UML2.0 实战教程

(基于年度最佳 UML 建模产品 Trufun Kant X)

作者：trufun(www.trufun.net)

2010.3

目录

第一章 理解面向对象

在面向对象思想日益流行的今天，应用面向对象方法进行软件开发已经成为一种不可阻挡的趋势。 面向对象建模语言作为面向对象开发的组成部分，逐渐引起了更多人的注意。

虽然很多公司在使用工具在建模，很多开发人员使用的开发语言、开发工具都是面向对象的，他们也会使用建模工具，然而，在今天有多少公司从建模中受益，又有多少产品因使用了建模而摆脱了长期低层次重复开发的命运呢？又有多少公司在软件开发过程中使用模型图进行交流呢？这个问题的答案可能有多个，但有一点是肯定的，人们在应用新技术的时候，会不自觉地运用以往的面向过程的方式进行思维，没有从面向对象的角度，以一个全新的视野去观察问题，去解决问题，也就没有真正发挥面向对象的优势。

本章从初学者角度，介绍了面向对象的基本概念、基本观点以及主要方法。它是我们后面学习和讨论 UML 建模的预备知识。

第一节 对象

面向对象思想来源于人们对周围真实世界的认识，在真实世界里，对象可以是一个物理实体，如桌子、书等。也可以是特定的实体，如我的桌子，他的 UML 建模工具书；另外对象也可以是无形的事物，如经济效益、交易等，虽然无形，但仍然可以描述、创建和销毁。所以它们也被称为对象。

对于真实的对象，我们可以用一组属性来描述它，如桌子有颜色、大小，高低、形状、用途等等。这些被称为对象的属性，另外桌子还可以被清洁、搬动和维修。这些被称为对象的操作。

软件领域的对象是真实世界对象的一个映射，它是一个客观实体，作为软件运行时的基本单元而存在。它由名称、属性、操作构成，虽然它来自客观世界，但它却被赋予了很多软件工程的思想，这使得面向对象在重用、沟通、修改、扩展、确认、验证等方面表现出十分优异的特性。

真实世界的对象有成千上万个属性，是不是把它们通通搬进计算机世界里来呢？回答显然是不行的，分析人员往往把那些只对用计算机管理业务所必须的对象属性映射到计算机里，而忽略那些无助于处理业务的无用信息。这个过程叫作对象抽象，对象抽象会因各个软件系统处理业务的不同而不同，比如前面提到的桌子，如果该软件系统是专门处理桌子销售的，木材的产地、工匠姓名都是无用信息；如果该软件系统是桌子生产厂专门管理生产的软件，木材的产地、工匠姓名都是有用信息。

经过抽象的对象信息一般分为以下四类：

（1） 对象的独特身份号。
（2） 用于对象结构创建的结构信息。
（3） 对象当前的状态信息。
（4） 存储供访问的信息。

对象的操作就是对象所能做的事情，包括对象所能做的所有事情，面向对象有一个很重要的特性是把操作的行为赋予给操作的对象而不是操作的使动者。行为必须保存在对象里，这将大大简化代码，任何使用该对象的对象，必须通过对这个对象内的行为调用才能实现。

所以，对象从行为上可分为两类：

（1） 对自身可以做什么？

（2） 为其他对象可以做什么？

第二节 类

在前面讨论对象的基础上我们做进一步讨论，还比如前面讲过的桌子这个产品，如果经销商要买上百个桌子，要记录所有真实桌子（每个桌子有一个唯一身份）的销售轨迹，几乎是不实际的，为此，在面向对象中定义了一个新概念——类。类首先是一种表示法，它包含了足够信息，根据这些信息，就可以反映出这上百个桌子的销售轨迹。其次它是一种规则集合，是适合所有桌子的规则集合，也是能反映出经销商关心问题的规则集合。最后它是一个模版，通过这个模版，可以精确地表示一个具体的对象，每一个对象都可以经过这个类的创建而成，它们都有一个与众不同的身份。

分类是人类在解决复杂问题时常用的方法，分类后针对不同的问题采取不同的方法，可以大大简化对问题的处理。面向对象也是采用这样的处理手法，也就是说，不对每个对象进行逐个描述，而是描述代表一批对象共性的类。

类是对象的抽象，是反映了该类所有对象的属性、操作的抽象定义。所有对象在这个类定义的集合中，每一个对象只是类的实例。

第三节 封装

封装是面向对象开发方法的一个新特征，封装是把对象的属性、操作结合在一起，构成一个独立的对象，这样可以更有效地使用对象。一旦封装，内部信息对外界是隐藏的，也就象一个四周密封的房子一样，不允许直接对对象的属性进行操作，要想操作，只能通过局部接口（相当于房子的大门）。外部只能看到对象对操作的反应，而不知道对象是如何做出这一反应的。所以，封装包含两个方面的含义，一个是信息隐藏，另一个局部开放，使外界可以使用该对象，而不必关系其内部机制。

