则Analyzer询问是否覆盖它。

Visual C++逆向转出工程代码的步骤

本节介绍Visual C++逆向转出工程代码为Rose模型所需的步骤。逆向转出工程代码过程有两种：从Visual C++代码建立新模型和更新现有模型。

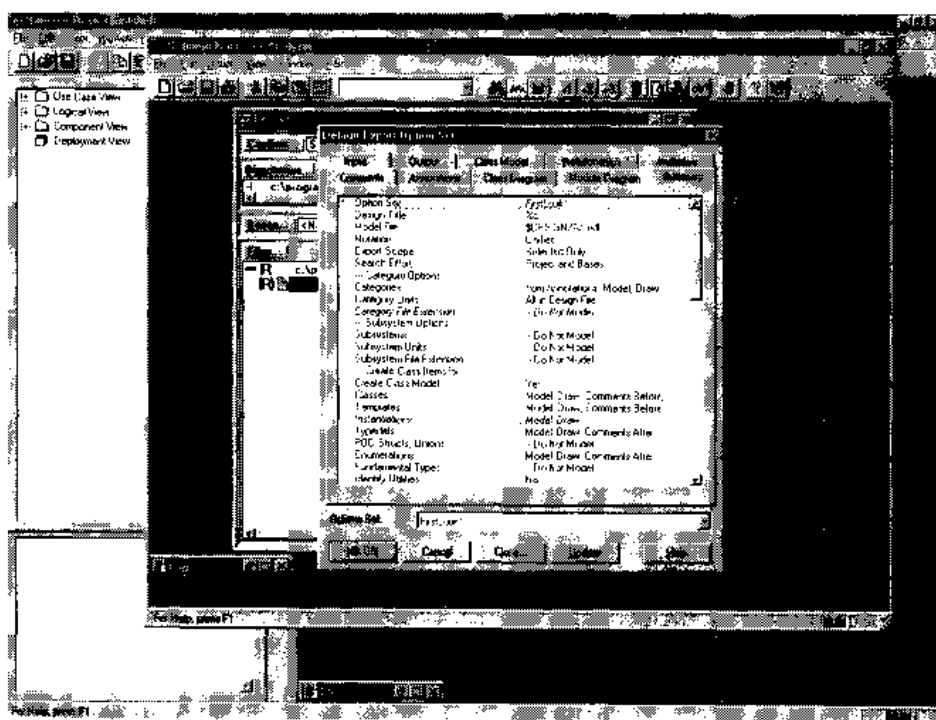


图20.19 浏览逆向转出工程代码小结

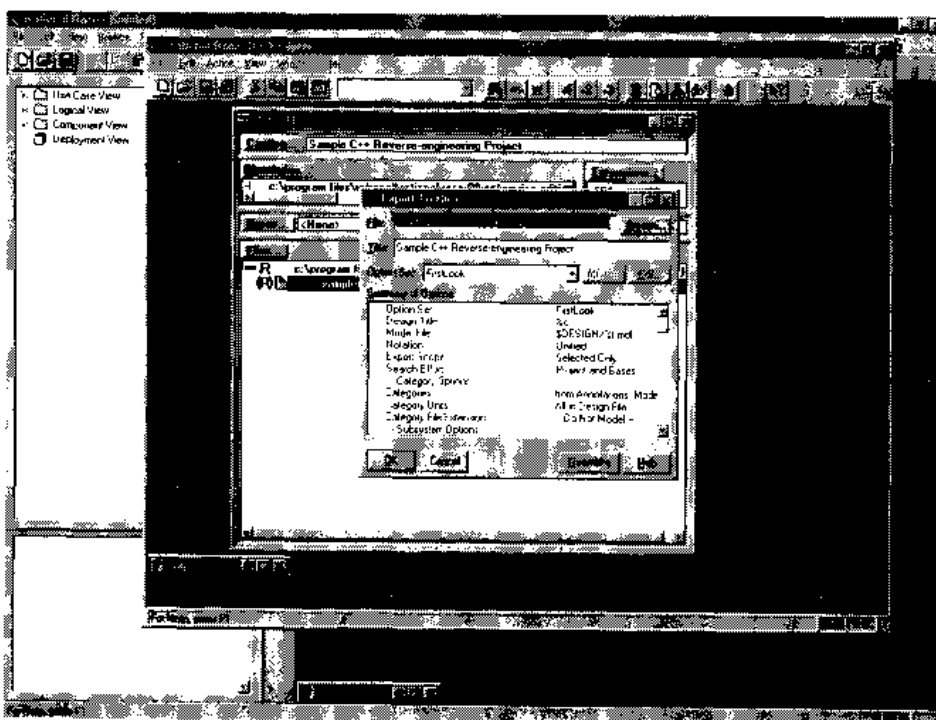


图20.20 将项目输出到Rose

第一步：开始逆向转出工程代码向导

Visual C++逆向转出工程代码是向导驱动的，向导会带你完成代码逆向转出工程代码所需的所有步骤。要打开向导，选择Tools>Visual C++>Update Model from Code。

出现Model Update Tool Welcome屏幕，如图20.21。

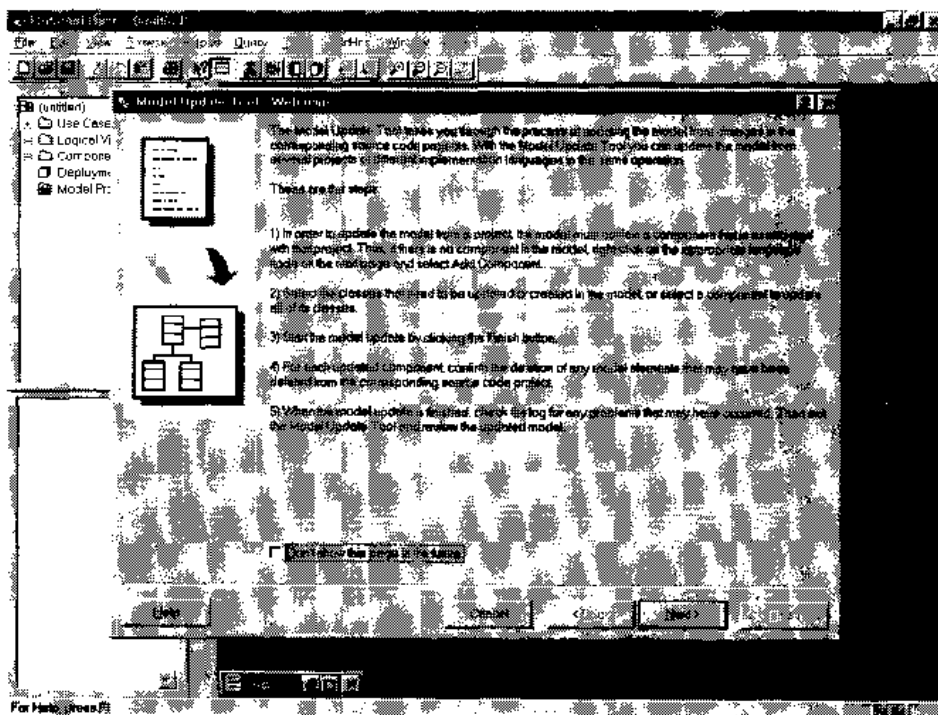


图20.21 Visual C++ Model Update Tool——Welcome屏幕

第二步：选择逆向转出工程代码项目

启动向导后，出现的下一个屏幕是Select Components和Classes窗口。Rose模型中必须存在组件并与一个VC++项目相关联。如果组件不存在，则可以生成组件，即右单击语言节点并选择Add Component。如果组件存在，则选择要逆向转出工程代码的组件或类。有些组件的旁边有问号，这是不完整的组件，不能逆向转出工程代码。

选择所有要逆向转出工程代码的组件或类后，单击Next按钮继续。

第三步：检查与完成

这一步可以浏览要逆向转出工程代码的组件或类。如果要进行改变，选择Back回到Select Components和Classes窗口，否则单击Finish按钮将所选组件和类逆向转出工程代码到Rose模型。

进行逆向转出工程代码时，代码中有而模型中没有的新类Rose模型中，代码中类的任何改变也反映到模型中。

逆向转出工程代码过程完成时，结果出现在小结页面中。逆向转出工程代码过程的错误和警告与逆向转出工程代码项目在一起显示。

从C++代码生成的模型元素

本节要介绍各段C++代码如何出现在生成的Rose模型中。具体地说，我们要介绍类、数据成员、成员函数、“朋友”声明和类模板。

逆向转出工程代码时，Analyzer使用代码和逆向转出工程代码选项屏幕中选择的信息。这些选项确定每个C++结构生成什么内容。

类

C++代码中的每个类表示为模型中的类。缺省情况下，类还出现在生成的Class框图中，并对类生成包。

下面介绍下列C++头文件逆向转出工程代码时生成的代码：`\reverse\SampleClass.h`

```
class SampleClass {  
    int a, b, c;  
protected:  
    int d, e, f;  
};
```

在逆向转出工程代码过程中，Analyzer分析这个文件，并确定生成哪些类、属性、操作和其他模型元素。图20.22显示了上述代码生成的模型。

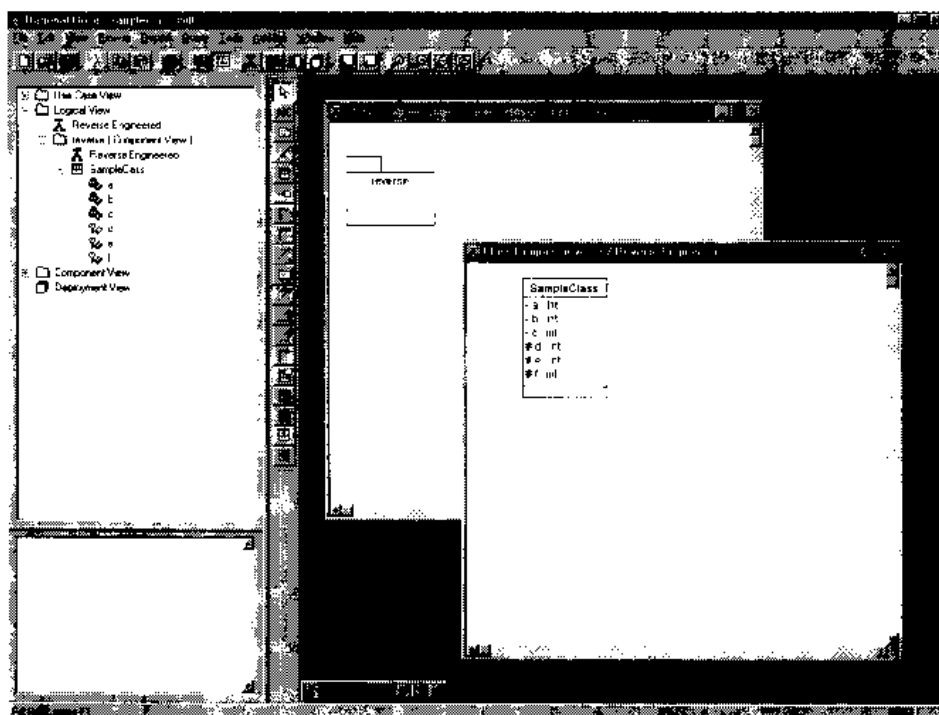


图20.22 类的逆向转出工程代码

模型中生成的元素包括：

- 类本身，在逻辑视图中生成。
- 类的属性和操作，将在下节介绍。
- 类所在的Logical视图包，包名为源代码所在的目录名。本例中sampleclass.cpp文件在目录reverse中。
- 显示逆向转出工程代码类的Class框图。缺省情况下，这个框图的名称为Reverse Engineered。要改变框图名，可以改变C++ Analyzer中的逆向转出工程代码属性。
- 显示逆向转出工程代码类包的Package框图。这个框图缺省名称为Reverse Engineered。要改变框图名，可以改变C++ Analyzer中的逆向转出工程代码属性。

数据成员

类中的每个数据成员表示为生成模型中的必要性或关系。如果数据成员的数据类型为模型中另一个类，则显示为累积关系。如果数据类型不表示模型中另一个类，则显示为属性。

```
#include Class_B.h

class Class_A {
    int x;
    Class_B the_class_b;
};

class Class_B {
    int a, b, c;
};
```

图20.23显示了上述两个类逆向转出工程代码时生成的模型。可以看出，B类中的数据成员a、b、c是属性，因为模型中没有int类；但A类中的数据成员_Class_b是累积按值关系，如果A类中的数据成员_Class_b为指向B类的指针，则生成的关系为按引用关系。

如果代码中将数据成员标为静态，则在生成的Rose模型中也标为静态。

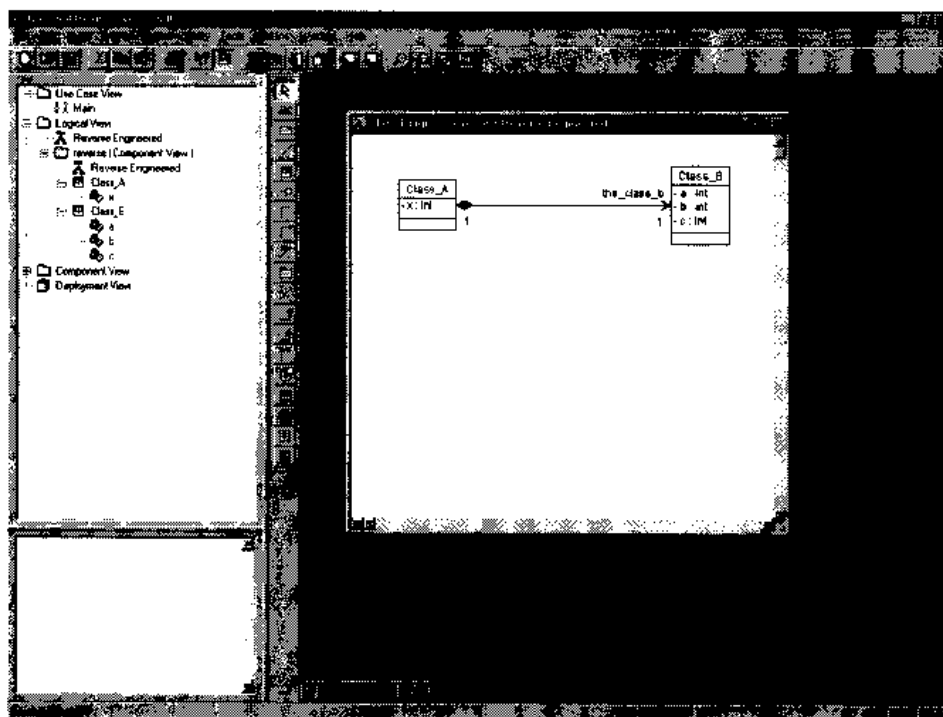


图20.23 数据成员逆向转出工程代码

成员函数

成员函数在Rose模型中生成操作。如果函数的变元或返回类型用模型中另一个类作为数据类型，则和数据成员一样生成相应关系。

下例中, Analyzer生成SampleClass类, 包括保护操作foo。图20.24显示了从这个类生成的模型。

```
class SampleClass {
    int a, b, c;
protected:
    int d, e, f;
    int foo (int parameter1) { };
};
```

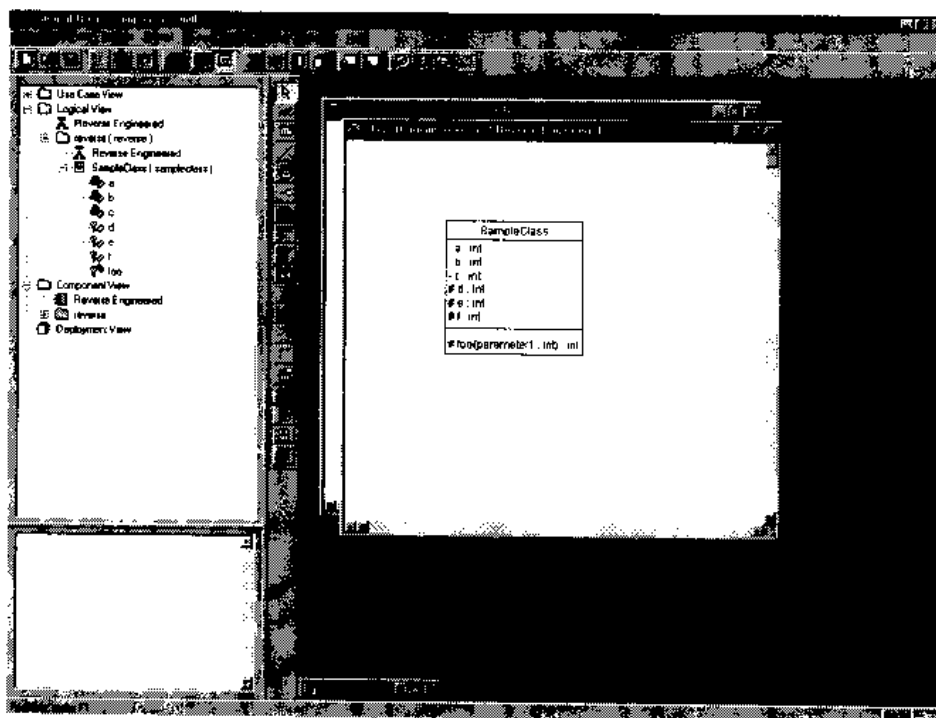


图20.24 成员函数逆向转出工程代码

类“朋友”声明

C++代码中的“朋友”声明生成Rose模型中的依赖性关系。

下例中, Class_B和Class_A之间生成支持朋友声明的依赖性关系。图20.25显示了这两个类生成的模型。

```
#include Class_B.h

class Class_A {
    int x;
    friend Class_B;
};

class Class_B {
    int a, b, c;
};
```

