

## 面向对象的概念

### ■ 面向对象的概念

#### ➢ 什么是面向对象的观点？

❖ Coad和Yourdon给出的定义如下：

面向对象=对象+分类+继承+通信

### ■ 对象

➢ 对象——是对客观世界中事物的一种抽象。是由数据（属性）及其之上的操作（行为）组成的封装体。

或：


类和对象封装了对描述某些现实世界实体的内容和行为所需的数据和过程的抽象。

➢ 对象可以量化，可以区分，可以是具体的，也可以是概念化的。

➢ 对象是对象类的一个成员，有时也叫“实例”。

#### ➢ 对象的符号：

类名/对象名
属性：
操作：



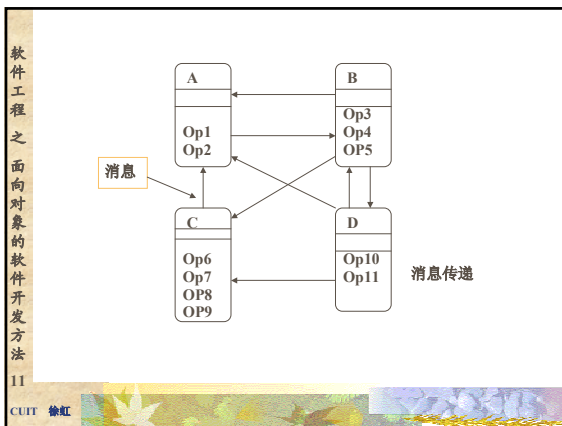
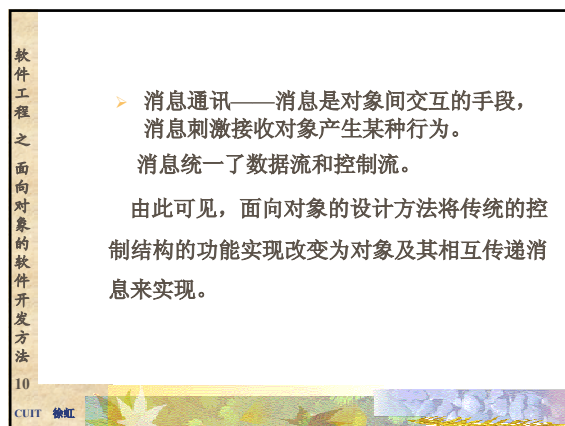
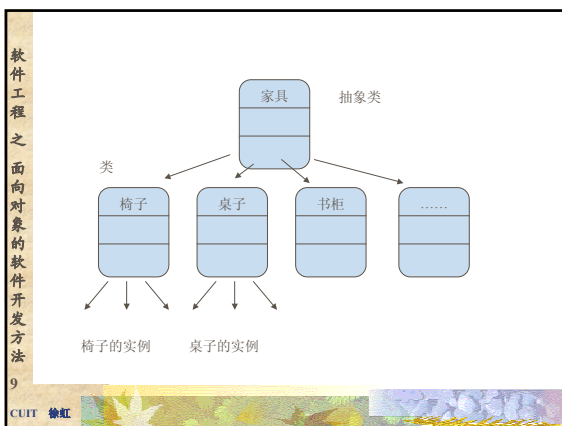
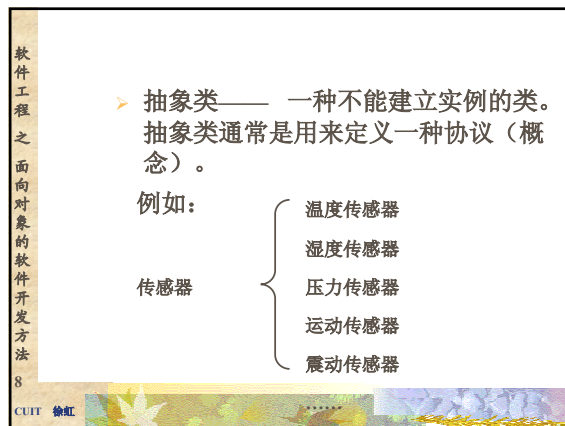
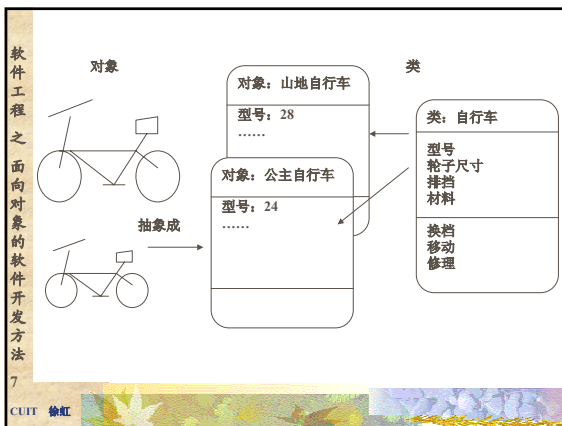
## 第七章