由于采用了封装，任何属性变量都隐藏在类中，没有了全局变量等概念，使用对象的属性是有条件的，只有对象本身才能使用它的属性信息。要保证对象对外界请求作出正确的反应，对象必须了解自身的结构，在需要时，对象可以通过它的操作来访问和支配和它在一起的对象属性。因此，对象必须封装使自己正常运转的信息，这是一个基本要求。

另外，对象还必须把最低能使用它的信息公开出去，才能保证其他对象知道并访问自己，而不要把其他对象不感兴趣的信息也公开出去，实现对象这个功能的元素就是对象的接口。

是否有了以上两点对象就可以正常工作了，回答是不肯定的，对象能否正常工作，还与它当时所处的状态有关，所以对象的状态信息也必须封装进去。

外部对象使用某个对象是有一定的目的性的，尽管对象对外定义的公开接口相同，外部对象通过接口可以调用不同的对象，来实现自己的目的，这就是后面要讨论的多态。封装保证了对象的数据不会被其他对象破坏。

第四节 继承

对目的相同的各种对象特性进行组织的过程叫做泛化，“属于某种类型”经常用于描述这种关系，对这种关系的实现叫做继承。子对象不仅继承了父对象的所有特征，而且还添加了自己独有的特征。父类和子类之间的关系又叫做一般-特殊关系。这种继承是自动地、隐含地复制父类的所有属性和操作，并且是一种逻辑上的必然。如果这个子类又被更下层的子类继承时，它继承来的属性和操作和自己新定义的属性和操作又会被更下层的子类继承下去，这就是继承的传递性。

继承的重要意义在于，简化了人们对事物的认识和描述，比如如果考虑博士生也是学生这个事实，那么，他理所当然地继承了学生类的全部特性，要上课、要考试。这时人们看问题的重点是发现博士独有的特征，如需要在主要的期刊上发表论文。所以在定义子类博士生时，不要把学生的属性和操作再重复定义一编，只需要定义它是哪个子类，并定义自己特殊的属性和操作，这将大大降低编程开发的强度。面向对象开发一个很重要的思想就是复用，子类继承父类，这本身就是复用。

一个类可以是多个父类的子类，它从多个父类中继承了属性和操作，这叫做多继承，比如刚才提到的博士生，除了学习以外，还要做研究，博士生也是研究人员，所以，博士生这个子类是学生和研究人员两个父类的子类，在开发时，只需要为新子类定义更少的特性，就可以满足要求，更多的属性和操作来自于各个父类属性和操作的交集。在这里需要特别注意，不同的父类可能有同名的方法或属性。需要解决各个父类属性和方法同名的问题。当然，这需要实现语言支持这一特性。

第五节 消息

在系统中的对象只有对外提供服务，才能发挥自己的作用，当系统中的其他对象请求这个对象提供服务时，该对象就给予响应，并完成指定的操作。在这个过程中，其他对象要求提供服务这个信息就叫作消息。

前面讲过的封装使对象具有了独立性，现在讨论的消息则使对象之间发生联系，这种联系是动态的，是根据需要发出的。为了完成整个系统的功能，对象之间只有不断发出消息进行通信，才能相互配合完成系统的职责。比如上电影院看电影，你只要向售票小窗口说看几点什么电影并递上钱，票和找零的钱就从小窗口送出来，你无法直接把手伸进小窗口去直接拿票，你也无须知道售票员是男是女，票是怎样出的，帐是怎样算的，你只要对着售票小窗口发出消息并提供消息的参数（几点什么电影）即可。

一般来讲，消息应包含以下信息：提供服务的对象标识、服务标识、输入参数、响应信息。消息的接收者提供服务，它必须能够理解其他对象发出的消息，也就是说消息必须采用规定的消息协议格式。在具体的实现语言上，消息就是被调用的函数、子程序。对象和对象之间这种动态的联系，叫消息链接，消息链接是有方向性的。

第六节 结构

在现实世界里，任何对象都不是孤立的，对象之间都是存在着一定的关系。来自真实世界的对象映射到软件世界中，仍然存在这种关系，一般情况下，这几种关系是指：对象间的分类关系，对象间的组成关系，对象属性之间的静态关系，对象行为之间的动态关系。

如果一个结构是一组具有一般-特殊关系的类组成，类为结构的节点，由继承关系构成结构的连接骨架。则由单继承形成的结构叫层次结构，由多继承关系形成的结构称作网络结构。

在现实世界的很多对象，都存在着由其他小对象组成这个事实，在面向对象世界里，这种关系仍然成立，人们称之为整体-部分结构。

如果对象间是一种比较松散的关系，整体对象和部分对象可以独立地创建，并在整体对象中设置一个部分对象的属性，它可以是部分对象的标识，也可以是指向部分对象的指针。同时，部分对象也可以属于多个整体对象，其生命周期与整体对象不同，这种对象间的关系叫聚合。