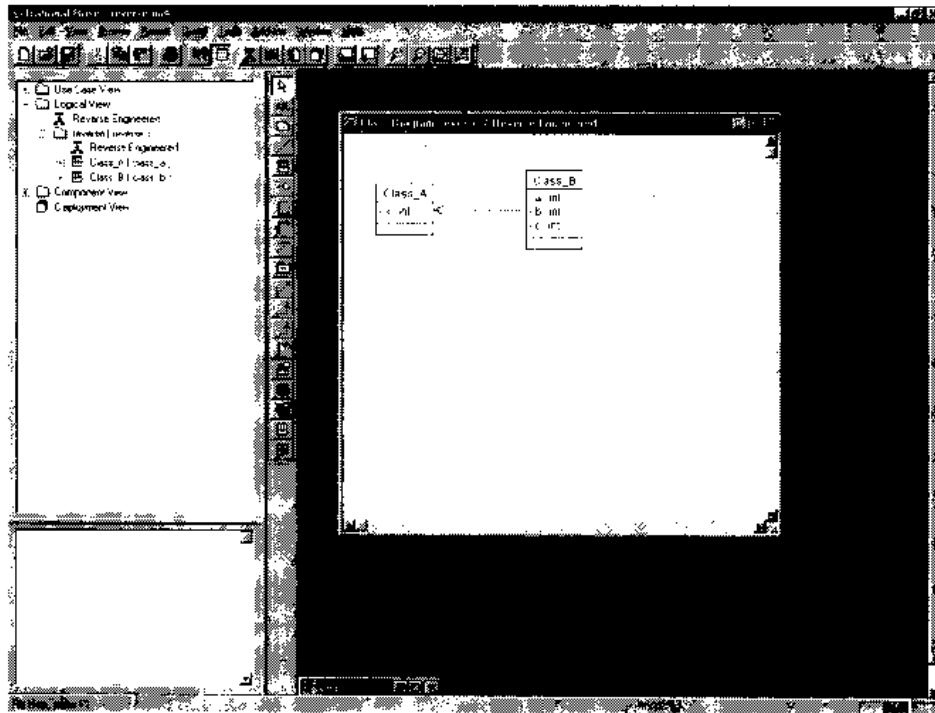


图20.25 朋友声明逆向转出工程代码

类模板

C++代码中的类模板显示为Rose模型中的参数化类。下面是Stack类的样本代码。在Rose模型中，其显示为参数化类，与Item类有依赖性关系。

```
template<class Item> class Stack {
    Item *theitem;
    int size;
public:
    Stack (int size);
    ~Stack();
    void push (const Item&);
    Item& Pop();
};
```

图20.26显示了生成的Rose模型。可以看出，Stack类与Item类有依赖性关系。

派生类

C++代码中的继承关系用Rose中的一般化关系构造。Analyzer逆向转出工程代码父类和子类，并将其放在Class框图中。

让我们看看下列类生成的模型：

```
class Parentclass {
    int a, b, c;
};
```

```
#include Parentclass.h

class Childclass : public Parentclass
{
    int x, y, z;
};
```

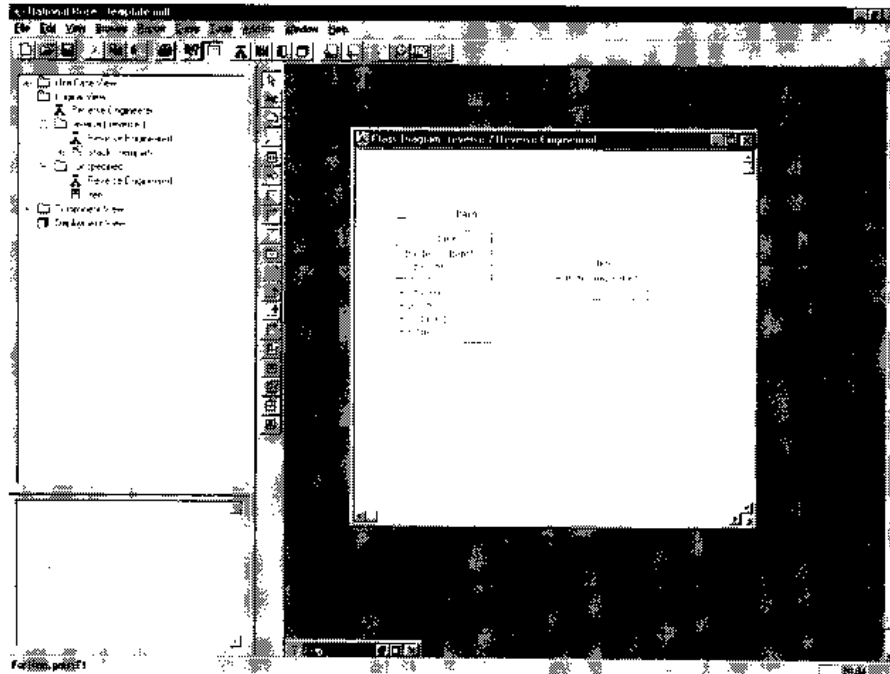


图20.26 类模板逆向转出工程代码

图20.27显示了上述类生成的Rose模型。可以看出，C++代码中的继承关系用Rose中的一般化关系构造。

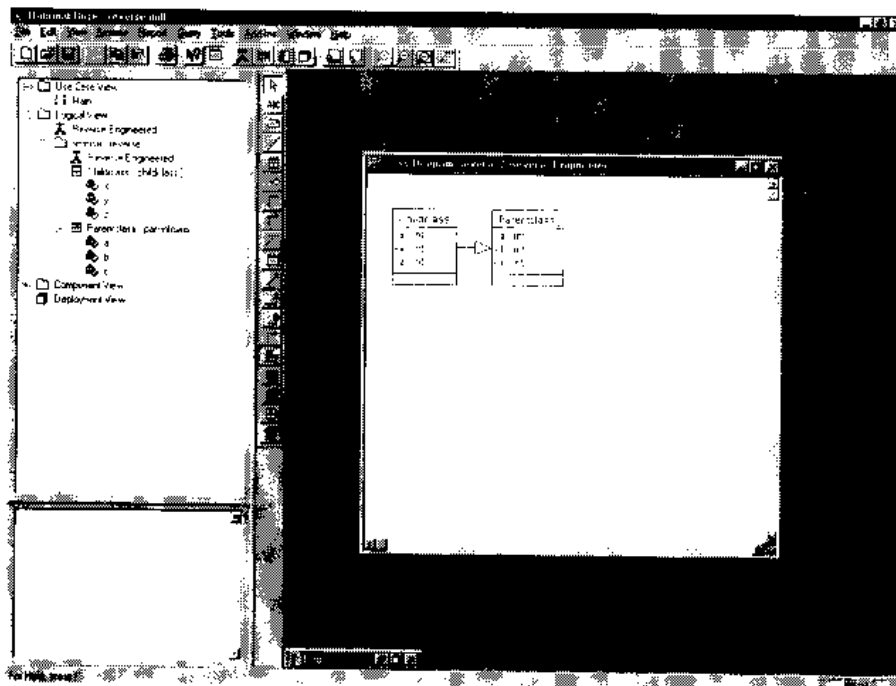


图20.27 派生类逆向转出工程代码

小结

本章介绍了C++代码如何逆向转出工程代码为Rose模型。按照本章介绍的步骤，可以检查现有C++应用程序的结构。

将文件逆向转出工程代码为Rose模型后，可以对模型进行必要的改变。例如，可以重新构造工程或重新构造现有应用程序。逆向转出工程代码功能可以显示现有代码的结构。然后可以对模型进行必要的改变并通过Rose正向工程功能重新生成代码。由于Rose支持双向工程，因此不会覆盖已经放在代码中的工作成果。

第21章 Java逆向转出工程代码

- 在Rose 98中输入Java开发工具库
- 选择逆向转出工程代码文件
- 将Java代码逆向转出工程代码为Rose模型
- 映射Java结构与Rose模型

本章要介绍将Java代码逆向转出工程代码为Rose模型。首先介绍逆向转出工程代码的步骤，然后介绍生成的Rose元素。

进行逆向转出工程代码之前，先要在Rose 98中输入Java开发工具库（JDK），向模型提供基础Java类和数据类型。在Rose 98中，要装入JDK，选择Tools>Java>Import JDK 1.1X。在Rose 98i中，可以用Framework向导输入JDK框架。完成这个步骤后，Java类即可在Logical视图的Java包中提供。图21.1显示了输入JDK之后的浏览器。

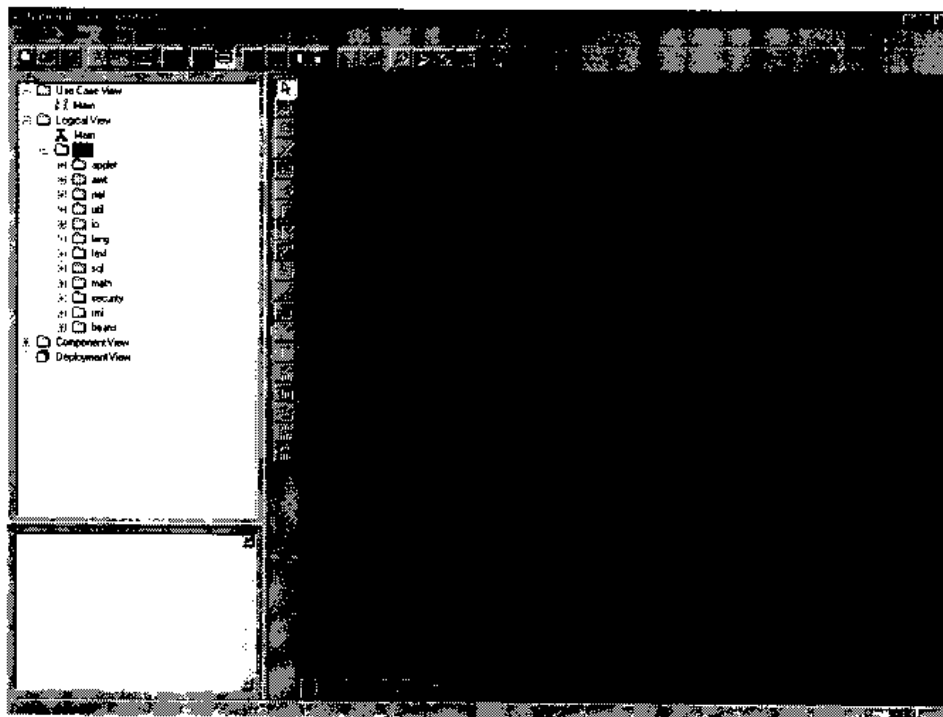


图21.1 在Rose 98中输入JDK

逆向转出工程代码步骤

本节介绍Java代码的逆向转出工程代码步骤。代码中的信息直接输入当前打开的Rose模型中。

1. 首先，选择Tools>Java>Reverse Engineer Java，Rose打开Java Reverse Engineering窗口，如图21.2。利用这个窗口，可以选择要逆向转出工程代码的文件。如果出现消

息说CLASSPATH环境变量未设置,则还要先生成环境变量CLASSPATH,并将这个变量设置为Java类文件所在的目录。



图21.2 Java Reverse Engineering窗口

2. 从目录树结构中,选择要逆向转出工程代码的文件所在目录。改变目录时,窗口右上角的列表框中出现可用文件,如图21.2。
3. 选择要逆向转出工程代码的文件并按Add按钮。这时文件出现在下方窗口中,如图21.3。

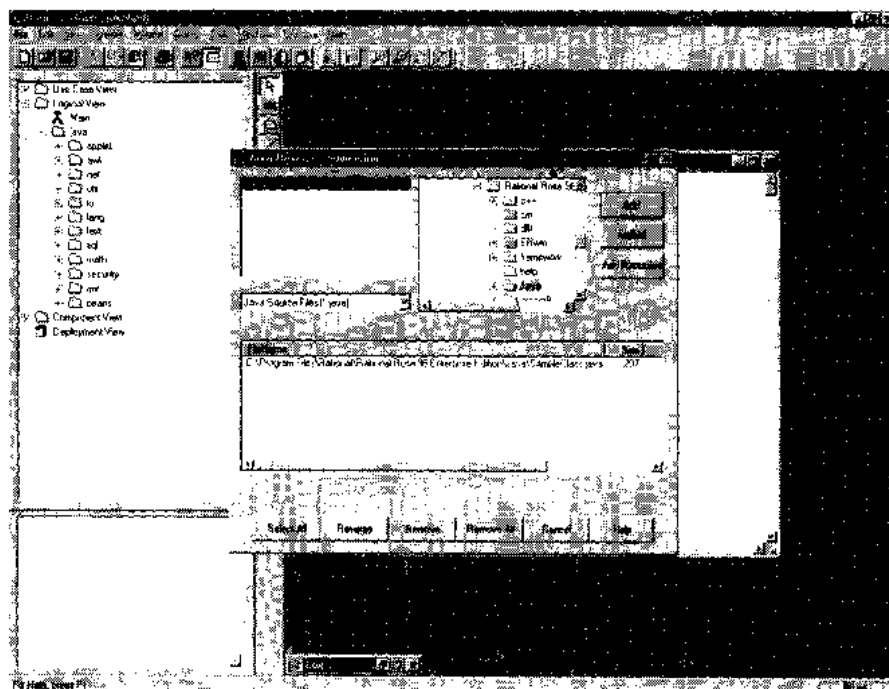


图21.3 设置要逆向转出工程代码的文件

4. 将需要的所有文件加进窗口底部后，单击选择要逆向转出工程代码的文件（或按Select All按钮选择所有文件），并单击Reverse按钮。如果遇到错误，则将错误写入日志窗口。

从Java代码生成的模型元素

本节介绍从Java的各种结构生成的模型元素。例如，我们将介绍JAVA文件中的类如何映射Rose模型中的类。

但首先要看看Java结构与Rose模型间的映射。表21.1列出了Rose中生成的各种Java代码类型。

表21.1 Java结构与Rose模型间的映射

Java结构	Rose说明
Class（类）	Class（类）
Method（方法）	Operation（操作）
Attribute（属性）	Attributes和relationships（属性与关系）
Interface（接口）	Class with interface stereotype（版型为接口的类）
Package（包）	Package（包）
Inheritance Relationship（继承关系）	Generalization（一般化）

类

下面先介绍类。Java代码中每个类表示为模型中的类。类在Logical视图包中生成，对应组件在Component视图包中生成。下面看看这个类逆向转出工程代码时生成的模型：

```
// Source file: JavaClasses/SampleClass.java

package JavaClasses;

public class SampleClass {
    private int attribute1;

    SampleClass() {
    }

    public int operation1() {
    }
}
```

本例中，Rose生成SampleClass类，有一个属性，两个操作。图21.4显示了这个类生成的Rose模型。

模型中生成的元素包括：

- 类本身，在逻辑视图中生成。
- 类的属性和操作，将在下节介绍。
- 类所在的Logical视图包，包名为源代码所在的目录名。
- 类的Component视图中的组件。
- 组件的Component视图包。