# 面向对象的软件开发方法

### 面向对象的概念

### 面向对象的分析方法

### 面向对象的设计方法



- 封装——又称信息隐藏。用户只能见到对象封装界面上的信息，对象内部对用户时隐藏的。

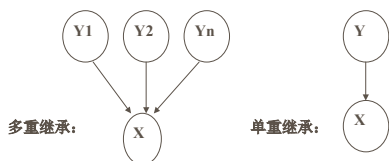
封装的目的是将对象的使用者与设计者分开。

封装有效的实现了模块化。

- 继承——是自动共享类、子类、对象中方法和数据的机制。

继承意味着可以利用已有的定义、设计和实现，简化了相似类的重复定义。实现了软件的可重用性。

- 从结构上讲类是分层的，一个类的上层可以有超类（父类），下层可以有子类，继承具有传递性。



## 面向对象软件的开发过程

- 采用面向对象开发方法构造的软件具有以下特点：

- 面向对象技术构造的模型与客观世界一致；
- 适应变化的需要，修改局限在模块中；
- 具有可复用性

## ■ 面向对象软件的开发过程

- 分析阶段
- 高层设计阶段
- 类的开发
- 实例的建立
- 组装测试
- 维护

## ■ 面向对象技术衍生出了许多方法

- Coad和Yourdon方法
- Booch方法
- Jacobson方法（也称OOSE面向对象软件工程）
- Rumbaugh方法（也称OMT对象建模技术）
- Wirfs-Brock方法
- 统一建模语言UML

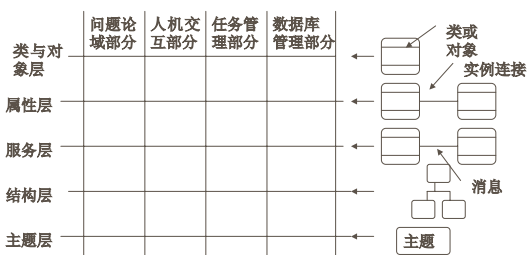
## ■ OOA的基本概念

- 面向对象分析的目标：开发一系列模型，用来描述客户需求的计算机软件。定义所有与待解决问题相关的类。

## ■ OOA必须完成的任务：

- 在客户和软件工程师间沟通，了解基本的用户需求。
- 标识类（即：定义属性和方法）。
- 划分类层次。
- 表示对象间的关系。
- 建立对象的行为模型。
- 在任务1~5之间重复，直至完成建模。

## ■ Coad&Yourdon的软件开发模型



## ■ OOA方法的步骤：

- 确定类和对象
- 确定结构
- 定义主题
- 定义属性和实例关联
- 定义操作和消息关联

## ■ 标识对象和类

### ■ 对象的分类

- 外部实体（如：其它系统、设备、人员）。
- 事物、装置（如：报告、显示、文字、信号）。
- 发生的事情或事件（性质变迁、一系列遥控运动）。
- 角色、所起的作用（管理者、工程师、销售人员）。
- 组织机构（分支、小组、小队、公司）。
- 场所、位置（制造场所、商场、住户、室外）。
- 结构（传感器、计算机、四轮交通工具）。

例：对家庭安全系统的描述如下：

“家庭安全系统可以让房主在系统安装时为系统设置参数，可以监控与系统连接的全部传感器，可以通过控制板上的键盘和功能键与房主交互作用。

在安装中，控制板用于为系统设置程序和参数。每个传感器被赋予一个编号和类型；设置一个主口令使系统处于警报状态或警报解除状态；输入一个或多个电话号码，当发生一个传感器事件时就拨号。