如果部分对象作为整体对象以部分对象类型定义的属性而存在，也就是说，对象中存在对象，这个部分对象只能属于这个整体对象，并且和整体对象具有相同的生命周期，这种关系被叫作组合。

如果对象之间的关系是永久的、固定的，一个对象了解并知道另一个对象，并可以通过对象的属性表达出这种关系，但这两个对象间又不具备整体和部分的语义。这种关系叫做关联，有时又被叫作实例连接（用来与消息链接区别）关系。

如果对象之间的关系是动态的、可变的，并且是暂时性的，这种关系叫做消息连接关系。

第七节　多态

如果在父类定义的属性和操作被子类继承以后，表现出不同的属性和操作，这种现象叫作多态。多态的基本表现是：属性名或操作名在子类和父类中相同，但语义不同。比如笔可以写字，但写什么样的字并没有确定，在执行时，子类钢笔、毛笔、粉笔都继承了笔的写字操作，但功能却不一样，钢笔写出钢笔字，毛笔写出毛笔字，粉笔写出粉笔字，毛笔子类无法写出钢笔字，同理，如果毛笔类再分为大字笔和小字笔两个子类，它们两个的功能又不一样……因此，对于同一个消息，让继承关系中的不同对象去执行，执行的结果也不同。

多态的实现主要靠以下几个机制：

重载，即在子类中对继承来的属性或操作进行重新定义（俗称改写），有很多开发语言支持这一特性。

动态绑定：即消息的接收对象根据接收的消息，动态地确定调用自己的那个操作。

类属：即服务的参数类型是参数化的，根据这些参数类型，动态地确定调用自己的那个操作。

定义接口类：另一种实现多态的方式是定义一个特殊的抽象类（缺乏完整定义的类）接口类，它只声明一些操作标记，告诉其他对象如何触发该行为的细节，如名称，参数、返回值，再创建接口类多个子类，在每个子类中实现接口类定义的所有操作，从实现相同的接口类这个角度来看，表现出相同的一组接口由不同的子类去实现这样的多态。

第八节　永久对象

永久对象指生存周期可以超越程序的执行时间而长期存在的对象，又叫持久对象。一般情况下，对象的生命周期不能超过程序的运行时间，一旦程序运行停止，所有对象都会立即消失，下次运行程序，再生成对象；或者在上次结束时，把所有的对象都用文件或数

据库保存起来，下次运行再恢复出来，虽然从表面看好像延长了对象的生命周期，但存在以下缺陷：首先要编制对象与文件或数据库之间要进行格式转换操作，其次是对象有一段时间是以普通数据的身份而存在的（不是面向对象的）。

利用永久对象的概念，无须额外编制任何程序，只要在程序中对某个对象声明为永久对象，则系统会自动完成它的存储、转换、恢复等一系列问题，对象和数据库之间直接交互，对象从库中提取出来放到内存，和原来的一摸一样，有相同的属性和操作，能和其他对象直接交互，如果有多个对象，它们之间的继承和关联关系也和原来一样，能够使对象在不同的程序之间实现共享，这将大大减轻编程的工作量。

永久对象的实现要求有较高的技术支持，其中重要的技术如存储和管理永久对象的对象管理系统，同时还需解决对象的共享、并发、一致性等问题。

第九节 主动对象

如果按照一般的概念理解面向对象思想，只有当对象接到消息后，才能向消息的发送方提供服务，那么如果系统存在多个并发执行的任务时，就无法解说这多个任务的并发执行，是不需要接收任何消息这个事实。

主动对象就是设计用来刻画对象的这种行为，主动对象的行为被设计成一个任务，用来描述业务领域识别的任务和分析时识别的任务。它封装了一组属性和操作，但其中至少有一个操作，不需要接收参数，就可以主动执行，并能完成相应的任务。实现时要求能并发，并能主动执行，通常是用进程或线程来实现的。当然，主动对象的一些操作，也可以是在接收消息之后才执行的一般操作。主动对象的引入，为描述一个多任务系统提供了依据。这个依据将成为编程人员编程的主要依据。

与主动对象相对的还有一个概念就是被动对象。它是指对象的每个操作都是在消息驱动下被动地执行。向对象发一个消息，它就响应这个消息，执行被请求的服务，否则，它的操作就不能被执行，通常是用函数、例程或过程来实现。

第十节 小结

对象是描述客观存在的事物，类是同类对象的抽象表达，封装是把对象的属性和操作结合起来，形成一个独立的单位，继承是讨论父类和子类之间的共性问题，消息是一个对象向另外一个对象发出的服务请求。对象间存在着一般和特殊关系、整体和部分关系、永久关联关系，暂时依赖关系。多态是指在一般-特殊关系中，虽然一般类的属性或方法与特殊类的属性或方法名称一样，但语义却不一样。永久对象是指生命周期超过程序执行时间的对象。主动对象是指不需要接收消息就可以主动执行操作的对象。