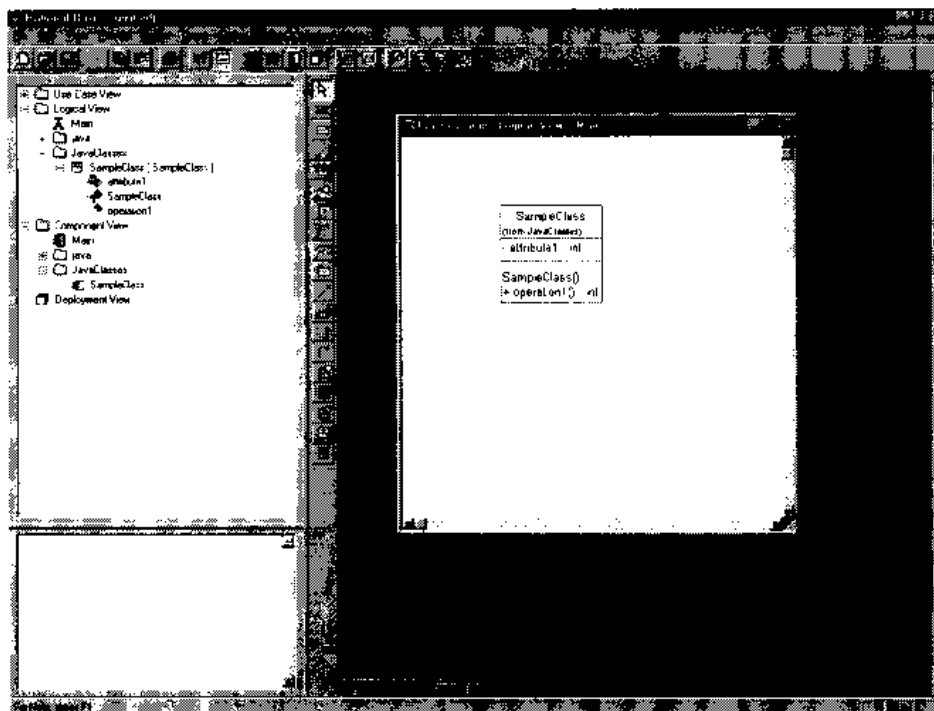


图21.4 类逆向转出工程代码

属性

源代码中每个属性表示为生成模型中的属性或关系。如果数据成员的数据类型为模型中的另一类，则它显示为关联关系。如果数据类型不表示为模型中另一个类，则数据成员显示为属性。

下面举两个属性的例子。一个生成属性，一个生成关联关系。

// Source file: JavaClasses/Class_A.java

package JavaClasses;

import java.javaclasses.class_B;

```
public class Class_A {
    private int attribute1;
    public Class_B the_Class_B;
```

```
    Class_A() {
    }
}
```

// Source file: JavaClasses/Class_B.java

package JavaClasses;

```
public class Class_B {
    private int attribute2;
```

```
    Class_B() {
    }
}
```

```
}
```


从图21.5可以看出，两个类都在Rose模型中生成。A类中attribute1属性在生成的类中生成属性，并具有可见性和数据类型信息。但_Class_B属性生成Class_A和Class_B之间的一对一关系。Java代码逆向转出工程代码时，Rose检查倍增性并将其加进关系中。下面再看看主例，这时让倍增性大于一。注意这两个类生成的模型：

```
// Source file: JavaClasses/Class_A.java
```

```
package JavaClasses;
```

```
import java.javaClasses.class_B;
```

```
public class Class_A {  
    private int attribute1;  
    public Class_B the_Class_B[];  
  
    Class_A() {  
    }  
}
```

```
// Source file: JavaClasses/Class_B.java
```

```
package JavaClasses;
```

```
public class Class_B {  
    private int attribute2;  
  
    Class_B() {  
    }  
}
```

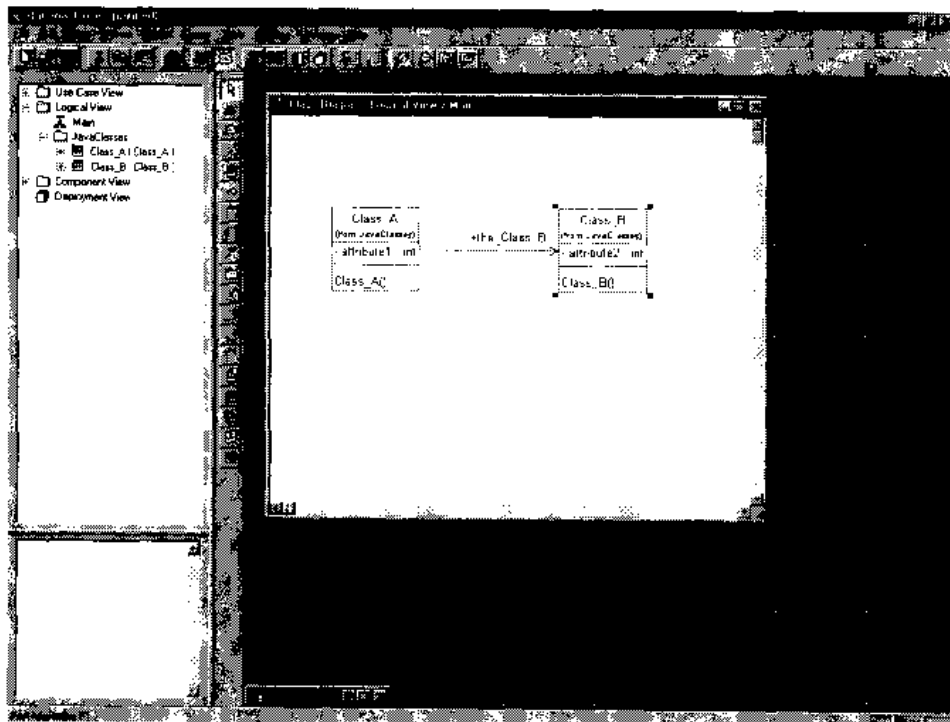


图21.5 属性和关联关系逆向转出工程代码

图21.6显示了这两个类生成的模型。

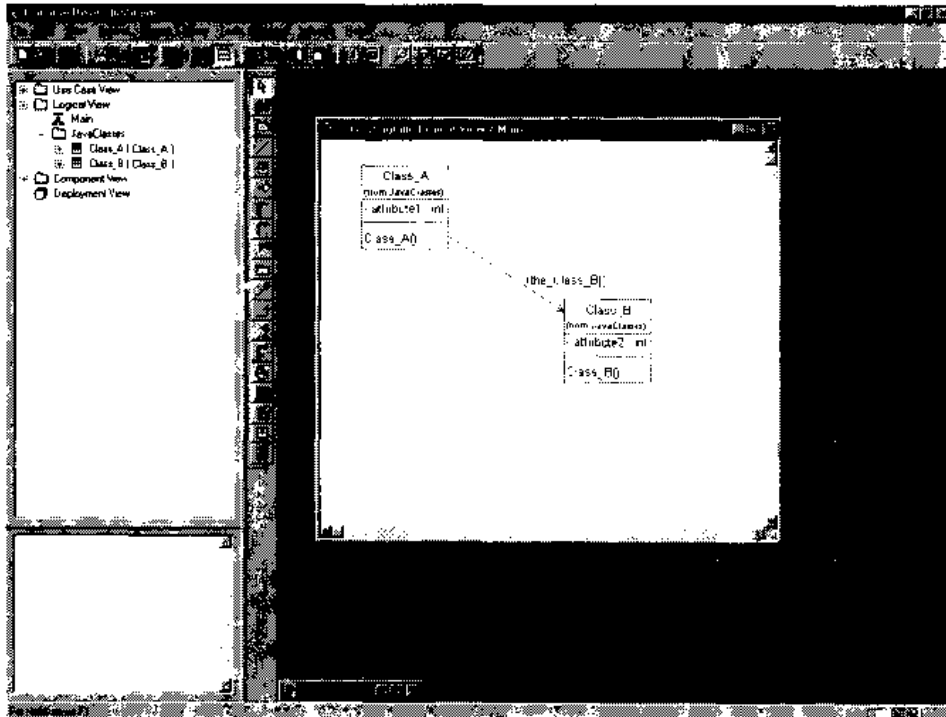


图21.6 倍增性大于一时关系的逆向转出工程代码

方法

就像代码中的属性一样，代码中的每个方法也包括进Rose模型中，Rose检查方法名、可见性、参数、参数数据类型和返回值并将其包括进生成的模型中。下面介绍下列Java类生成的模型：

```
// Source file: JavaClasses/SampleClass.java

package JavaClasses;

public class SampleClass {
    private int attribute1;
    private int attribute2;

    SampleClass() {
    }

    public int operation1(int parameter1) {
    }
}
```

从图21.7可以看出，构造器和operation1方法都逆向转出工程代码到Rose模型中。

接口

Java接口以版型为接口的类放进Rose模型中。例如，看看下列接口生成的模型：

```
// Source file: SampleInterface.java  
  
public interface SampleInterface {  
  
    public int operation1();  
  
}
```

本例中，接口SampleInterface成为Rose模型中的类。图21.8显示了生成的类。

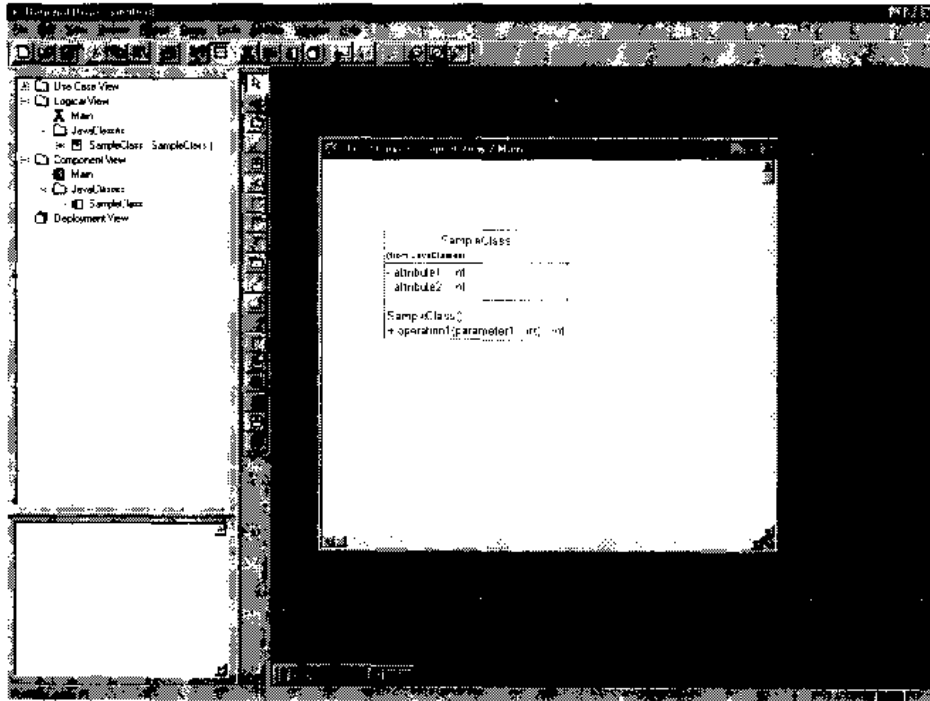


图21.7 Java构造器与方法逆向转出工程代码

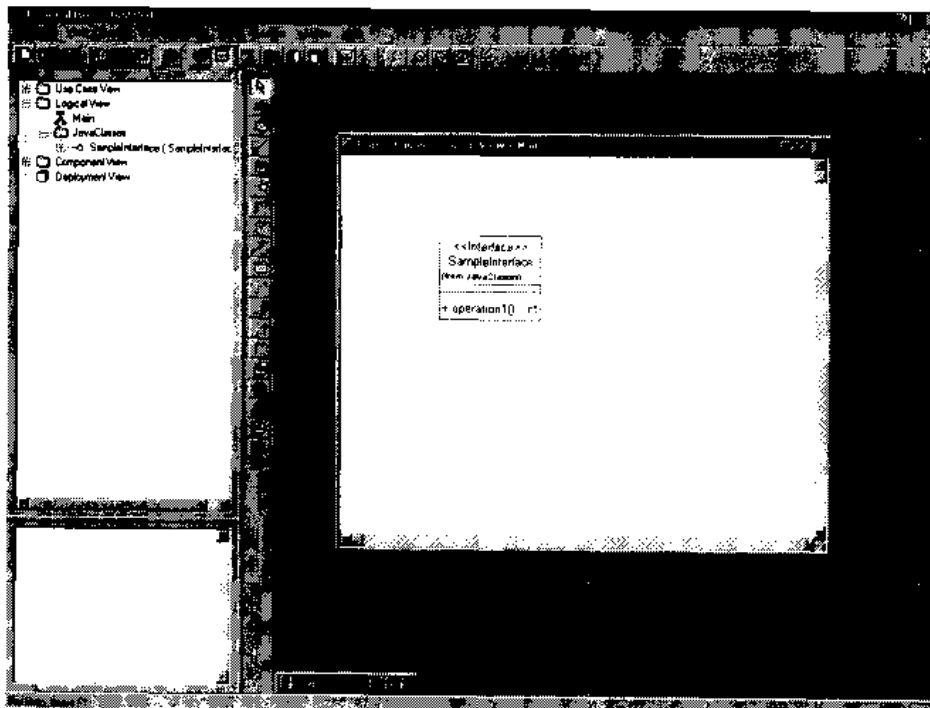


图21.8 接口逆向转出工程代码

另一个例子，我们将接口和实现该接口的类进行逆向转出工程代码。看看下面两个类生成的模型：

```
// Source file: JavaClasses/SampleInterface.java

package JavaClasses;

public interface SampleInterface {

    public int operation1();
}

// Source file: JavaClasses/SampleClass.java

package JavaClasses;

import java.javaclasses.sampleinterface;

public class SampleClass implements SampleInterface{

    SampleClass() {
    }

    public int operation1() {
    }
}
```

这两个类生成的模型如图21.9。

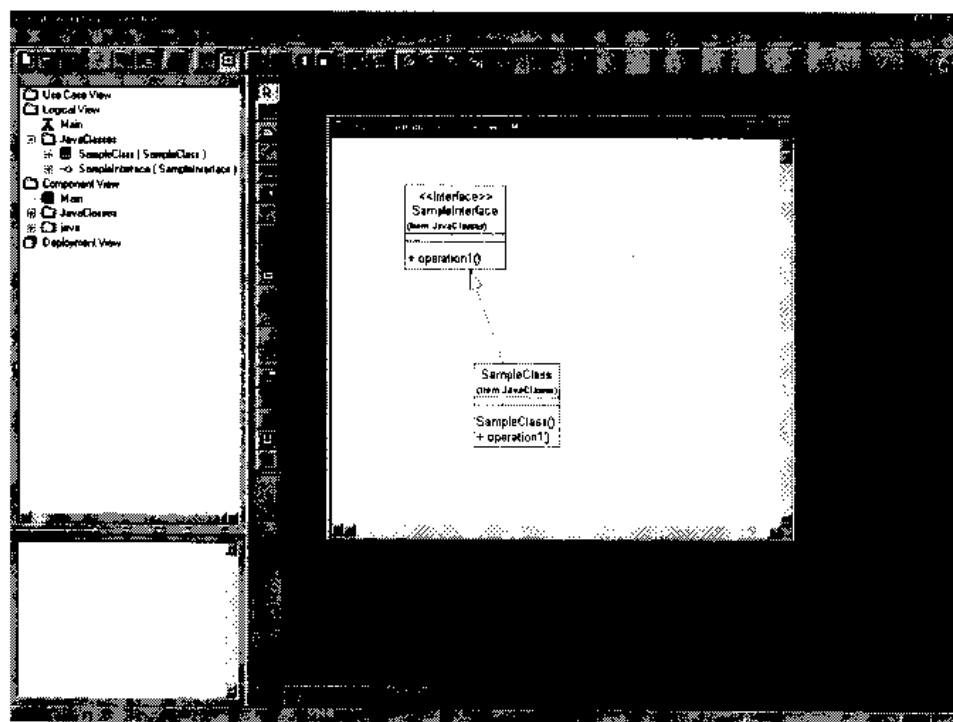


图21.9 接口与类逆向转出工程代码

可以看出，Rose在类和接口之间放上实现关系。由于Rose的Java部分了解接口的概念，还可以在作出必要改变后重新生成这些类。

继承关系

Java源代码中的继承关系在Rose中用一般化关系建模如下:

```
// Source file: JavaClasses/Parentclass.java

package JavaClasses;

public class Parentclass {
    private int attribute1;

    Parentclass() {
    }
}

// Source file: JavaClasses/Childclass.java

package JavaClasses;

import java.javaclasses.parentclass;

public class Childclass extends Parentclass {
    private int attribute2;

    Childclass() {
    }
}
```