当一个传感器事件被软件检测到时，连在系统上的一个**警铃**鸣响，在一段**延迟时间**之后（房主设置），软件拨一个**监控服务**电话号码，提供**位置信息**，报告**事件状况**。电话每20秒拨一次，直到拨通为止。

安全系统的交互子系统负责读取键盘和功能键，在LCD**显示屏**上显示**提示信息**和**系统状态信息**。”

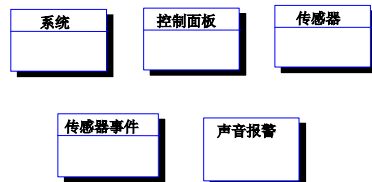
抽取其中的名词，提出一组潜在的对象。

### 对象的特性

- 保留的信息——系统运转时，必须被记住的信息。
- 需要的服务——拥有一组可标识的操作，能够以某种方式修改对象的属性值。
- 多个属性——仅具有单个属性的对象，在分析阶段最好将其归为另一个对象的属性。
- 公共属性——这些属性适用于对象每一次发生的事件。
- 公共操作（方法）——（同上）
- 基本需求——出现在问题空间的外部实体并对系统的任何解决方案都是必不可少的。

潜在的对象类	一般归类	符合特性
安全系统	物	√ 全部符合
屋主	角色或外部实体	符合(6)，不符合(1)(2)
安装	事件	拒绝
传感器	外部实体	√ 全部符合
控制板	外部实体	√ 全部符合
键盘	外部实体	√ 全部符合
.....	.....	.....
主口令	物	不符合(3)
电话号码	物	不符合(3)
传感器事件	事件	√ 全部符合
.....	.....	.....

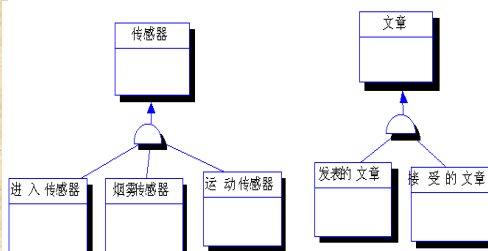
对象的确定具有主观性，以后可能还会修改，但OOA的第一步必须定义对象。



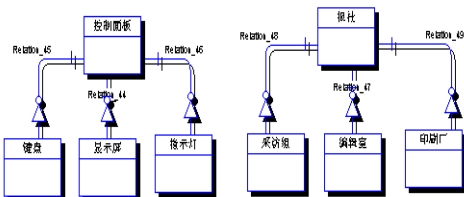
### 确定结构

结构是一种组织方式的思维，在面向对象分析中，结构是问题域复杂关系的表示。

- 分类结构（泛化-特化结构）——表示类（不是对象）的“一般-特殊”关系。泛化类是超类，特化类是子类。

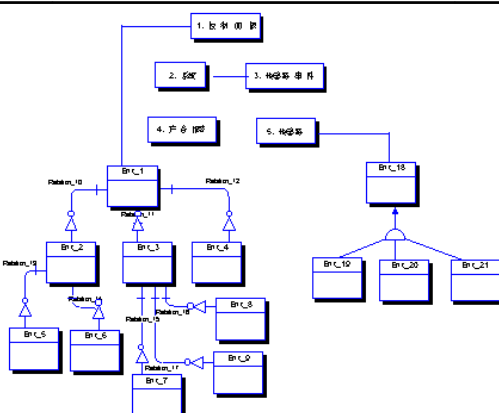
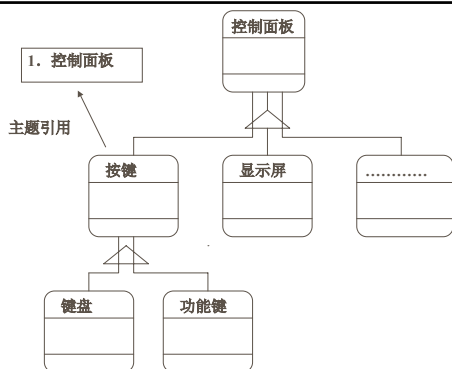