第十一节 习题

1. 举例说明属性、操作、接口之间的关系。
2. 试对对象间关系的紧密程度从高到低排个序。
3. 试从面向对象的概念中找出多个具有辩证关系的两个相反的概念如：主动对象和被动对象，永久连接和暂时连接……

第二章 UML 入门

第一节 UML 的发展历史

设计 UML 的主要目的就是描述面向对象系统，早在 20 世纪 70 年代，很多开发人员发现，应用面向对象的术语更容易进行编码的讨论，这需要先画出对象模型，然后根据模型进行设计，会使设计变得更容易。同时，由于模型元素和程序代码有很多相同点，用模型代替代码进行设计，可以提高抽象的层次，避免过早地陷入代码的细节中去，并且使模型转换为代码更加简单、直观。

建模技术并不是在软件行业首先使用，事实上，建筑业、数据库设计等行业较早就使用建模技术，随着计算机软件的复杂程度的不断提高，人们对软件系统的要求也越来越高。几乎在同时，全世界有数以百计的人在研究建模技术，以期望解决新出现的问题，由于没有统一的组织和管理，大家各自为政，且相互攻击，结果谁也拿不出适应于所有行业的通用最佳方案，使建模工具开发商和企业无所适从。整个软件建模行业处在低层次徘徊状态，开发一个灵活的，可扩展的、可靠的、健壮的、满足软件和其他行业的建模需求的建模技术成了当务之急。

到了 20 世纪 90 年代，出现了几个有代表性的人物和事件，首先是 Ivar Jacobsons 提出的面向对象软件工程（ＯＯＳＥ）中用例概念， 它一方面为建模找到了原始信息来源，另一方面是建模和需求采集结合起来；其次是 James Rumbaugh，他提出的对象建模技术（ＯＭＴ），从静态建模（类图、包图）、动态建模（顺序图、状态图、活动图）基础定义入手，试图找出基础数据与系统目标之间的关系；最后 Grady Booch 提出的 Booch 方法所采用的对象模型要素是：封装、模块化、层次类型、并发。使用的图形文档包括六种：类图、对象图、状态转换图、交互图、模块图和进程图。该方法使问题域和责任域很好地对应起来，上面三人工作都从某一方面对建模技术作出了贡献，只有把这三人方法结合起来，才能满足创建一个统一的、全面的、建模技术所要求标准。

1995 年秋，Booch 和 Rumbaugh 完成了两种方法的结合，产生了 UMLl0.8 版本，后来 Jacobson 加入到该团队，用例的概念也融入了新的 UML 标准，1996 年秋天，发表了 UML0.9 版本，UML 的发展很快引起了很多组织和企业注意，特别是面向对象国际组织（OMG）发布征求意见书，以及 Rational 公司建立的 UML 联盟，吸引了包括微软、HP、Oracle 等大企业加入，这些努力共同促进了在 1997 年 1 月发布 UML1.0.，至此，许多公司开始在实际工作中采用 UML。后来一些其他公司的建议也被吸收进了 UML，1997 年 9 月，发布了 UML1.1 版本，这时 OMG 正式接管并负责 UML 标准的开发。UML1.1 开始成为行业标准。所有的方法竞争才告停止。2000 年，UML1.4 在语义上添加了动作语义的定义，使得 UML 规格说明在计算上更加完整。2005 年，UML2.0 规范形成，定义了许多可视化语法，特别是元模型的定义，至此，代表早期最好思想的、融合的 UML 已经呈现在我们面前。

第二节 UML 介绍

UML 的目标：

UML 的目标是是自己真正成为帮助企业解决实际问题的标准，它不可能满足所有人的所有要求，它只能满足特定的目标，这个主要体现在以下几个方面：

（1） 不用修改可以用在所有领域。
（2） 可视化的表示法和一致严谨的语义非常有利于建模技巧的使用。
（3） 可扩展的特征文件延伸了核心概念的语义。
（4） 独立于开发语言。
（5） 独立于开发过程。
（6） 为理解模型提供文本说明或约束语言。
（7） 与其他面向对象工具结合（而不是纯粹画图）。
（8） 支持框架、模式、构件等新概念的使用。
（9） 支持进程和线程建模。
（10） 可以处理不同抽象层次对象之间的关系。
（11） 更高层次的抽象（如构件）可以使模型在不同环境中实现。