图21.10显示了上述代码生成的Rose模型。可以看出,父类和子类都在Rose模型中生成。此外,两个类之间增加了一般化关系。

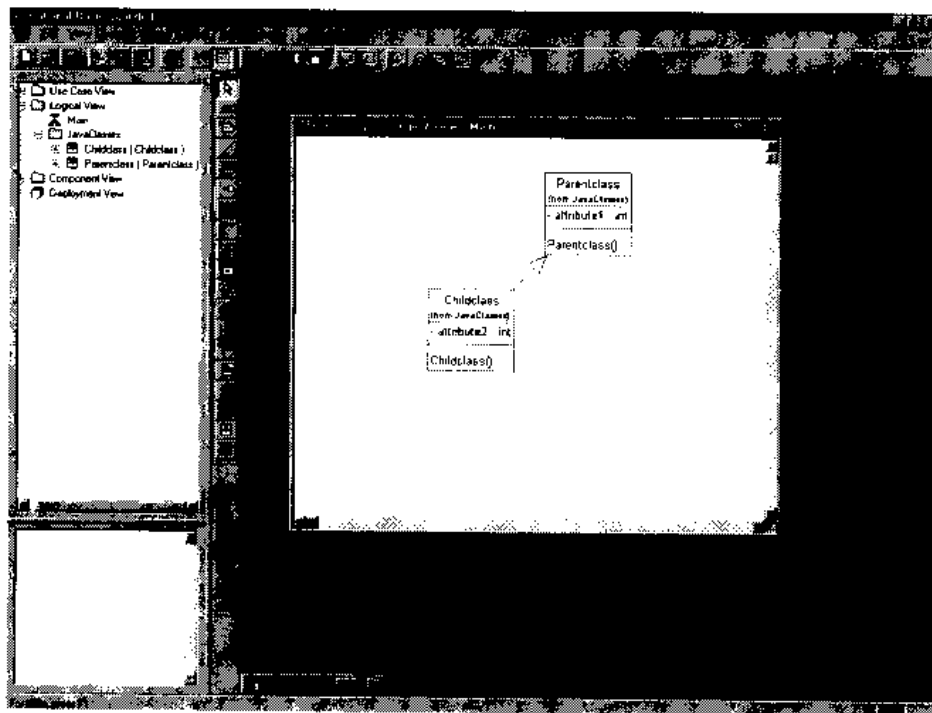


图21.10 继承关系逆向转出工程代码

小结

本章介绍Java代码的逆向转出工程代码过程并检查Java向UML的映射。通过逆向转出工程代码现有代码，可以详细了解现有系统的结构。Rose的正向和逆向转出工程代码功能可以保证模型和代码保持一致性。

下一章介绍如何逆向转出工程代码现有Visual Basic代码。Rose对Visual Basic提供向导驱动的逆向转出工程代码过程，能用现有VB应用程序生成模型。生成模型后，可以检查VB应用程序结构，进行模型修改或重新从Rose生成VB代码。

第22章 Visual Basic逆向转出工程代码

- 用Framework向导逆向转出工程Visual Basic代码
- 选择逆向转出工程代码的类和组件
- 了解每种Visual Basic结构在Rose模型中的表示

利用Rose可以将Visual Basic项目中的信息逆向转出工程代码为UML模型。然后可以浏览现有系统结构，修改模型和重新生成代码（如果需要）。

在逆向转出工程代码过程中，Rose利用Visual Basic类、属性、操作和其他结构的信息。逆向转出工程代码过程完成后，可以得到：

- Classes（类）
- Class diagrams（框图）
- Packages（包）
- Components（组件）
- Attributes（属性）
- Operations（操作）
- Association relationships（关联关系）
- Aggregation relationships（累积关系）
- Generalization relationships（一般化关系）

Visual Basic逆向转出工程代码过程是向导驱动的。Rose的Model Update工具能使逆向转出工程代码过程充分自动化。本章介绍这个向导，并介绍如何逆向转出工程代码Visual Basic代码。

逆向转出工程代码步骤

要逆向转出工程代码Visual Basic代码，首先用Framework向导输入相应版本的Visual Basic类型（如标号和文本）到Rose模型中。要启动向导，选择Tools►Visual Basic►Update Model from Code，出现图22.1所示的欢迎屏幕。

然后选择要用的Visual Basic类和组件。如果模型中没有Visual Basic组件，可以生成组件，即选择图22.2所示的Visual Basic按钮。如果生成新组件，则Visual Basic项目会逆向转出工程代码到这个组件。如果选择现有组件，则向导与模型同步化，反映代码中的改变。

选择逆向转出工程代码的类和组件后，向导出现图22.3所示的结束屏幕。这个屏幕列出要更新的类和组件，提供开始操作前的最后确认机会。单击Finish按钮即可逆向转出工程代码这个项目。

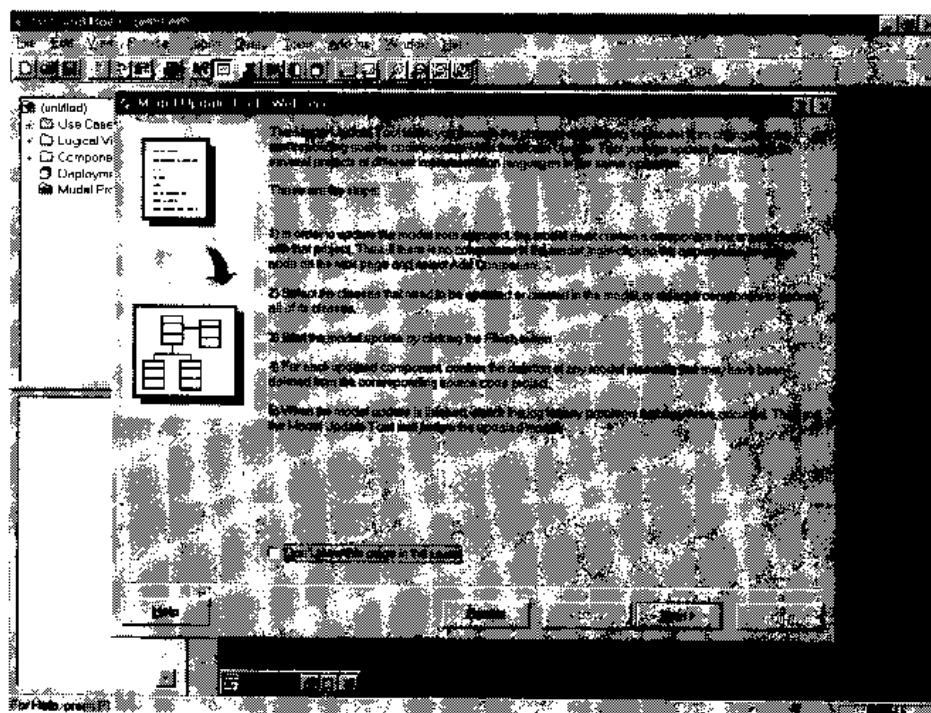


图22.1 Model Update向导欢迎屏幕

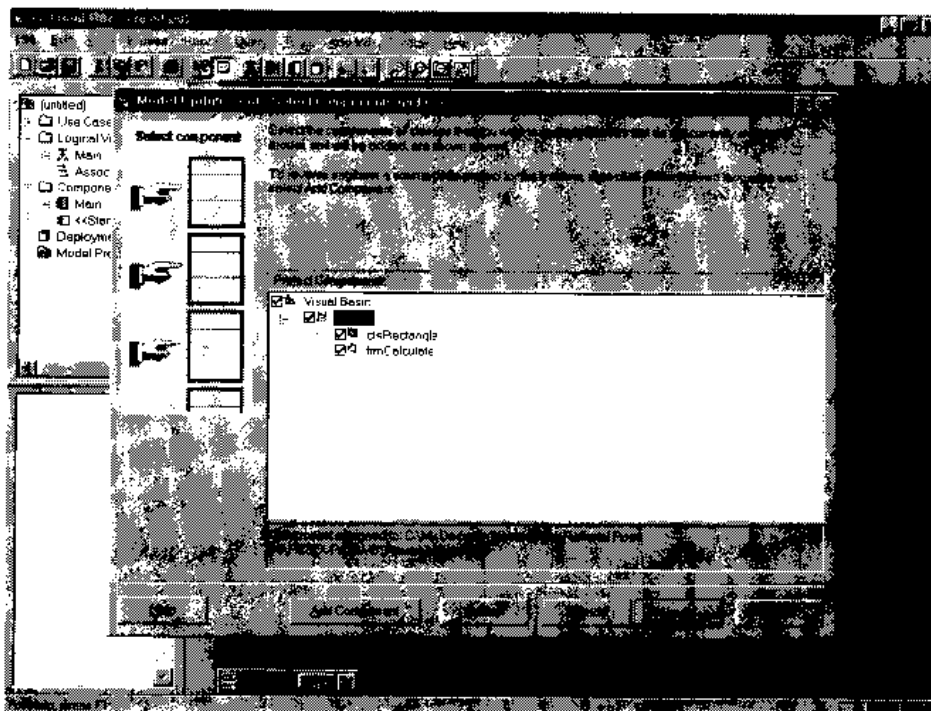


图22.2 选择逆向转出工程代码的类和组件

逆向转出工程代码完成后，出现图22.4所示的小结屏幕，显示逆向转出工程代码的组件记录。单击Close按钮关闭小结屏幕，结束逆向转出工程代码过程。



图22.3 Model Update向导结束屏幕

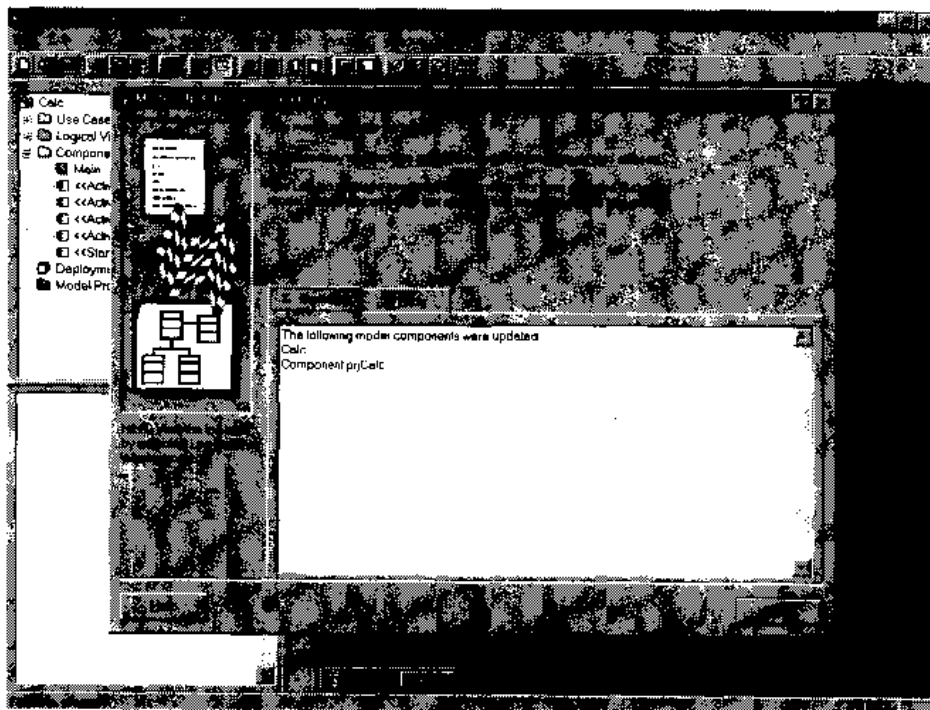


图22.4 逆向转出工程代码模型更新工具小结屏幕

从Visual Basic代码生成的模型元素

前面介绍了Visual Basic代码逆向转出工程代码为Rational Rose的步骤。下面要介绍在Rose模型中的表示。

大多数Visual Basic元素逆向转出工程代码为版型类。例如，窗体逆向转出工程代码为版型为Form的类，其他Visual Basic结构逆向转出工程代码为接口。窗体上的控件被逆向转出工程代码为接口CommandButton、TextBox等等。本节介绍一些例子。

窗体和控件

窗体逆向转出工程代码为版型为Form的类窗体上的所有变量表示为类的属性。变量数据类型、缺省值和可见性的信息也放进Rose模型中。

图22.5显示了Visual Basic窗体frmCalculate。

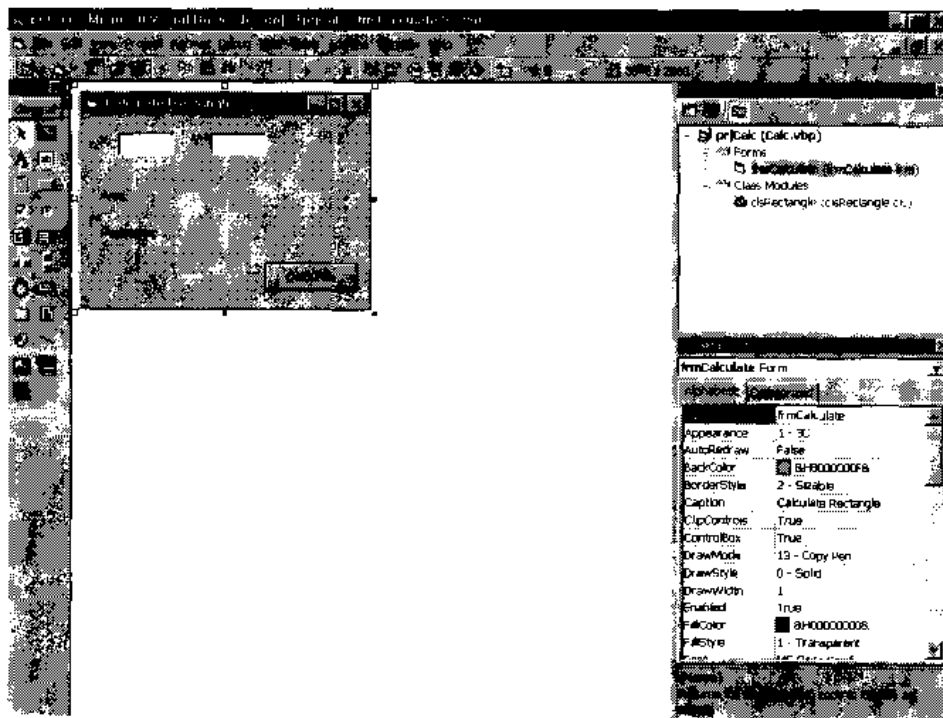


图22.5 frmCalculate Visual Basic窗体

图22.6显示了这个窗体逆向转出工程代码为Rose的结果。可以看出，窗体的所有方法成为Rose的操作。窗体上所有变量和包含的控件显示为关联关系。本例中，窗体有一个变量M_rectangle，类型为clsRectangle。代码逆向转出工程代码时，变量造型为frmCalculate类与clsRectangle类间的关系。关联名与变量名相同，这里为M_rectangle。

窗体上的控件也逆向转出工程代码。上例中有九个控件：四个标签，用于X、Y、Area和Parameter字段；四个文本框，用于X、Y、Area和Parameter字段；还有一个命令钮。每种控件（命令钮、标签等）都显示为框图上的接口。要显示放在窗体上的控件，Rose用累积关系，如图22.7所示。累积关系中的作用名取窗体上的控件名。例如，命令钮在VB中的名称为cbCalculate，因此Rose中的作用名用cbCalculate。所有包含控件的累积关系版型为Contained-Control。