- 装配结构（整体-部分结构）——表示事物的组成结构，即由属于同一类或不同类的成员聚合而形成新的类。



## 定义主题

- 主题是指指导读者或用户研究大型复杂模型的机制。
- 每个主题相当于一个子模型，或子系统。分析者可以根据这一概念区分主题。
- 为每个结构增加一个相应的主题；为每个独立的对象增加一个主题；当主题数大于7，进行精简。



## 定义属性和实例联系

### 定义属性

- 对每个对象应回答下列问题：“在当前的问题范围内，什么数据项完整地定义了该对象？”

标识信息 = 系统标识号+验证电话号码+系统状态

传感器信息 = 传感器类型+传感器编号+报警临界值

警报响应信息 = 延迟时间+电话号码+警报类型

启动/撤消信息 = 主口令+暂时口令+允许尝试次数

由此可得到对象“安全系统”的属性表：

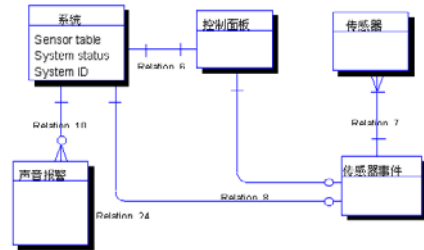
对象：系统

系统标识号  
验证电话号码  
系统状态  
传感器信息  
传感器类型  
传感器编号  
报警临界值  
报警延迟时间  
电话号码  
警报类型  
主口令  
暂时口令  
允许尝试次数



## ■ 定义实例连接

- OOA模型的属性层包括对象属性以及对象之间的关系，即实例联系。
- 实例联系是对象之间的依赖关系。
- 实例联系表示了对象所需要的一部分状态信息。联系可以是双向的。
- 步骤
  - 添加实例连接线
  - 定义多重性
  - 定义参与性



## ■ 定义操作和消息路径

定义操作和方法，如：检索、维护、计算、事件响应、状态改变、消息传送等这样一类操作和具体采用的某一种算法。

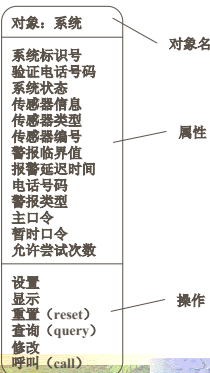
OOA模型不仅定义了对对象间的联系，而且也定义了对对象间消息的传递路径。



## ■ 定义操作

- 操作通过某种方式处理由属性导出的数据结构来实现。
  - 以某种方式处理数据的操作。
  - 完成一次计算。
  - 监控对象以控制某个事件的发生。
- 具体做法：进一步研究问题的过程描述，分离出动词。

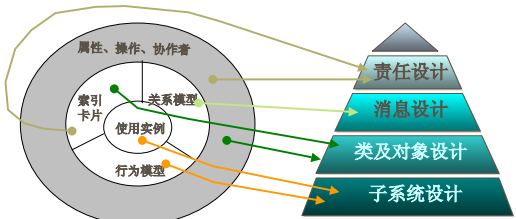
## 完成对象定义



## 面向对象设计

### ■ 面向对象设计概述

#### ■ OO分析模型到OO设计模型的转换：



## ■ 分析和设计的含义

- 分析——是一种研究问题域的过程，该过程产生对外部可见行为的描述。
- 设计——在分析的描述基础上，加入实际计算机系统实现所需细节的过程。

## ■ OOD通常可以分为两个阶段：

- 高层设计——建立应用的体系结构。
- 低层设计——集中于类的详细设计。

## ■ 高层设计（系统级设计）

开发软件的体系结构，构造软件的总体模型。高层的设计包括：

- 将系统划分为子系统的决策；
- 子系统的软、硬件分配；
- 设计框架的主要概念和策略性决策。

## ■ 高层设计模型

### ➢ 客户服务器模型

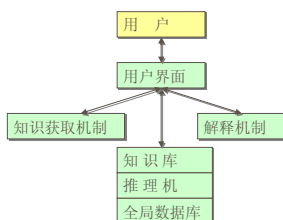
- ◊ 导出的系统即适合于过程设计也适合于面向对象的设计。
- ◊ 客户-服务器模型将系统分为2个部分：客户子系统和服务器子系统。
- ◊ 请求服务的对象都归于客户子系统；服务器接受请求并提供服务。即：客户是服务的驱动者。所以，客户必须了解服务者的接口，而服务器没有必要知道客户的接口。