UML 除了以上目标外，能否实现跟上技术的发展，充分发挥其辅助软件开发的目的，也是广大软件开发人员对 UML 的期望目标，比如针对目前常见的一些热点问题，如大规模软件系统、分布式、并发、团队开发等，能够有合适的表示方式。最后，追求简单也是 UML 的另外一个目标，客观世界的业务往往是十分复杂的，如何把复杂的业务用简单的模型表示出来，使建模不至于成为开发工作的额外负担，当然，这也不是件容易的事情，象许多面向对象的语言一样，建立层次化的建模元素包似乎是一种必然趋势，这样使学习者不必全部学习建模语言，而只学习核心部分，对其余部分可以根据需要学习。

UML 模型的意义：

模型图主要有以下三方面的作用，首先是表意，用一套逻辑元素（如类、关联、状态等）表示应用系统的含义，它的规范描述包括句法、规则和执行机制，但只在定义 UML 的标准文献中使用。其次是表示法，可以帮助人们进行浏览、编辑，可视形式展示语义信息，比较直接，符合人们的习惯，另外，模型的布局，不但可以帮助人们理解语义，并且还可以表达另外的语义，并给人以启迪。最后是模型的上下文语境，包括模型内部的组织结构、注释说明、缺省值集合、前提条件等，它们对开发也是非常重要的。

在模型图的概念中，有些概念之间存在着辩证统一的关系，如抽象与具体：建模语言不是开发语言，每种模型都是具体实现在某个精确度层次上的一个抽象，一般情况是分析的前期精度低，后期精度高；规格说明与实现：规格说明只告诉我们做什么，而实现告诉我们怎样做，规格说明是建模的开始，实现是建模的结束，而连接这两点就是模型，它由一系列关系组成，某层次的实现就是下一层次的规格说明；描述与实例：模型是描述性的，它描述一个个实例，但实例只在运行时才存在，某事物是描述性还是实例性的，因观察角度不同而有所不同。

UML 的结构：

UML 从组成结构上讲是由以下三大部分组成，第一部分是指包含 UML 建模的基本元素、关系和图构造块部分，其中基本建模元素是整个模型的基础，有时被称为物件，又可细分为结构性、行为性、分组性、注释性建模元素，关系是说明多个模型元素在语意上的相关性，并可形成更高层次的语意定义，主要用在结构性和分组性的元素之间。可细分为依赖、关联、聚合、组合、包含、泛化、实现。图是指模型视图，从系统的不同侧面讲述软件系统的故事，可细分为类图、复合结构图、组件图、部署图、对象图、包图、活动图、用例图、状态图、交互图（包括顺序图、通信图、交互概览图、时序图）。

第二部分是实现特定目标的 UML 公共方法，公共机制包含规格说明、公共分类、修饰、扩展机制。其中规格说明是模型语意的文本描述，是模型的语意背板，视图是背板

的可视化投影，用语意背板可以保证模型的完整性和一致性。修饰是指 UML 的建模元素在不同的展示场合可以选择不同的表示方式，如类有长格式和短格式，没有必要每次都把图的所有部分都表示出来，使图更容易阅读。 公共分类是另外一种分类方法，包括类元/实例和接口/实现两个公共分类，扩展机制包括约束、构造型、标记值机制。

第三部分是构架，它反映系统的组织结构、包括组成、关联、交互等等，反映在系统中最高级别的概念。包括逻辑视图、进程视图、实现视图、部署视图、用例视图，又被称为 4+1 视图。

UML 语言包：

UML1.4 语言体系由三个包构成，它们是基础包、行为包、管理包。在基础包中主要包括一些抽象类的模型元素核心子包，剪裁现有元素和用途的扩展机制子包，以及元模型中数据类型子包；行为包包括一组核心行为的定义，如：协作、用例、状态机、活动图；管理包包括对包、子系统、模型的定义。

UML 图的分类：

UML 图的分类方法很多，下面介绍另外一种分类方法，它把模型图分成五类，包括是动态行为视图、静态视图、部署视图、模型组织视图、特征描述和设计构造。

动态视图主要是根据对象和外界的交互（用例图），引起对象本身的状态变化（状态图），引起内部对象之间暂时相互通信（顺序图和通信图），最终引起了业务的执行过程发生了变化(活动图)，动态视图主要用来生成程序的函数、进程部分。

静态视图主要定义未来系统应该拥有的关键概念，以及它们的特征和相互关系，如果没有这些概念，将无法形成用户的业务，为实现用户业务的计算机化的软件开发也就无从谈起，静态视图的核心概念是类（类图），它是包含一组信息和实现行为的离散对象（对象图）组成，对象和对象之间固定永久的关系（类关系图），构成了基本的业务式样，这些关系主要包括关联、泛化、依赖。类对外的行为表现是通过类的接口来展示的。静态视图主要用于生成程序的声明、定义部分。

部署视图主要用来描述运行时节点的系统配置和节点上的制品布置情况，软件开发的代码经过打包后，形成多个具有不同意义软件实体（组件图），这些软件实体和硬件、支持环境等资源（部署图）结合起来，用来反映系统运行表现情况。