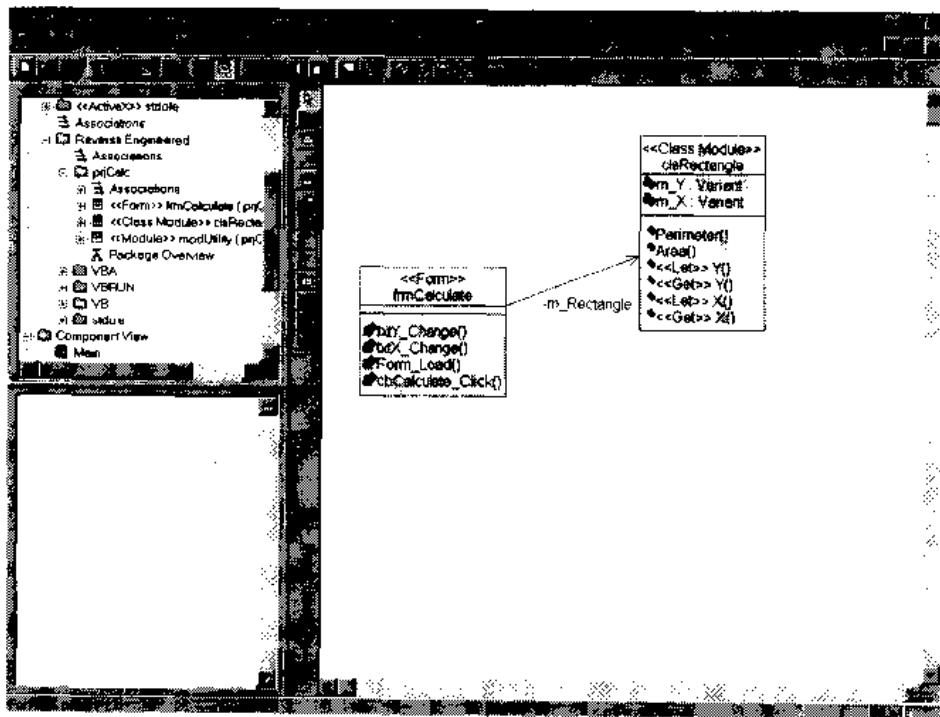


图22.6 Rose模型中的frmCalculate窗体

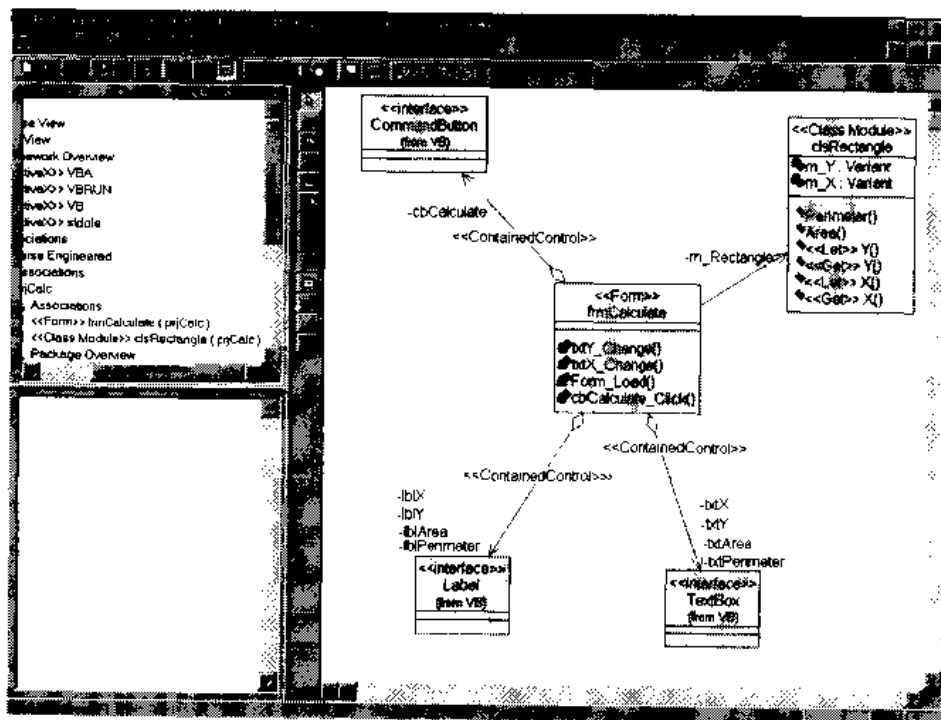


图22.7 Rose模型中窗体上的控件

类模块

逆向转出工程代码Visual Basic类模块时，它在Rose模型中表示成版型为Class Module的类。与类模块相关的方法和事件显示为版型操作。类模块中的任何变量显示为与其他类的关联关系。图22.8显示Visual Basic类模块clsRectangle。

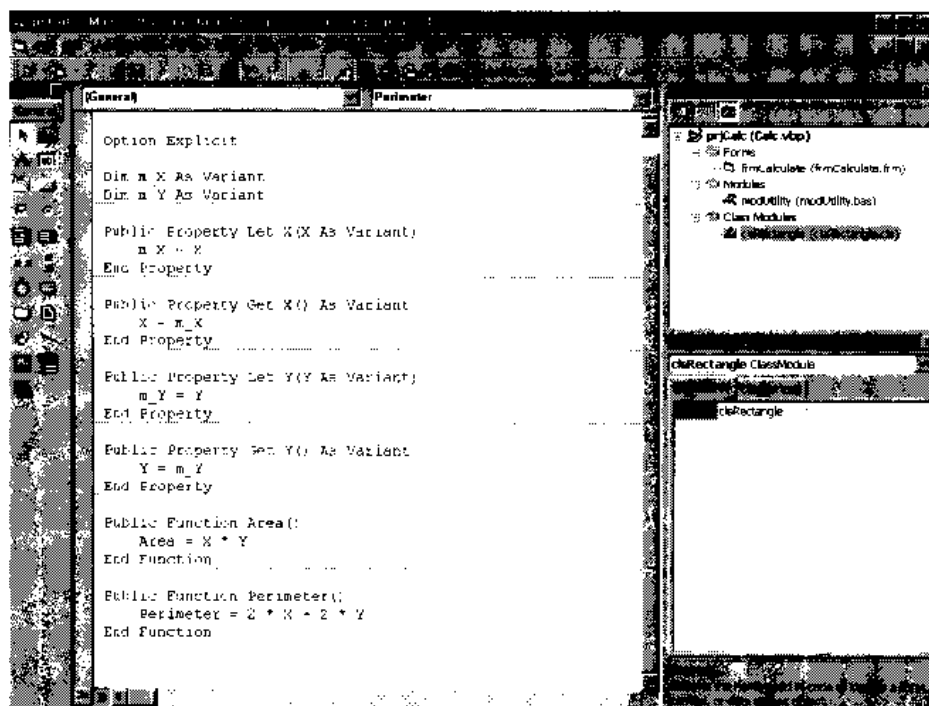


图22.8 样本Visual Basic类模块

这个类模块逆向转出工程代码时，关于其属性、操作和关系的信息也逆向转出工程代码。图22.9显示了Rose模型中的这个类。从本例可以看出，Rose用Set、Let和Get关键字指定相应操作的版型为Set、Let和Get。

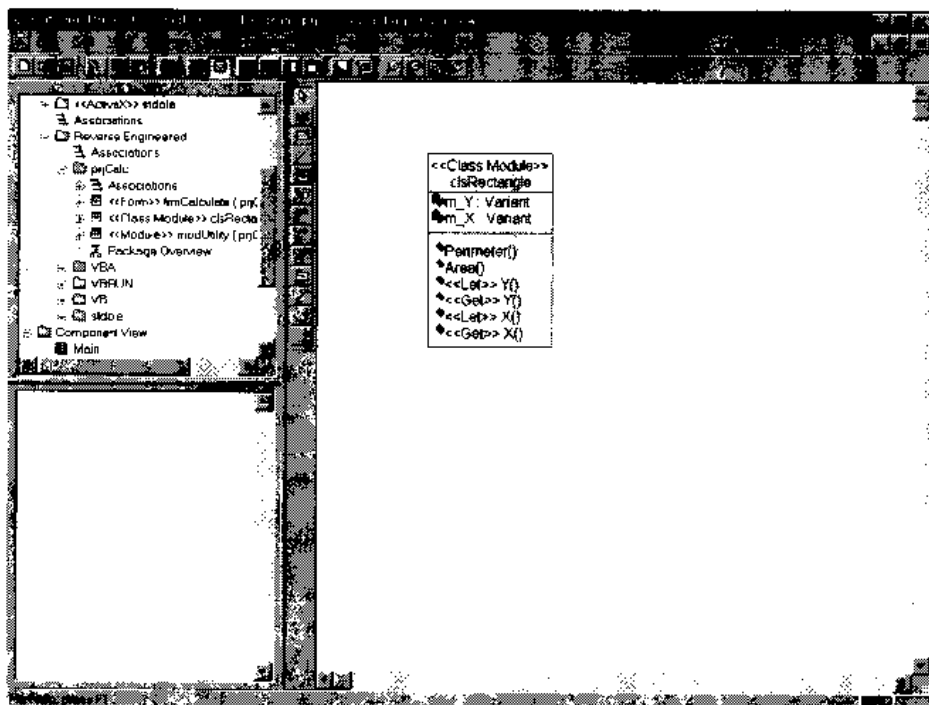


图22.9 Rose模型中的类模块

逆向转出工程代码时，Rose检查代码中模型元素前面的说明语句。例如，逆向转出工程代码一个属性时，Rose用属性声明前面的说明语句。逆向转出工程代码过程读取的说明语句放在相应模型元素的Documentation窗口。

模块

Rose中所有操作都应放在类中。因此，Visual Basic模块用Utility类输入Rose。Utility类包含相关操作的集合，可以放置全局变量和函数。

每个模块在Rose中表示为版型为Module的类。图22.10显示了逆向转出工程代码到Rose中的模块。

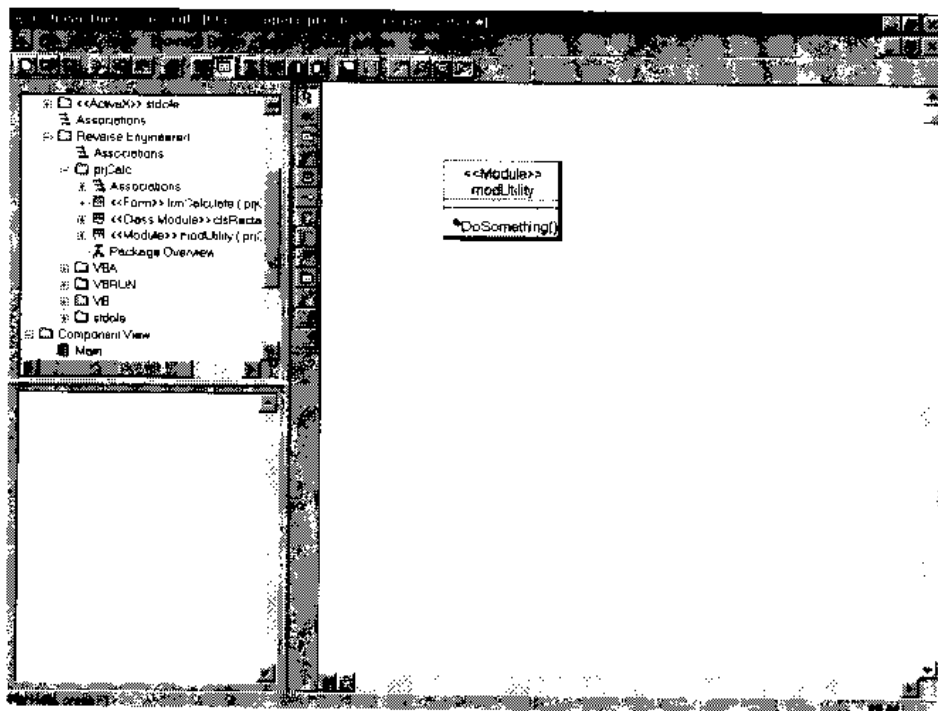


图22.10 Rose模块中的Visual Basic模块

实现语句

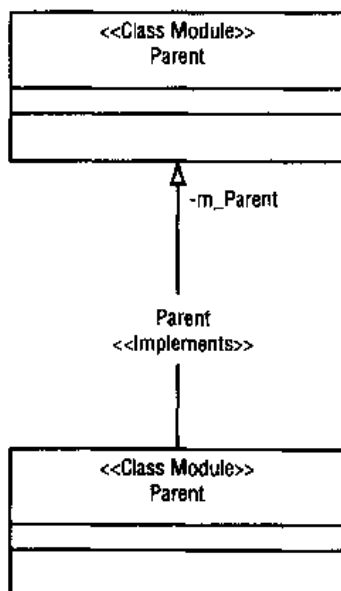
Visual Basic中的实现语句部分实现继承。因此，实现语句在模型中变成一般化。例如，子类声明实现下列父类：

```

Option Base 0
Option Explicit
    Implements Parent
Private mParentObject As New Parent

```

下列框图显示了Rose模型中的这个父子关系。一般化建模以显示实现语句的用法。



小结

本章介绍了将Visual Basic项目逆向转出工程代码为Rose模型所需的步骤。大多数对象直接变成Rose中的版型类。此外，Rose捕获代码中的变量、函数和关系信息。然后可以改变模型并将所作改变正向工程为Visual Basic项目。

下一章介绍PowerBuilder的逆向转出工程代码过程。和C++、Java与Visual Basic一样，Rose可以从现有PowerBuilder代码生成UML模型。每个窗口、数据窗口和其他PowerBuilder对象建模为类。下一章将详细介绍这个过程。

第23章 PowerBuilder逆向转出工程代码

- 选择逆向转出工程代码的类和文件
- 从各种PowerBuilder结构逆向转出工程代码Rose模型元素
- 逆向转出工程代码每个类的属性、操作和关系信息

PowerBuilder Link for Rational Rose具有代码生成和逆向转出工程代码功能。代码生成可以对新系统生成框架代码或对现有系统生成一些修改代码。逆向转出工程代码功能可以分析现有PowerBuilder系列结构。

插入件支持PowerBuilder 5、6和6.5代码的逆向转出工程代码。逆向转出工程代码完成后，可以得到：

- Classes（类）
- Class diagrams（框图）
- Packages（包）
- Components（组件）
- Attributes（属性）
- Operations（操作）
- Association relationships（关联关系）
- Aggregation relationships（累积关系）
- Generalization relationships（一般化关系）

本章介绍逆向转出工程代码PowerBuilder代码生成的模型。PowerBuilder插入件包括许多版型，每个PowerBuilder结构在Rose中有对应版型。表23.1列出了PowerBuilder版型和对应Rose版型。

表23.1 PowerBuilder版型

版型	适用模型元素	PowerBuilder类型
Application	类	Application对象
Menu	类	PowerBuilder菜单
Structure	类	PowerBuilder结构
System Class	类	PowerBuilder系统类
User Defined Object	类	User Object（用户定义对象）
Window	类	Window
Control	属性	PowerBuilder控件（button, text box等）
Field	属性	Structure element（结构元素）
Property	属性	PowerBuilder（对象属性）
Variable	属性	Instance variable（实例变量）
EventExtend	操作	Event extended from ancestor（扩展事件）
EventOverride	操作	Event that overrides ancestor（覆盖事件）

(续表)

版型	适用模型元素	PowerBuilder类型
EventOverrideInternal	操作	PowerBuilder中不显示的覆盖事件（如create和destroy）
Function	操作	PowerBuilder函数
Subroutine	操作	PowerBuilder子程序
UserEvent	操作	User-defined event（用户定义事件）

利用这些版型，可以在Rose中建模各种PowerBuilder结构。表23.2列出了PowerBuilder实体及其在Rose模型中的表示。

表23.2 PowerBuilder向UML映射

PowerBuilder元素	Rose元素
Class（类）	版型类
PBL	组件（Subprogram Specification或Main Program）
Application Object（应用程序对象）	版型为<<Application>>的类
Window（窗口）	版型为<<Window>>的类
Nonvisual User Object（不可见用户对象）	版型为<<User Defined Object>>的类
Menu（菜单）	版型为<<Menu>>的类
控件（button, datawindow control等）	版型为<<Control>>的类
Datawindow（数据窗口）	版型为<<Data Window>>的类
Global Function（全局函数）	版型为<<Function>>的类
Query（查询）	版型为<<Query>>的类
Structure（结构）	版型为<<Structure>>的类
Shared Variable（共享变量）	静态标志设置为True的属性或关联
Non-shared Variable（非共享变量）	静态标志设置为True的属性或关联
Window或User Object Function	版型为<<Function>>的属性
Event（事件）	版型为<<EventOverride>>或 <<EventOverrideInternal>>的操作
User Event（用户事件）	版型为<<EventUser>>的操作

逆向转出工程代码步骤

许多PowerBuilder元素（如窗口和非可见用户对象）可以在Rose中生成并正向工程到PowerBuilder中。但有些PowerBuilder结构（如数据窗口和菜单）在PowerBuilder中更容易生成，然后再逆向转出工程代码到Rose中。利用正向和逆向转出工程代码特性，可以保证模型和代码保持一致。

本节介绍PowerBuilder代码逆向转出工程代码步骤。利用向导可以选择要逆向转出工程代码的PowerBuilder库文件（.PBC）和要放进Rose中的具体PowerBuilder类。

首先，选择Tools►PowerBuilder►Reverse Engineer，打开Rose PowerBuilder Link对话框，如图23.1。

用这个对话框，可以选择逆向转出工程代码的PBL文件和对象。单击Browse按钮选择.PBL文件。选择.PBL文件后，该文件中的PowerBuilder类出现在PowerBuilder Analyzer窗口中，如图23.2。