### ➢ 应用框架结构

根据应用领域系统的特有结构划分子系统（模板、主题、抽象类的集合）。

然后设计细化每个子系统的对象模型、动态模型和功能模型。

如专家系统的典型结构：

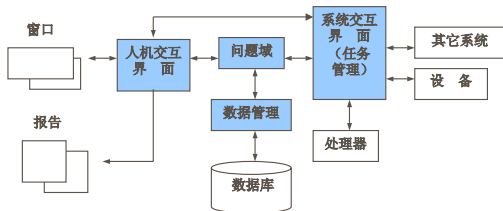


## ■ Caod & Yourdon的OOD方法

像其它的设计方法一样，面向对象设计的目标是生成对真实世界问题域的表示并将之映射到解域，也就是映射到软件上。



## ■ Caod & Yourdon的设计模型



## ■ OOD的内容

- 问题域组元 (Problem Domain Component, PDC)
- 人机交互组元 (Human Interface Component, HIC)
- 任务管理组元 (Task Management Component, TMC)
- 数据管理组元 (Data Management Component, DMC)

## ■ 设计问题域组元

- 问题域组元 (PDC) ——构造特定应用的OOD模型的基本组成部件, 包括: 数据结构、应用域构件、语言构件。
- 设计问题域组元的主要理由——**寻求系统结构的稳定性**。而这种稳定性正是将一个问题域中的系统转变到一个相似问题域中的系统时, 可以重用原有分析、设计及编程结果的关键。

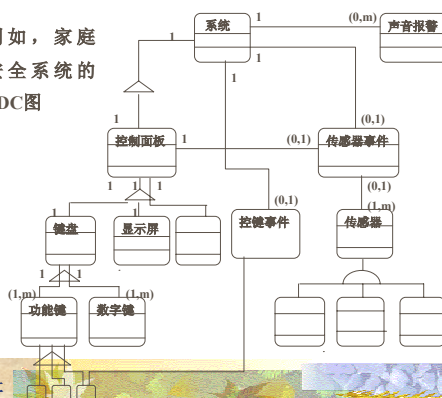
- 问题域部分的设计要针对特定的实现环境, 对OOA的结果加以增补, 其策略如下:

- 重用设计和编程类

对PDC进一步修改。将OOA建立的有关类, 替换成库中的类或子类, 并继承库中类的属性、建立相应的关联。

- 将问题域专用类组合在一起  
在设计过程中, 有时会将原来一些相互独立的类, 在问题域中归入一个新的超类。
- 对继承进行调整  
当OOA模型中某个类或对象的继承关系和采用的语言不一致时, 需要调整。
- 改进性能

例如, 家庭安全系统的PDC图



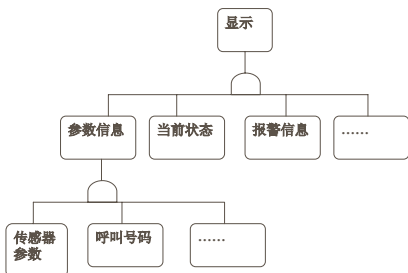
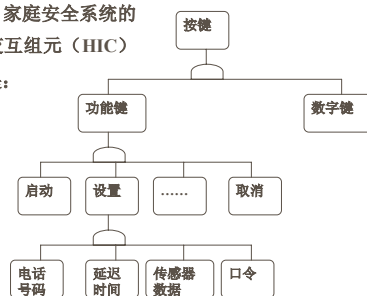
## ■ 设计人机交互组元（HIC）

HIC——表示用户与系统打交道的命令以及系统提供给用户的信息。

HIC将用户界面技术从系统的其它部分中分离出来。这一部分包括：用户输入、显示、交互方式、响应过程、菜单窗口、数据表示、报告、网络接口、控制面板（如：按钮、指示灯、仪表等）。