模型组织视图主要用来处理大型系统的建模要求，对于一个庞大的系统，需要有多个人的团队进行开发，开发人员之间要进行合作，又有分工，需要把系统划分为一个个合适的包（包图），每一个分析人员在一个模型片段内进行编辑、修改、保存模型，各个包之间是采用依赖关系进行连接。另外，由于系统庞大，往往需要从业务角度把系统分成几个部分（构造子系统），子系统之间是通过接口进行连接的。

设计构造主要用来描述有关的设计概念，如把类连接起来实现一定功能的部件，描述对象集合作用的协作等等。特征描述主要用来扩展模型表示范围，弥补模型有限而业务种类无限这个缺陷，既可以用来扩展静态模型也可用来扩展动态模型。扩展的方式主要有三种：

构造型：是用现有的模型元素添加一些约束而形成的一种新的模型元素。

标记值：它可以把一个用户定义的属性，应用到现有的模型元素上（而不是系统对象上）。

约束语言（OCL）：它是一种字符串表达式，一般为特殊目的而开发，并且被保存到模型库中，以供他人使用，但扩展机制要慎用，以防止出现交流困难。

UML 的统一性：

为什么叫统一建模语言呢？它事实上包含了一下的含义，首先，它结束了以前各种建

模思想、方法的竞争时代，并且吸收、融合了各种先进的建模思想，最后才形成的一种统一建模思想和方法。其次它在应用领域方面，不但能用在软件开发领域，完成一些如大型的、复杂的、分布的、计算密集系统的分析设计任务，还可以用在建筑、控制、通信等领域，辅助项目的设计。另外，UML 的统一性还体现在它与所有开发语言都可以紧密结合上，而不是针对象 java、c++几种开发语言，在前期的需求定义和分析模型上，可以做到完全一致，在与语言结合有关的部分稍有差别。最后，在 UML 的内部概念方面，由于这些概念来自不同的研究者，要形成一个体系，这些概念之间就必然要有逻辑联系，UML 在这方面也实现了统一。

第三节　小结

本章首先从 UML 建模的历史出发，介绍了在 UML 建模发展短短 20 年历史上，具有里程碑意义的人物和事件，接着，我们从 UML 的目标、内容、组成等方面对 UML 做了一个简要介绍。最后对 UML 的发展趋势作了一个概括介绍，以便使人们对 UML 建模体系有一个全面的了解。

第四节　习题

1. 在 UML 建模历史上，具有里程碑式事件有哪些？
2. UML 建模由那些视图组成，每个视图包括哪些模型图？
3. 用 UML 模型可以表现什么信息？

第三章 从需求开始

本章主要对 OA 系统的综合行政管理部分的业务需求进行一个概括性的描述，内容包括用户高层需求、系统功能、系统性能以及在系统需求定义阶段需要定义的几个图。本案例贯穿于全书的各个章节，在本章的最后我们对该案例建模过程作一简单介绍。

第一节 系统描述

OA 系统的综合行政管理肩负着企业主要资源的统一管理重任，作为行政管理部门进行日常管理的辅助手段，越来越受到企业的重视。如何利用先进的网络信息技术，构建一种大规模、大范围、规范化、大型的、智能化的网络信息系统，快速、有效、正确、安全和可靠地处理和使用大量的企业资源使用信息，实现信息资源的共享，全方位地提高自身工作效率，是综合行政管理建设的主要目标。

综合行政管理系统能否为其他部门做好服务，最大程度地发挥、利用企业资源的效率，除了依靠行政规章制定外，还必须依赖于企业各职能部门之间、员工与企业之间建立起高效的信息交流渠道以及现代化的管理手段，才能充分利用信息技术，实现考勤管理、会议管理、车辆管理、办公用品管理、图书管理、固定资产管理等多种业务过程的现代化和信息化。

在实际进行系统建模时，需要对系统进行分类，常用的归类方法是分包和分子系统，UML 对这种组织方式并没有作出限制，所以组织的方式有无限多种，系统代表了给定语境中最高级别的事物，而子系统提供了对整个系统的完整的、无交叉的划分，可以说系统是最高层次的子系统，子系统由实现阶段的制品构成。根据本案例，我们把整个综合行政管理系统划分为考勤管理、会议管理、车辆管理、办公用品管理、图书管理、固定资产管理 6 大子系统。