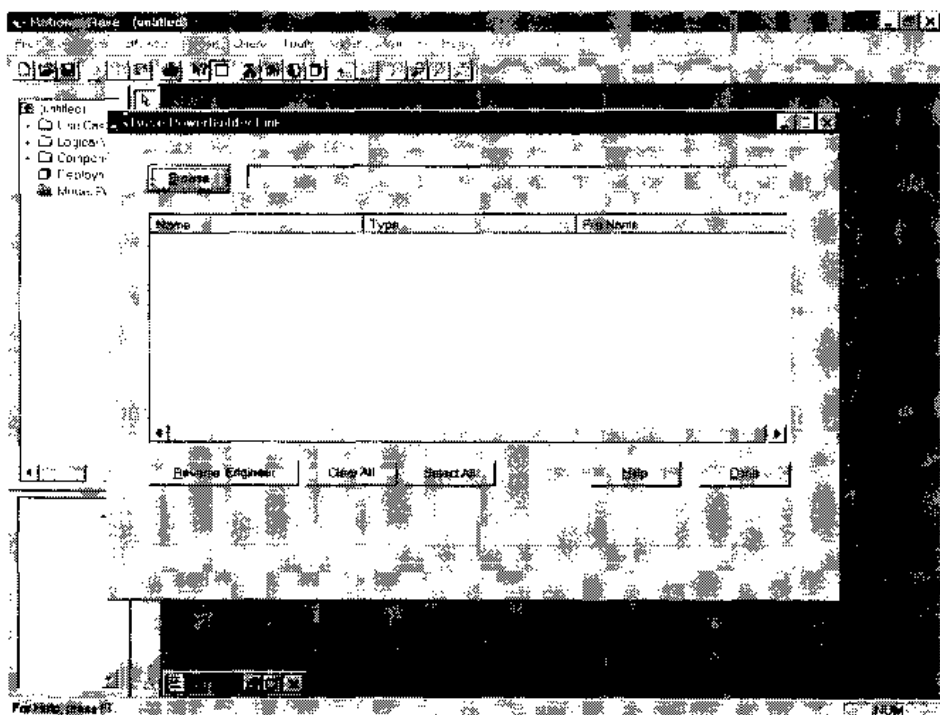


图23.1 PowerBuilder Analyzer窗口

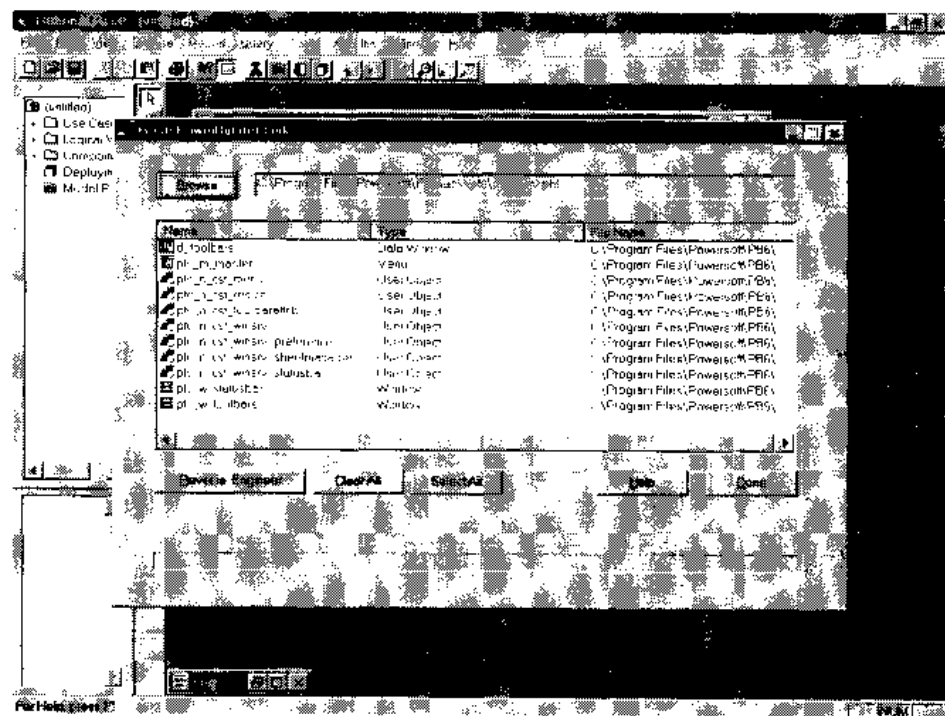


图23.2 PowerBuilder对象在Rose Power Builder Link窗口中

再次单击**Browse**按钮，继续选择PBL文件，直到**Select All**窗口中显示了要逆向转出工程代码的所有类。

在列表框中选择要逆向转出工程代码的类，或按Select All按钮选择所有类。单击Analyze将类逆向转出工程代码。如果出现错误，则这些错误写入日志窗口。

从PowerBuilder代码生成的模型元素

本节介绍从各种PowerBuilder结构生成的Rose模型元素，包括窗口、数据窗口和非可见用户对象。

大多数PowerBuilder元素逆向转出工程代码为版型类，见表23.1列出的PowerBuilder版型。除了类以外，Rose还逆向转出工程代码每个类的属性、操作和关系信息。

应用程序对象

PowerBuilder应用程序对象在Rose中表示为版型为<<Application>>的类。Rose用PowerBuilder应用程序对象名作为模型中的Application类名。

每个应用程序对象的实例变量出现在类中，版型为<<Variable>>。例如，事务对象sqlca和sqlsa显示为版型属性。每个应用程序对象的函数和事件显示为版型操作。图23.3显示了应用程序对象生成的Rose类。

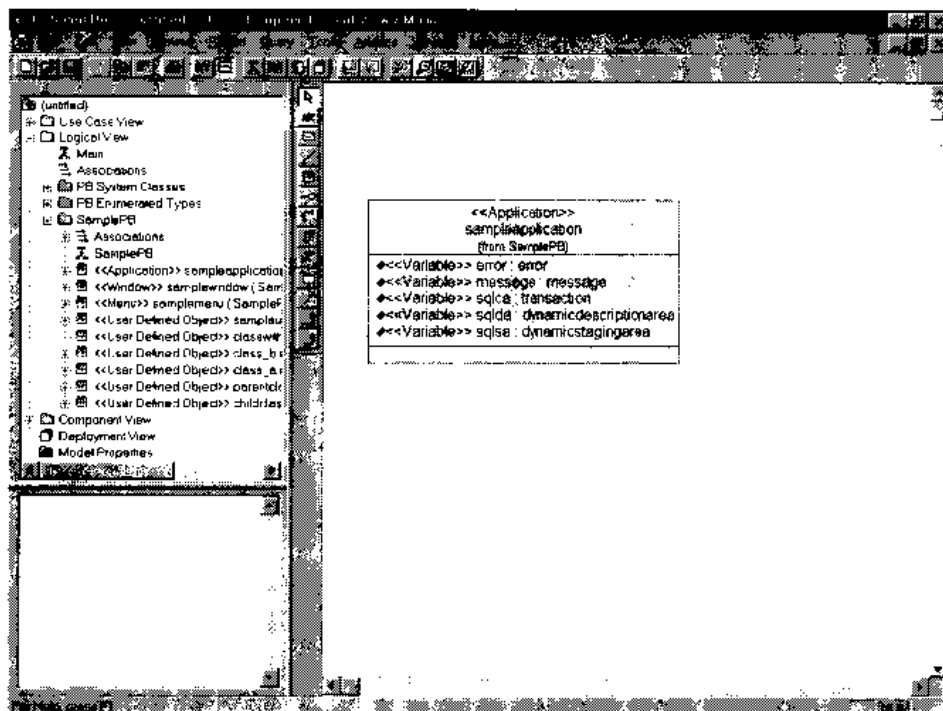


图23.3 逆向转出工程代码应用程序对象

典型的PowerBuilder应用程序对象有五个属性：sqlca、sqllda、sqlsa、错误属性和消息属性。每个属性都逆向转出工程代码为公开属性。应用程序对象的生成和删除函数被逆向转出工程代码为公开操作。这两个函数的前几行在文档窗口中显示。

非可见用户对象

非可见用户对象和应用程序对象一样，显示为Rose中的版型类。非可见用户对象的版型为<<User Defined Objec>>。任何非可见用户对象的用户对象函数和事件表示为Rose中

的版型操作。例如，构造器、生成和删除方法都表示为操作。逆向转出工程代码操作时，Rose将操作的前几行在文档窗口中显示。浏览模型时，可以通过文档窗口看到这个操作的用途。

用户对象的实例变量表示为版型属性。PowerBuilder将逆向转出工程代码属性名、可见性和数据类型的信息。实例变量的版型为<<Variable>>。

除了属性和操作外，Rose还包括非可见用户对象参与的关系信息。例如，如果对象从另一个非可见用户对象继承，则它们之间生成一般化关系。

图23.4显示了从PowerBuilder逆向转出工程代码非可见用户对象生成的Rose类。

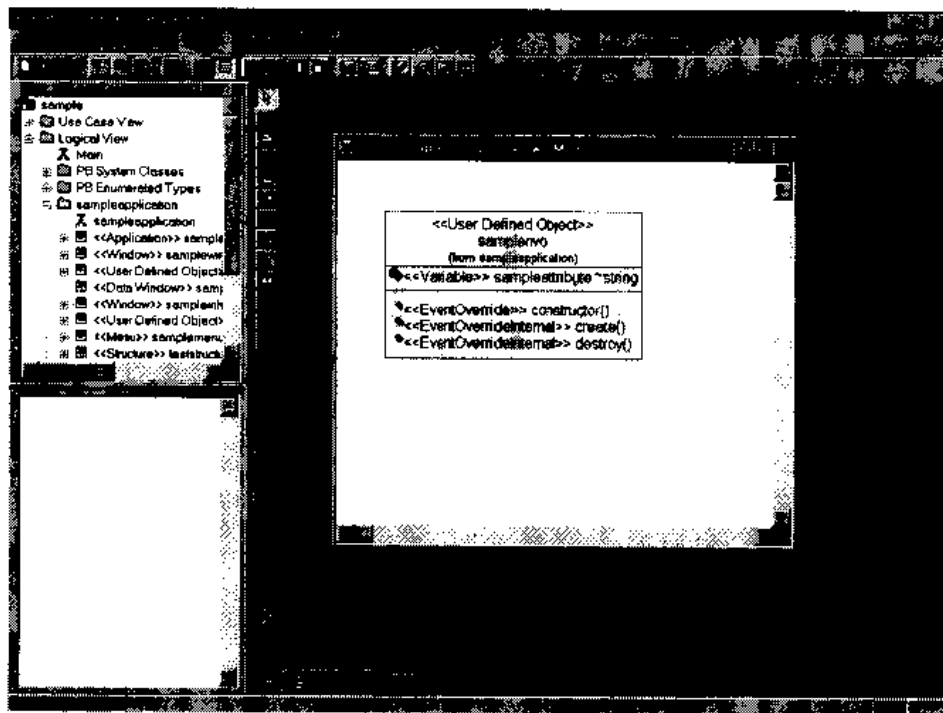


图23.4 非可见用户对象逆向转出工程代码

窗口

PowerBuilder窗口在Rose中显示为版型为<<Window>>的类。正向工程时，Rose用<<Window>>版型在PowerBuilder中生成窗口。

窗口中每个实例变量生成版型为<<Variable>>的属性。Rose包括属性名称、数据类型和可见性。

每个窗口属性逆向转出工程代码为版型为<<Property>>的属性。例如，PowerBuilder中的Height、Width、X位置、Y位置和Title窗口属性在Rose中显示为属性，并包括缺省值、数据类型和可见性。

窗口事件显示为版型为<<EventOverride>>或<<EventOverrideInternal>>的操作。窗口用户事件出现为版型为<<UserEvent>>的操作。模型中还包括操作的参数、参数数据类型和返回值。操作前几行出现在文档窗口中，如图23.5。

图23.6显示了逆向转出工程代码一个类生成的所有属性和操作。可以看出，所有实例变量、窗口函数和事件都列在Rose中，还有缺省值、数据类型和可见性信息。

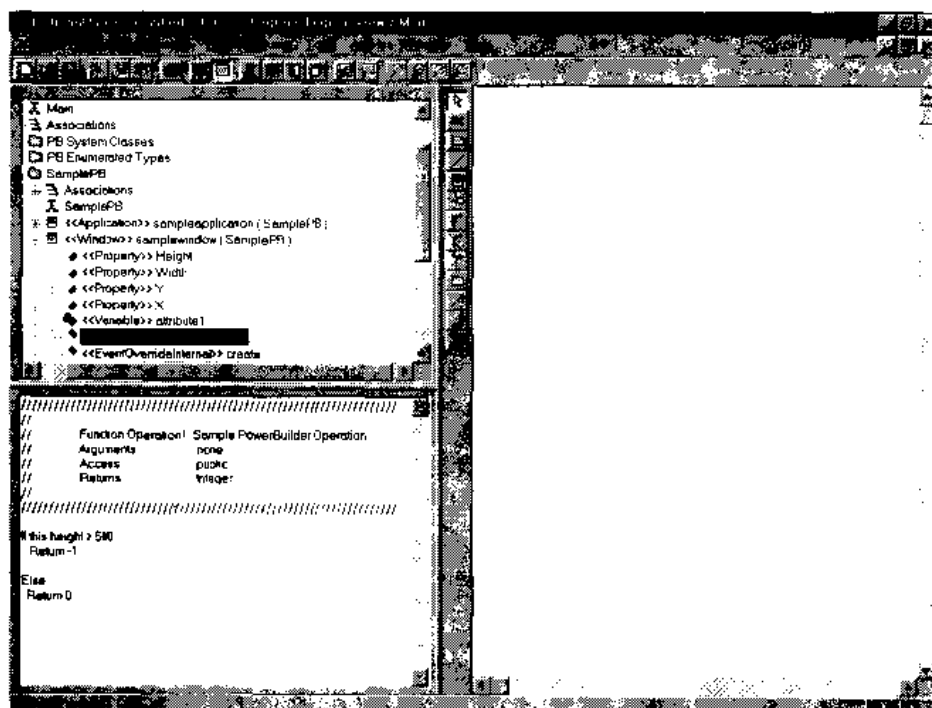


图23.5 操作的文档窗口

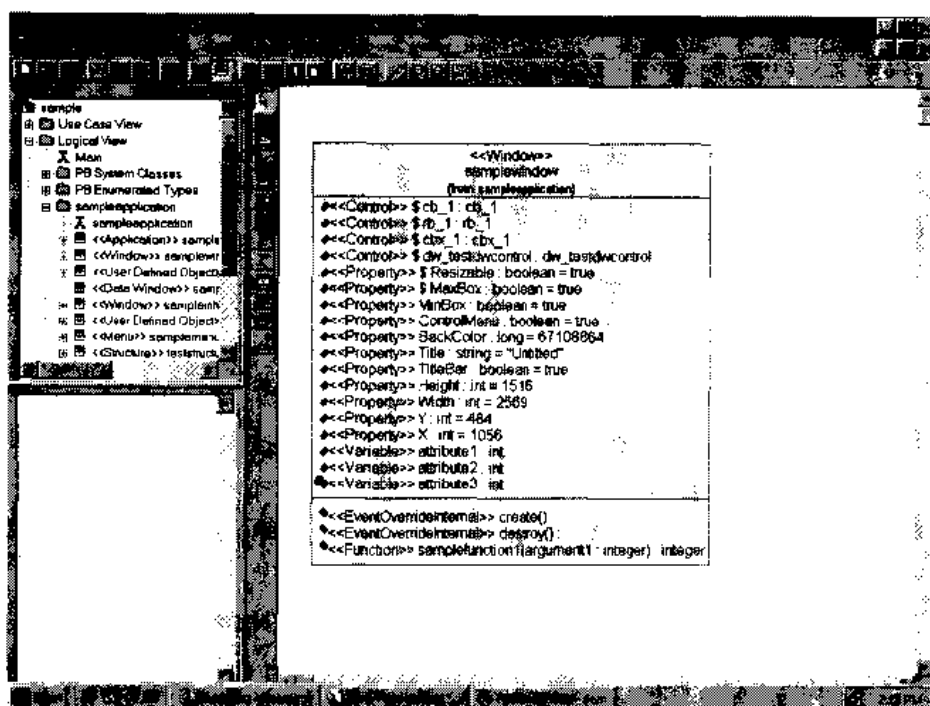


图23.6 窗口逆向转出工程代码

窗口中所放的任何控件也显示为属性。例如，如果窗口中有命令按钮cb_1，则它有个类型为cb_1的属性\$cb_1。cb_1类也出现在Class框图中，显示为版型为<<Control>>的类，具有对应于命令按钮属性的属性。例如，它有Text、TextSize、Height、Width、X、Y和TabOrder属

性。每个属性都有缺省值、数据类型和可见性设置。图23.7显示了刚刚介绍的窗口，上面放上控件。可以看出，Rose在窗口和控件的连接间采用累积关系。

属性

PowerBuilder类中声明的实例变量在Rose模型中显示为属性和操作。简单数据类型（如int和boolean）表示为属性。复杂数据类型（如命令钮）显示为属性和与类的关系。例如，命令钮显示为属性cb_1，并显示与Command Button类的关联。