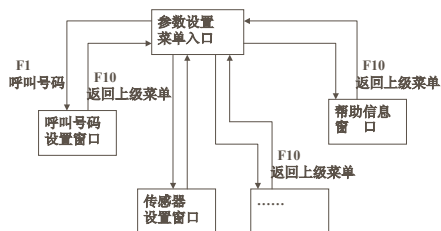
例如：家庭安全系统的人机交互组元（HIC）

可以是：

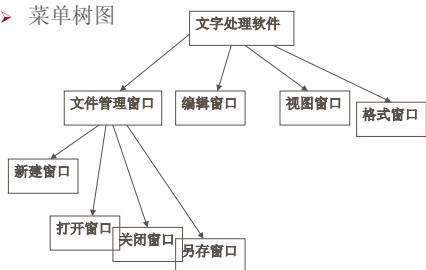


HIC对象模型还可以借助其它工具来补充说明：

➤ 状态——迁移图



➤ 菜单树图



➤ 屏幕样本



## ■ 任务管理问题（TMC）

- 对象中的每个服务最终总要被分配给某个计算机任务，这些计算机任务可看成是一些独立的可调度的实体。
- 许多系统都需要多任务并行处理，例如：
  - 对于具有数据获取机制、负责控制局部设备的系统，需要多任务。

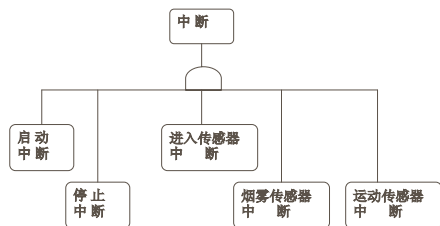
- 对于某种需要同时向多个窗口输入数据的用户接口，也存在多任务。
- 对于多用户系统，很可能存在一个用户任务的多重复制。
- 对于多子系统的软件结构来说，各子系统之间的协调及通讯需要多个任务完成。

- 对于多处理器的硬件结构，必须为各处理器分配任务并支持处理器之间的通讯。
- 对于单处理器，在多任务实时操作系统的支持下，也可以实现多任务并行处理。

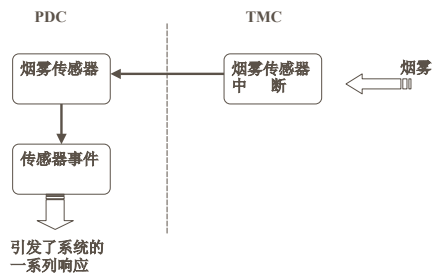
- 任务管理一般是在特定平台（包括：硬件和操作系统）。
- 通过TMC特定平台的处理机制对系统的其它部分隐藏了起来，这样如果应用系统需要移植到其它平台上时，只需替换TMC的类就可以了。
- 任务管理组元可以看成是应用系统与平台之间的接口。

- 任务的选择和调整的策略如下：
  - 识别事件驱动任务（一些与硬件设备通讯的任务）
  - 识别时钟驱动任务（以固定时间间隔激发的事件）
  - 识别优先任务和关键任务

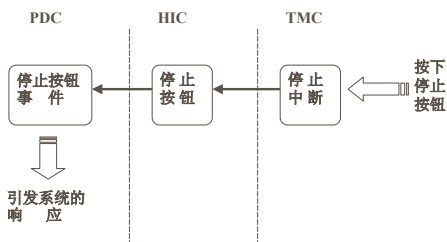
- 识别协调者
- 审查每个任务（要使任务数保持到最少）
- 定义每个任务（说明是什么任务、如何协调、如何通讯）



当烟雾传感器被激活时，系统的完整的执行机制为：



如果触发事件是“停止”按钮时，系统的执行机制会有所不同：



## ■ 设计数据库管理组元

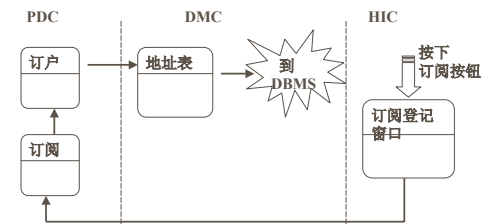
- DMC提供了数据管理系统中对象的存储及检索的基础结构。
- 建立DMC的原因主要是DMC可以独立于各种数据管理模式，使数据库技术从系统的其它部分中分离出来。