包主要用来进行建模组织，模型包含多个包，每个包里有多个元素，或者说每个元素只能属于一个包，用包把建模元素安排成可作为一个组处理的较大组块，可以控制这些元素的可见性，那些元素是包外可见，那些元素是包内隐藏。把在语意上接近并倾向于一起变化的元素组织在一起，使它们具有很高的凝聚力和宽松的耦合力，为了达到这个目标，首先要检查包中的用例或类是否存在于多个包中，如果回答是肯定的，就应该把它们分解成自己的包，以减少公用性和太多的依赖。并有助于包的独立进化。其次要检查有无为其他包提供基本服务的包，如果有，就把该包作为服务包，并把其他用例或类有类似功能的元素统统集中到这个服务包中，以便让其他包使用该包提供的服务。最后要评估包间的依赖性，使包间的依赖性降到最低。具体做法是进行包转移，即把用例或类关联性强的放在同一个包，以提高凝聚力。

包用一个带标签的文件夹表示，包有简单名称和限定名称，简单由任何数目的字母、数字和某些符号组成，限定包名只是在简单包名前附加外围包名称，并且用双冒号和简单包名称隔开。

包拥有的元素包括类、接口、组件、节点、用例甚至是包，  如果包撤销了，元素也就撤销了，包形成了一个命名空间，这意味着在一个包语境中的同一元素的名称必须唯一，同一个包中不同种类的元素可以有相同的名称，使用包使多个组共同开发成为可能。通过引用可以增加对其他包的访问。

第二节 企业高层需求

4.2.1 客户需求描述

用户高层需求是需求采集首先要调研的信息，它包含着客户（不一定是用户）对未来该软件的期望，与项目的重点、范围以及未来发展（扩展性）和项目的最终验收都有关系。必须严格遵守并贯彻到项目开发自始至终。

OA 系统中的综合行政管理的高层需求如下：

1. 提高行政管理部门的工作效率，进而为提高企业的经济效益服务。
2. 提高企业资源的利用率，使其使用价值最大化。
3. 降低运营成本。
4. 系统易于操作掌握。
5. 提高业务运行的准确性。

4.2.2 客户需求映像

对采集的客户需求进行整理后，要把这个需求映射到未来的综合行政软件系统，形成初步映像，该客户需求的主要映像如下：

1.为综合行政管理人员提供一个好的自动化办公环境，用户通过此系统可以了解完成该任务自己需要了解的任何相关信息，并可以和生产等其他系统实现信息共享。

2.对车辆、图书、会议室、固定资产的使用信息可以查询，行政管理人员可以监督其流转。

3.采用无纸化办公，与现代网络通信相连接。

4.操作尽可能提供模版、向导、帮助，只需简单培训就可以使用系统。

5.每一笔业务的发生，都可以得到相关岗位的审核、控制。

第三节 系统功能

4.3.1 考勤管理

考勤管理是对本企业在册员工进行的一种考核。考勤管理要实现工作时间设定，员工上班和下班的在线登记，同时要实现出差登记，请假登记，外出登记，补签登记，进而实现考勤记录的综合查询。

上下班时间设置，包括设置每天考勤的次数，以及每次考勤的上下班规定时间。可以用上下班的默认时间，也可修改默认时间。另外还可以对特定人设定上下班时间。外出登记要登记员工的姓名，部门，外出开始时间，外出结束时间，以及外出事由。对没有按时登记的人员进行补签，补签时要登记员工姓名，部门，补签日期，补签时间，补签事由。请假登记要登记员工的姓名，部门，请假开始时间，请假结束时间，以及请假事由。出差登记要登记员工的姓名，部门，出差开始时间，出差结束时间，以及出差事由。

考勤综合查询，需要实现出勤查询、请假查询、出差查询、外出查询。对每种查询，要输入部门、查询时段，显示查询结果包括日期，上午上班签到时间和上午下班签到时间，下午上班签到时间和下午下班签到时间，并且实现查询结果的分页显示。

4.3.2 办公用品管理

办公用品是对员工进行日常办公时使用的低值物品的管理。办公用品管理首先要实现物品类别管理和新建物品，其次要实现购买登记，领用登记和在库登记，最后实现领用查询。

物品类别的管理，要实现物品类别的新建、修改、删除、查询功能，并可实现分页显示，物品类别的信息包括类别名称，说明。新建物品要实现新建物品的信息的保存，记录信息包括：名称、类别、单位、规格、备注。购买登记是在新购买的办公用品入库前进行，登记的内容包括：采购人、单价、数量、金额、购买日期、备注，系统要实现输入信息的保存。领用登记要记录领用物品的名称、类别、领用数量、领用人、领用日期，管理员、备注，系统实现输入信息的保存。

领用查询包括按物品查询和按人员查询两种，按物品查询需要在输入物品类别和物品名称后，显示领用的明细信息，包括：领用人、领用时间、领用数量、管理员、备注，并可实现分页显示。按人员查询需要输入部门、人员，显示领用的明细信息，包括：领用人、领用时间、领用数量、管理员、备注，并可实现分页显示。