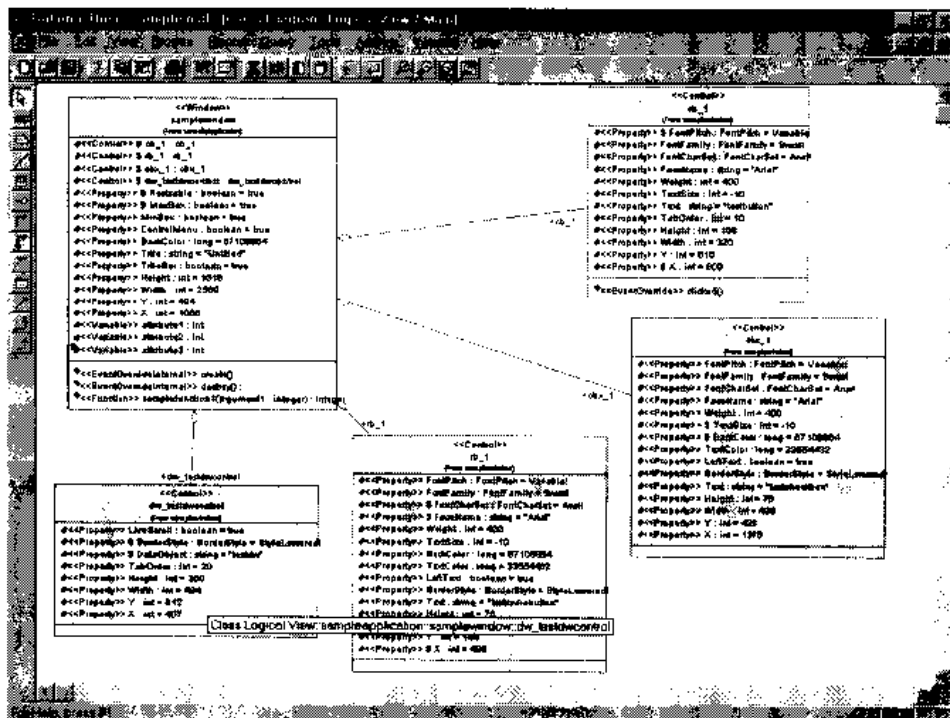


图23.7 窗口和控件的连接间采用累积关系

逆向转出工程代码过程生成属性及其缺省值、数据类型和可见性。共享变量在Rose中显示为静态属性，在属性名前面标上\$。

图23.8显示逆向转出工程代码非可见用户对象的实例变量时生成的属性。

操作

和属性一样，操作也可以从PowerBuilder代码逆向转出工程代码。PowerBuilder类中的每个对象函数和事件表示为Rose中的操作。例如，PowerBuilder窗口的函数create和destroy都逆向转出工程代码到Rose中。

逆向转出工程代码时，Rose放进操作名、参数、参数数据类型和返回值、可见性的信息。所有这些信息均在操作中自动生成。此外，Rose将操作的前几行放进文档窗口。然后可以用文档窗口迅速浏览操作的作用。

图23.9显示了从非可见用户对象逆向转出工程代码的操作。

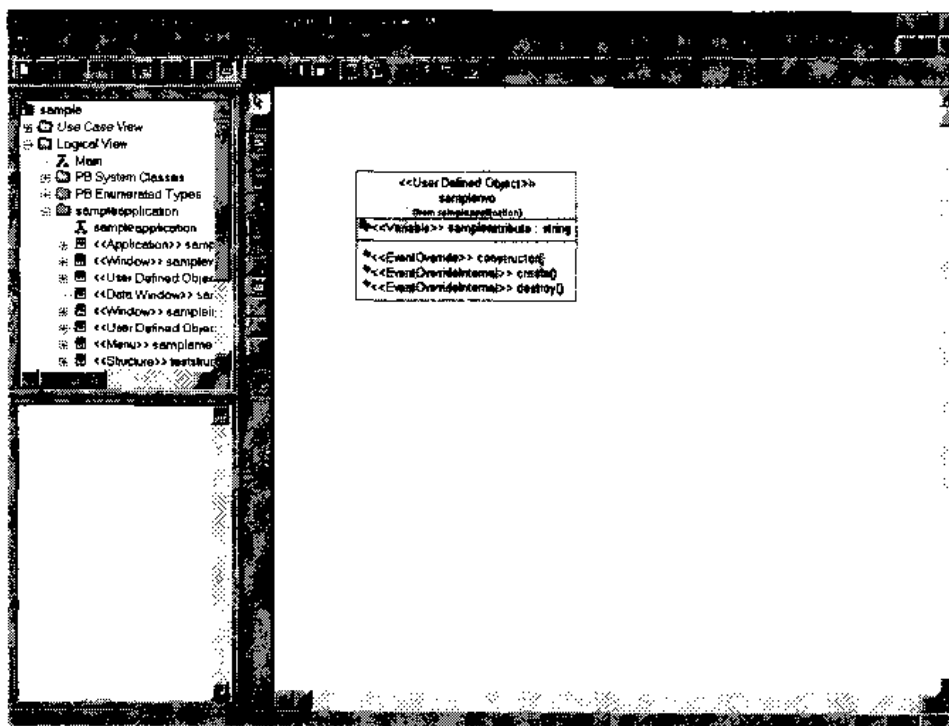


图23.8 逆向转出工程代码实例变量（选择类中的属性显示其文档）

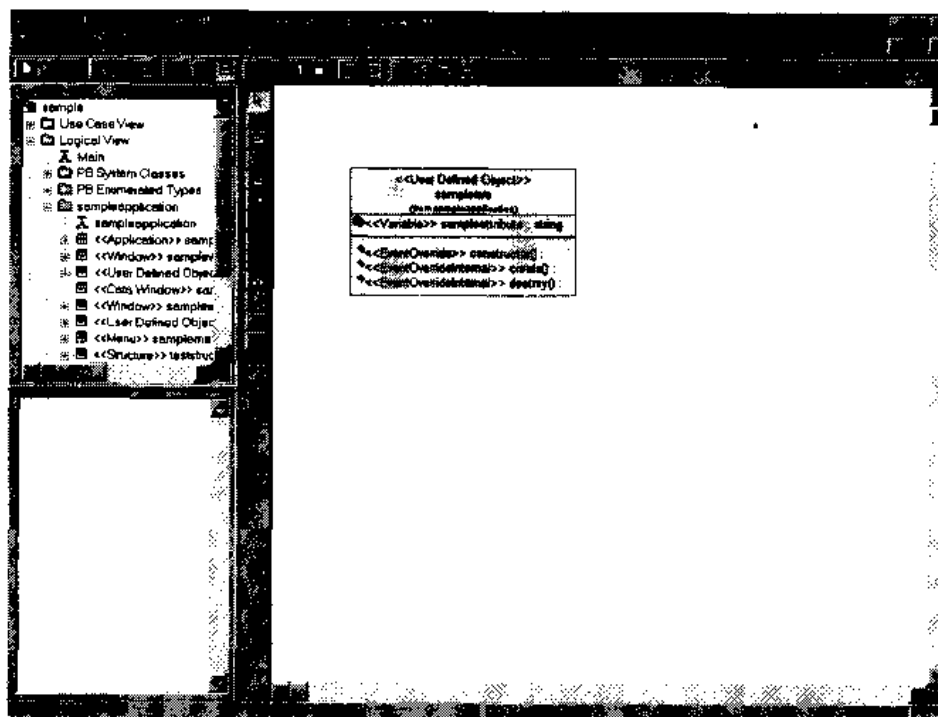


图23.9 逆向转出工程代码函数和事件（选择类中的操作显示其文档）

数据窗口

Rose PowerBuilder链并不支持数据窗口的正向工程。应先在PowerBuilder中生成Data-window对象，然后再逆向转出工程代码到Rose中。

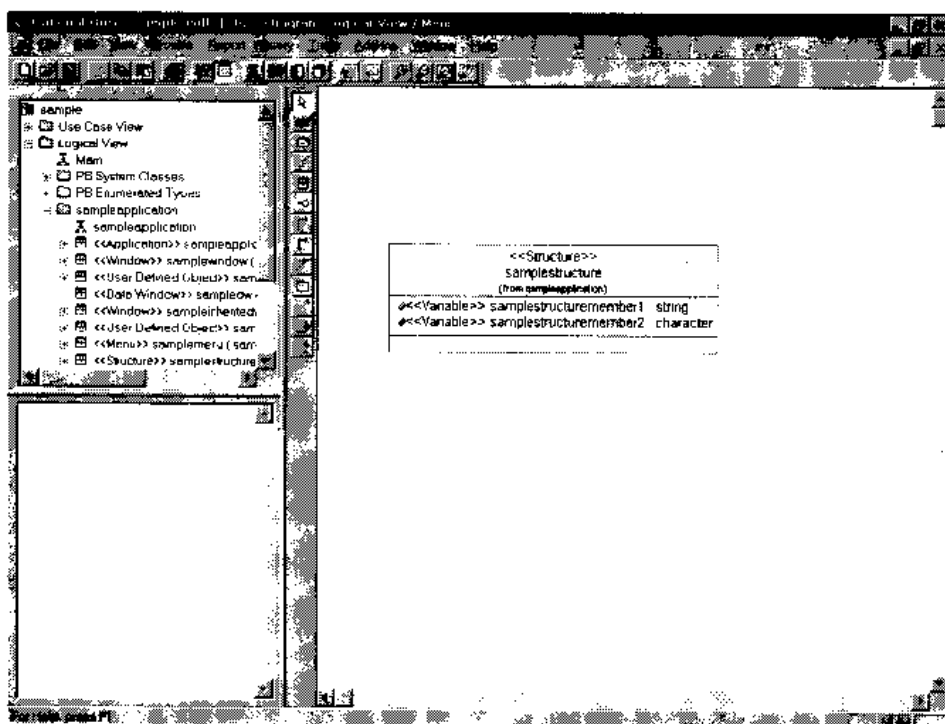


图23.11 逆向转出工程代码的PowerBuilder结构

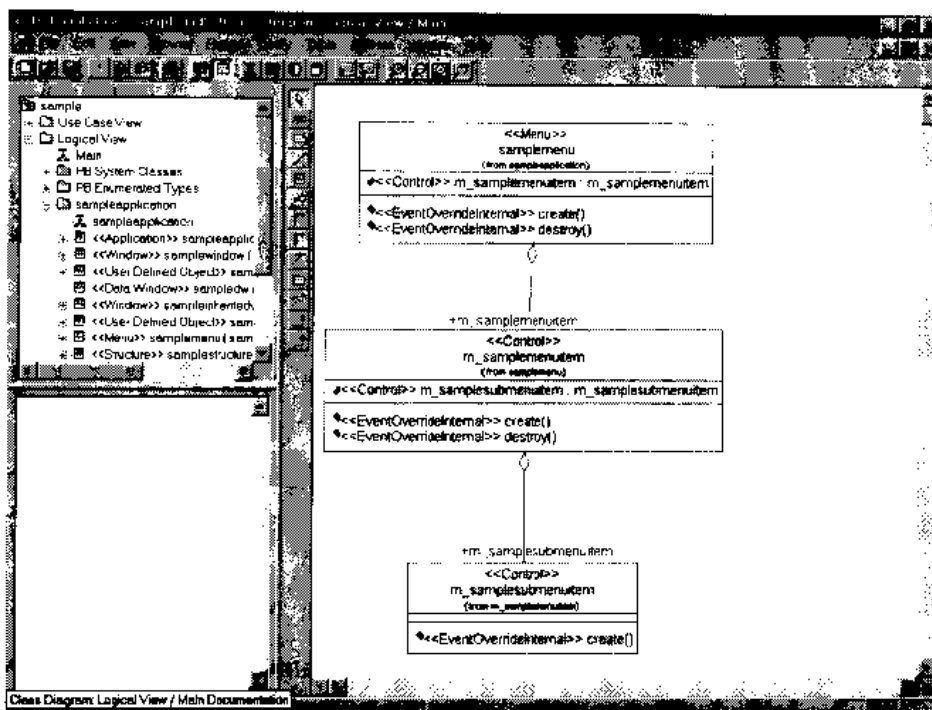


图23.12 菜单逆向转出工程代码

继承关系

代码逆向转出工程代码时，PowerBuilder中的继承关系也逆向转出工程代码。这些关系在模型中用一般化关系表示。例如，图23.13显示了两个窗口间的继承关系。

逆向转出工程代码过程还在逆向转出工程代码的类与PowerBuilder包中的基础PowerBuilder类之间建立一般化关系。例如,上面两个窗口都继承基础类Window。图23.14显示这三个类的继承结构。

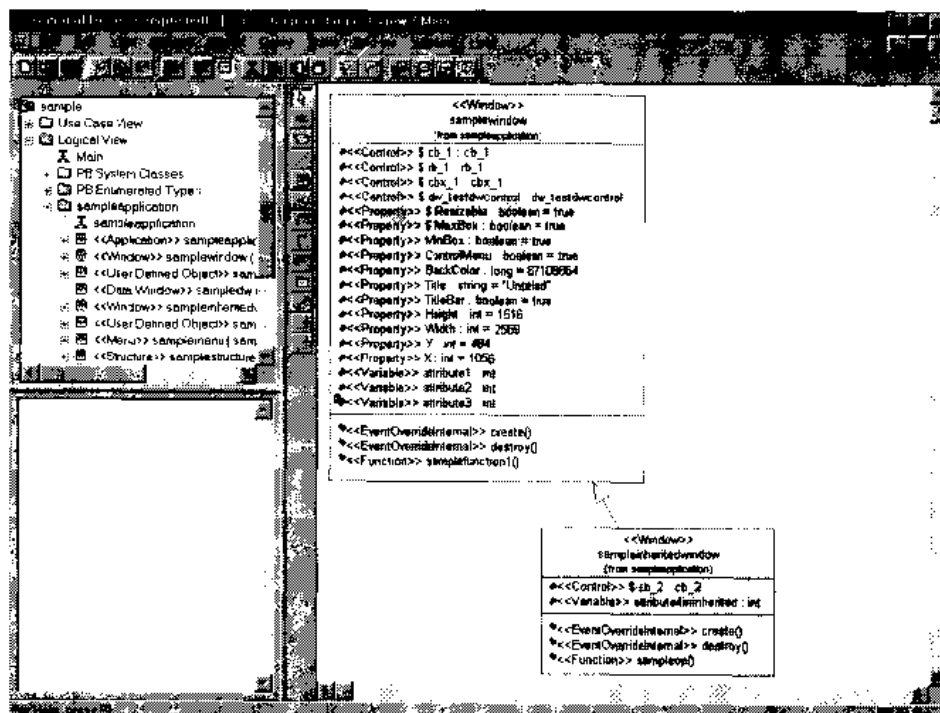


图23.13 继承关系逆向转出工程代码

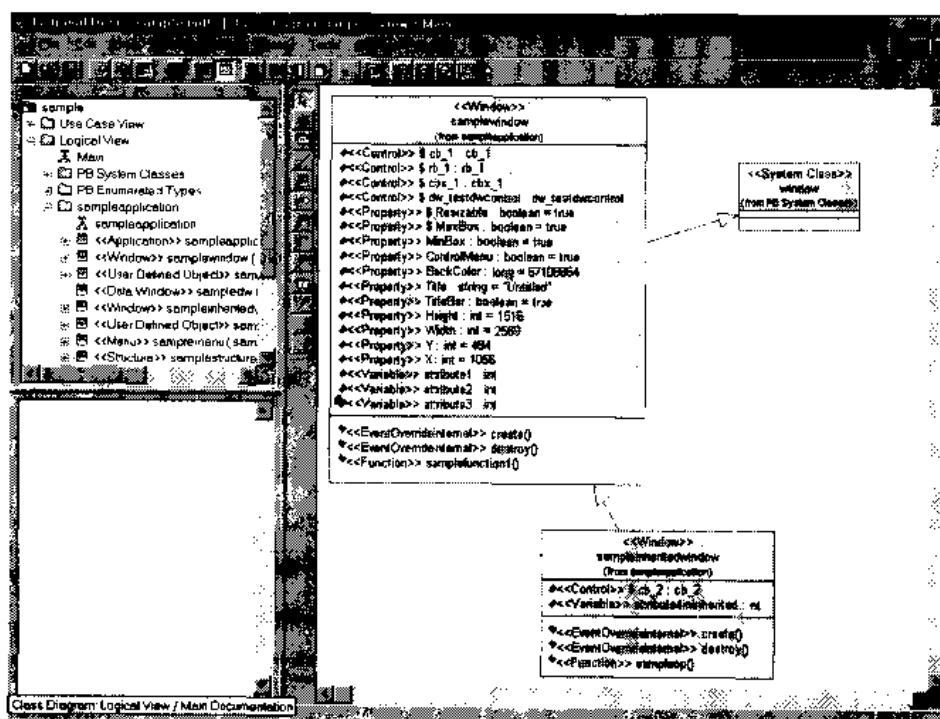


图23.14 从基础PowerBuilder类派生类

关联关系

生成代码时, 关联关系在类中生成实例变量。因此, 许多PowerBuilder中的实例变量在Rose中表示为关联关系。

例如, 假设窗口my_window有个实例变量, 类型为my_nvo。my_nvo是PowerBuilder代码中的不可见用户对象。窗口和不可见用户对象放进Rose中时, 它们之间建立关联关系, 这是从my_window类到my_nvo类的单向关联。图23.15显示了这两个类逆向转出工程代码的结果。