## ➢ 为了开发DMC，要考虑以下点：

- 哪种数据库查询方式能用于支持你的应用。
- 建立哪些对象来封装查询的实现（SQL）。

- 在DMC 与PDC对象之间应建立的实例连接。
- 检查一下HIC部分哪些屏幕、字段、或报表需要数据库查询？如果需要，则要在DMC中建立一个查询对象。

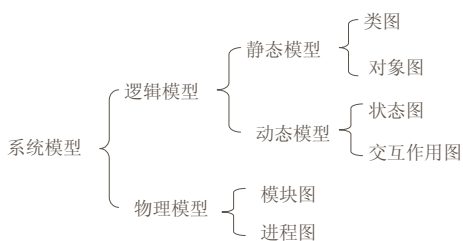
下图简单地说明了DMC与应用系统和数据库间的关系，及执行过程。



## Booch面向对象的方法

- Booch 是面向对象方法的最早倡导者之一。
- Booch认为开发过程为螺旋上升模式，每一次重复的步骤如下：
  - 从应用的问题域中发现类和对象；
  - 分析类和对象的功能、行为，确定其属性和操作；
  - 找出类、对象之间的关系；
  - 说明每个类和对象的界面和实现。

### ■ Booch方法表示系统模型：



### ■ Booch采用以下方法构筑系统模型：

- 类图——在Booch方法中作为逻辑的、静态模型的描述方法；描述系统的构成。

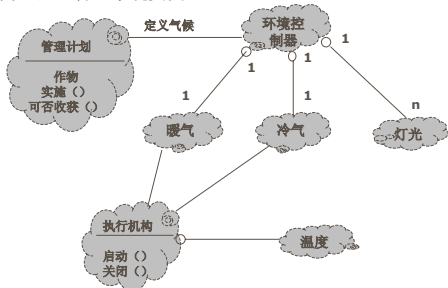
类的图形表示：



类之间相互关系的表示：

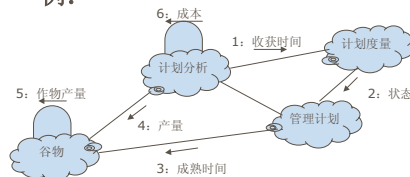


例：温室管理系统类图



- 对象图——在Booch方法中作为逻辑的、静态模型的描述方法；表示系统行为的基本结构。

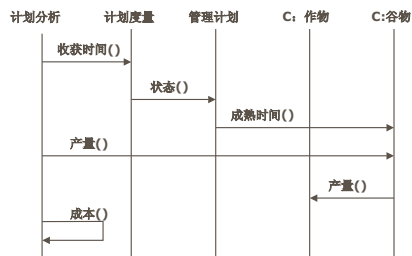
例：



- 状态迁移图——作为逻辑的、动态模型的描述方法；表示一个类的动态行为。
- 交互作用图——作为逻辑的、动态模型的描述方法；表示几个对象在共同完成一个系统功能时表现出的交互关系（亦即：一个类的动态行为）。

交互作用图与OMT的事件追踪图十分相似，区别是：交互图主要表示操作而不是事件；是对象图的另一种表示形式。

例：温室管理系统的交互作用图

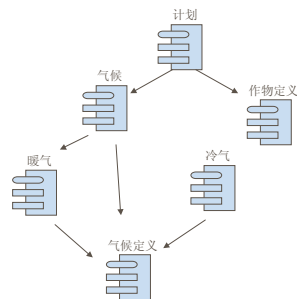


- 模块图——作为物理模型的描述方法；表示如何将类和对象分配到不同的软件模块中。每个符号表示一个模块，每个模块是一个文件



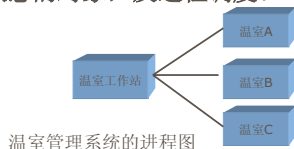
连接文件的箭头表示两个文件的编译依赖关系。

例：温室管理系统的模块图



- 进程图——作为物理模型的描述方法；表示如何将可同时执行的进程分配到不同的处理机上。

对于单处理级系统，表示处于活动状态的对象，及进程调度。



温室管理系统的进程图