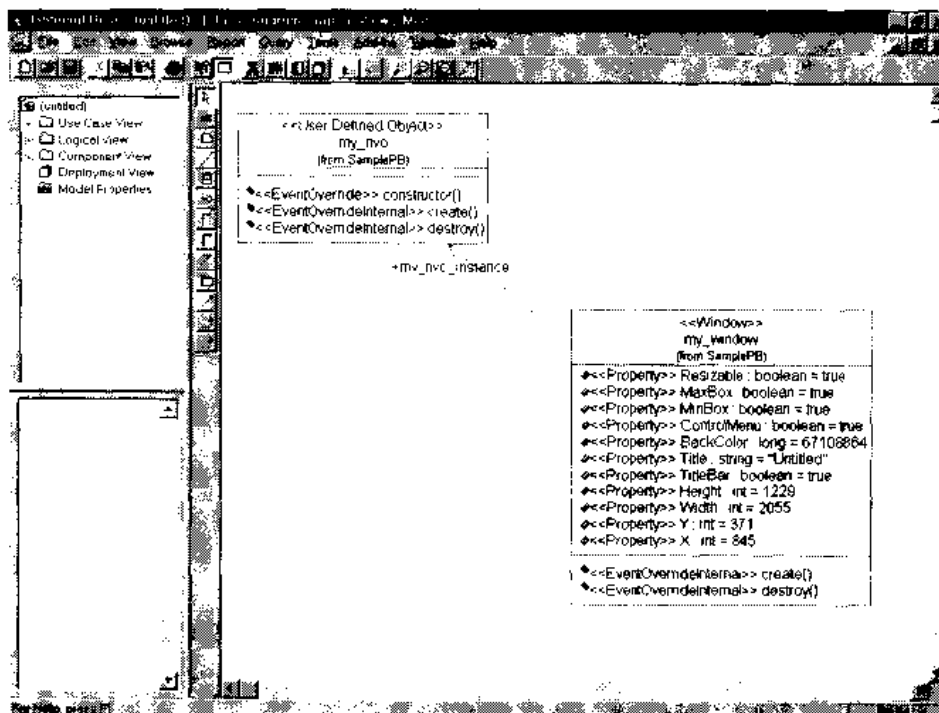


图23.15 关联关系逆向转出工程代码

小结

本章介绍将PowerBuilder代码逆向转出工程代码到Rose模型所需的步骤。大多数PowerBuilder对象直接变成Rose中的版型类。此外, Rose还捕获PowerBuilder代码中的实例变量、函数、事件和关系信息。

我们介绍了几种PowerBuilder对象在模型中的表示。表23.3是其小结。

表23.3 PowerBuilder对象与对应的Rose元素

PowerBuilder元素	Rose元素
Application Object (应用程序对象)	版型为<<Application>>的类
Nonvisual User Object (非可见用户对象)	版型为<<User Defined Object>>的类
Window (窗口)	版型为<<Window>>的类
Instance Variable (实例变量)	Attribute或Association (属性或关联)
Function (函数)	Operation (操作)

(续表)

PowerBuilder元素	Rose元素
Event (事件)	Operation (操作)
Datawindow (数据窗口)	版型为<<Data Window>>的类
Structure (结构)	版型为<<Structure>>的类
Menu (菜单)	版型为<<Menu>>的类
Menu Item (菜单项目)	版型为<<Control>>的类
Inheritance Relationship (继承关系)	Generalization (一般化)

第24章 Oracle8逆向转出工程代码

- 用Rose中的Schema Analyzer逆向转出工程代码Oracle8对象类型、关系型表、关系型视图、对象视图、VARRAY、嵌套表和对象表
- 逆向转出工程代码一般化关系

利用Rose，可以逆向转出工程代码Oracle8结构生成对象模型。本章介绍如何通过逆向转出工程代码Oracle8结构生成对象模型元素。利用Rose中的Schema Analyzer，可以从结构中逆向转出工程代码下列Oracle8元素：

- Object types（对象类型）
- Relational tables（关系型表）
- Relational views（关系型视图）
- Object views（对象视图）
- VARRAYS
- Nested tables（嵌套表）
- Object tables（对象表）

这些结构都建模为上述版型的类。此外，还可以逆向转出工程代码一般化和外部关键字关系。

Oracle8逆向转出工程代码步骤

要开始逆向转出工程代码过程，选择Tools►Oracle8►Analyze Schema。Rose提示输入要逆向转出工程代码的结构名，如图24.1所示。输入结构名并单击OK按钮继续。

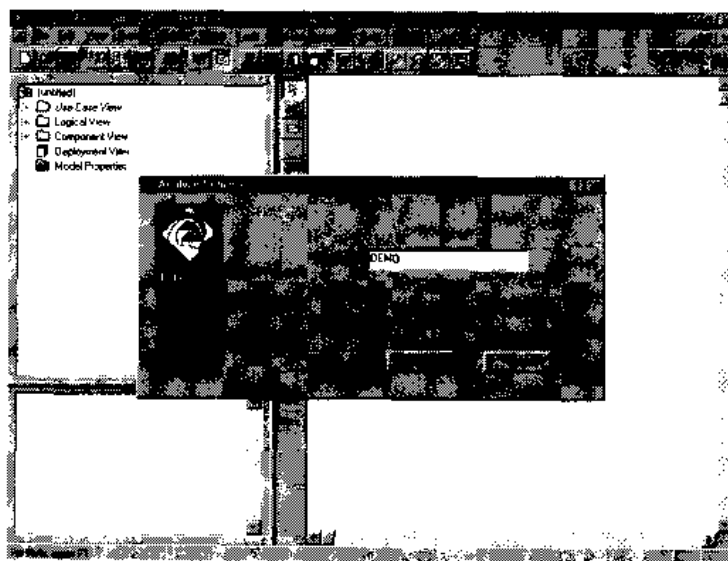


图24.1 Oracle8 Schema Analyzer

在提示下输入数据库服务器名、用户ID和口令，如图24.2。Rose建立与服务器的通信，以完成逆向转出工程代码过程。

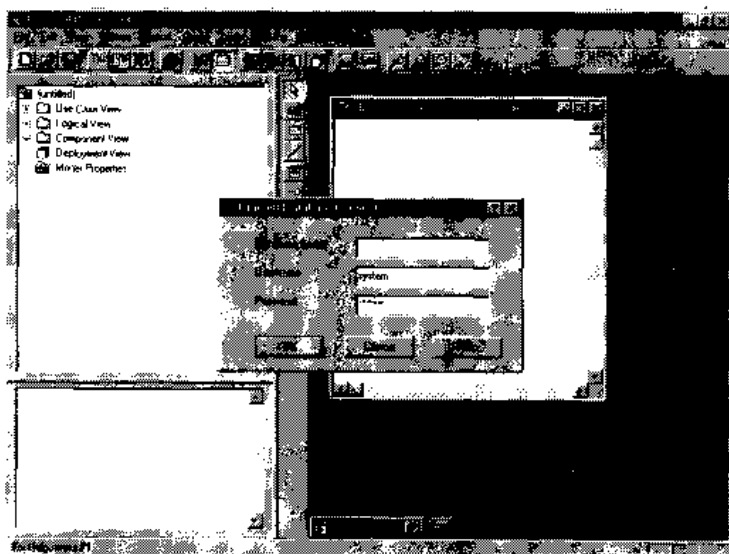


图24.2 Oracle8 Database Connect对话框

Rose在Component视图中生成Oracle8结构的新组件。结构中的任何关系型表、对象表、对象视图和其他的元素都生成Rose中的版型类。

从Oracle8生成的模型元素

Oracle8结构的每种元素都生成Rose中的版型类。本节介绍每种Oracle8结构元素如何映射Rose模型元素。

首先从结构开始。在Rose中，Oracle8结构表示为版型为<<Schema>>的组件。逆向转出工程代码Oracle8结构时，Rose自动将新组件放在Component视图中，以支持结构，如图24.3。

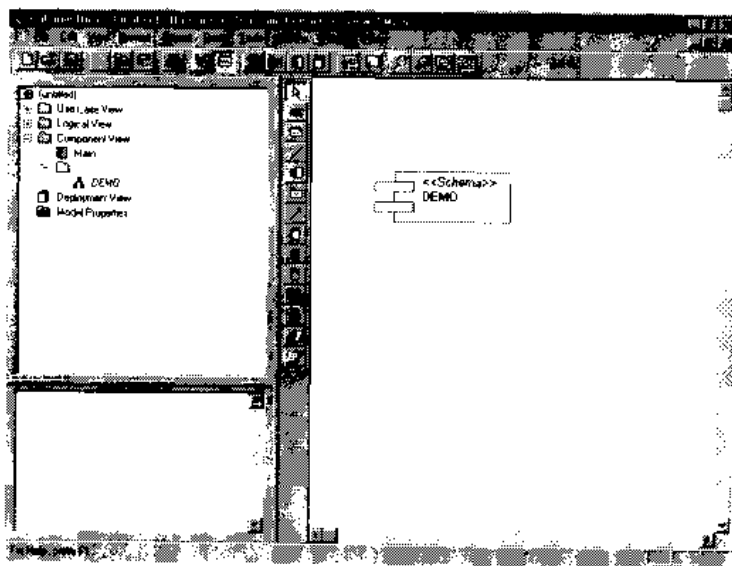


图24.3 Demo结构

结构中的每个关系型表显示为Rose中版型为<<Relational Table>>的类。图24.4所示的Product表就是一例。

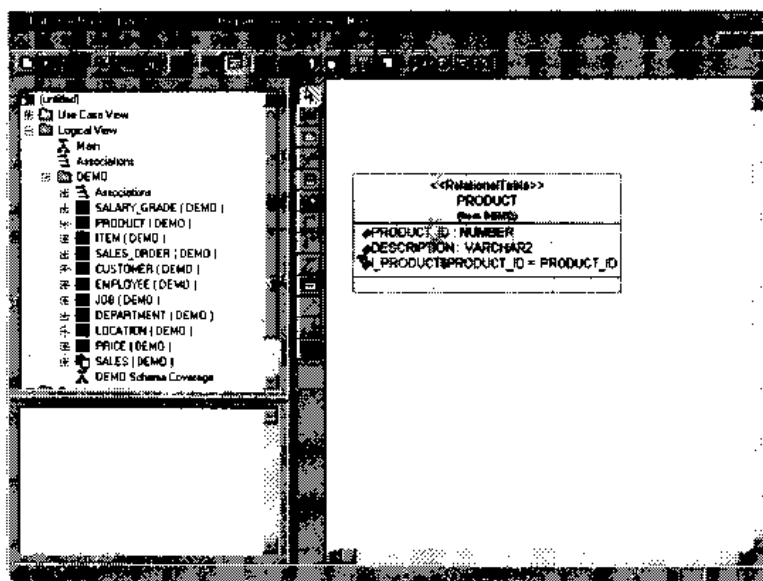
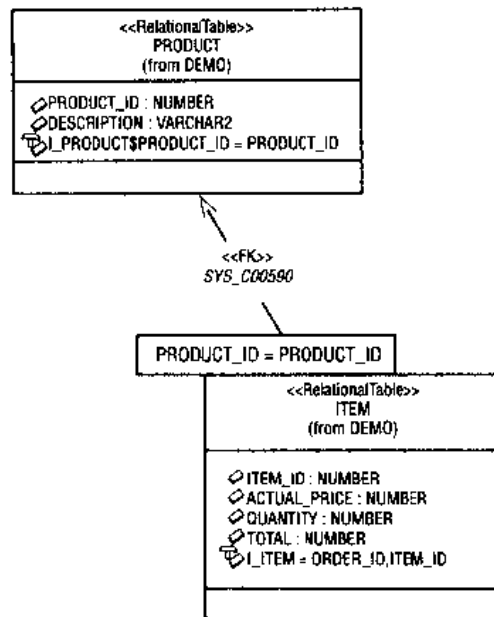


图24.4 Product关系表

数据库表格中每个字段显示为关系型表类的属性。数据库字段的数据类型作为该属性的数据类型。表格索引建模为关系型表类中的属性。索引属性的IsIndex属性设置为True。表格的触发器逆向转出工程代码为操作，可见性为包/实现。在属性的细节中，Rose包括字段规范。例如，如果字段不允许NULL值，则NullsAllowed属性设置为False。

OrderNumber	1	Override
IsUnique	False	Default
Length	22	Override
Precision	6	Override
Scale	0	Override
IsIndex	True	Default
IsPrimaryKey	True	Default
CompositeIndex	False	Default
IsNotNull	True	Default

如果表格包括外部关键字，则Rose将其建模为表与外部表之间的关联关系，采用相应限定符。关联的版型为FK。



结构中的每个关系型视图建模为版型为RelationalView的类。和关系型表类一样，关系型视图中的每个字段建模为属性。计算列建模为类中的方法。图24.5显示了Oracle8逆向转出工程代码的关系型视图。

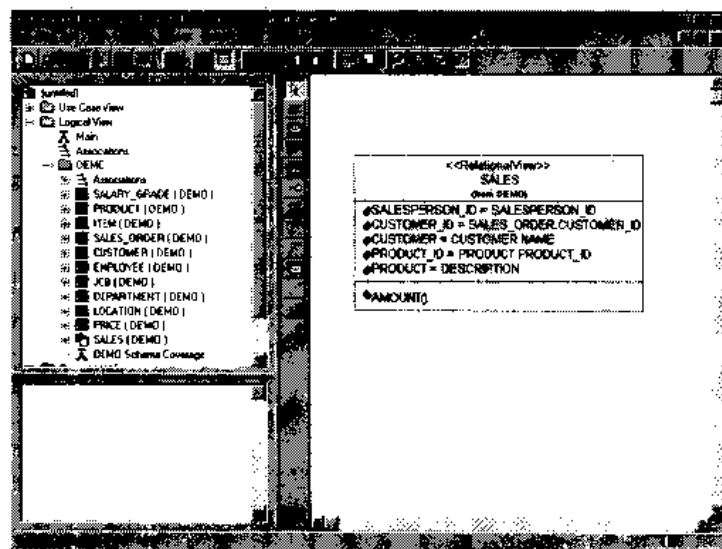


图24.5 Sales关系型视图

对Oracle8结构中的对象类型，对象类型建模为版型为ObjectType的类。对象类型字段建模为属性，函数和过程建模为操作。图24.6显示了逆向转出工程代码的样本对象类型。

小结

本章介绍如何将各种Oracle8结构元素逆向转出工程代码为Rational Rose模型。每个元素逆向转出工程代码为Rose中具有相应版型的类。逆向转出工程代码Oracle8结构后，可以作出修改，然后用Schema Generator将这些改变正向工程到Oracle8中。

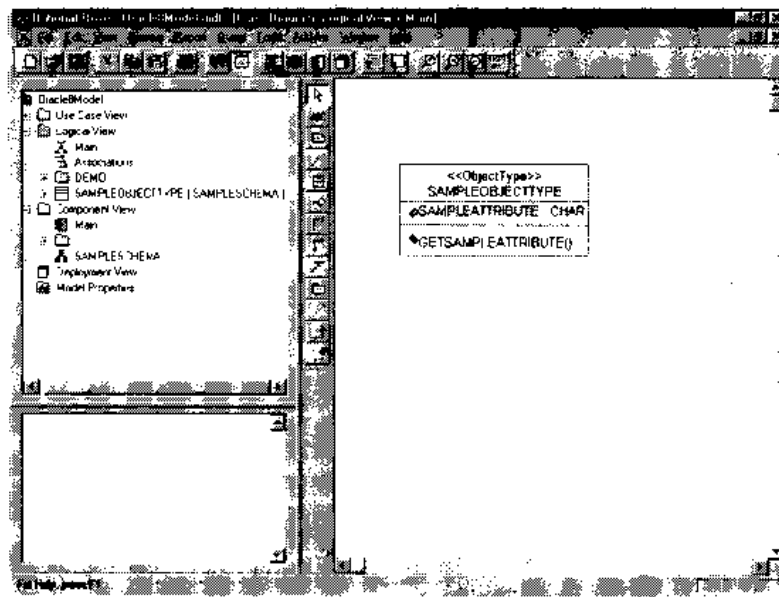


图24.6 样本对象类型