在库查询，只要输入物品的类别，即可显示该类别下物品的库存数量，显示信息包括：名称、库存数量、单位、规格、备注、所属类别。

购买查询，只要输入物品的类别、物品的名称，就可显示该物品的采购信息，包括：物品名称、类别、采购人、购买日期、数量、单价、金额、备注，并可实现分页显示。

4.3.3 图书管理

图书管理是对企业内部所购图书、杂志的使用管理。图书管理首先要实现图书类别的管理，其次要实现新书登记、借阅登记、归还登记，最后实现图书查询功能。

图书类别的管理，要实现图书类别的新建、修改、删除、查询功能，并可实现分页显示，图书类别的信息包括类别名称、说明。图书登记要实现图书登记信息的保存，保存信息包括：类别、编号、书名、作者、出版社、书号、介绍、购买日期、存放部门、存放位置，并可实现图书的新建修改、删除、查询电子版上传功能。借阅登记包括图书的类别、编号、书名、借阅人、部门、借阅日期、预计归还时间、备注。归还登记包括类别、编号、书名以及实际归还时间。

图书借阅历史查询。通过选择图书类别、图书的名称，系统显示借阅人、借阅日期、借阅管理员、归还日期、归还管理员、备注，并可实现分页显示。

另外，为了提高工作效率，系统还必须实现图书信息的本机导入，以及图书信息的本机导出。

4.3.4 固定资产管理

固定资产是具有较高价值的企业资源，提高固定资产的利用率，对提高企业的效益具

有很重要的意义。固定资产管理包括：固定资产类别管理、折旧类别管理、固定资产登记、固定资产折旧、固定资产查询。

固定资产类别管理，需要实现新建固定资产类别，还可以对固定资产类别进行修改、删除。新建固定资产类别，需要保存的信息包括：资产性质、性质说明。并可对这两个信息进行修改和删除。固定资产折旧类别管理，需要实现新建固定资产折旧类别，还可以对固定资产折旧类别进行修改、删除。新建固定资产折旧类别，需要保存的信息包括：类别名称、类别说明、折旧率、折旧周期。并可对固定资产折旧信息进行修改和删除。固定资产登记，需要保存的信息包括：资产编号、资产名称、购买日期、资产性质、折旧类型、资产原值、资产最低值、开始折旧日期、所属部门、保管人、登记人、登记日期、备注，对登记的固定资产还可以进行修改和删除，删除时要进行确认。系统实现对开始折旧的人工控制。通过选择固定资产，可以查看该资产的折旧信息，显示信息包括：执行日期、执行人、折旧日期、折旧前金额、折旧额、折旧后金额，并可实现分页显示以及固定资产总额的随时估计。

固定资产查询需要输入资产类别，系统会显示该类别下所有的详细信息，包括：资产编号、资产名称、所属部门、资产原值、资产现值。并提供分页显示。

4.3.5 会议管理

会议是保证行政管理实施的手段，会议管理包括会议类别设置、会议室设置、会议申请、会议审核、会议通知、会议纪要、会议查询、会议归档。

会议类型设置是进行会议管理的基础，需要保存的信息包括：会议性质名称、备注，并可对会议类型设置进行修改和删除。会议室设置需要保存的信息包括：会议室名称、容纳人数、会议室资源、使用情况、说明，并可对会议室设置进行修改、删除以及查看使用情况。会议申请是由会议申请人草拟的会议安排，输入信息包括：会议性质、会议议题、预算、会议附件（有附件上传功能）、主持人、记录人员、参加人员、会议地点、会议室、会议开始时间、会议结束时间、会议内容、审批人。可以将会议申请暂存、也可发给审批人或者放弃该申请。会议审核是办公室领导在阅读完申请后签署的修改意见，审核后可以发给办理人，让其发会议通知，或退回给会议申请人，由其发通知，接着由会议起草人起草会议纪要，内容包括：会议名称、纪要内容、附件（有附件上传功能）、记录员、管理员。会议纪要可以提交给会议申请人，由申请人归档或者直接保存。

会议查询包括：已开会仪查询、待开会议查询、会议纪要查询。待开会议查询显示信息包括：会议议题、主持人、地点、时间、与会人员，并可实现分页显示、删除、修改和结束会议。已开会议查询的显示信息和待开会议显示信息相同，可以对其进行删除。会议纪要的查询信息包括：会议名称、会议议题、主持人、开会时间、开会地点、与会人员，可以对会议纪要进行删除和修改和归档。

4.3.6 车辆管理

车辆是企业的重要资源，车辆管理包括;车辆档案、车辆状态、用车申请、派车管理、私车公用、车辆维修。

车辆档案是车辆管理的基础信息，车辆登记的信息包括：车牌号、品牌、车型、颜色状态、驾驶员，可以修改或删除车辆的档案。也可将车辆报废。用车申请是用车人用车前需要保存的信息，包括：用车人、事由、用车性质、开始时间、结束时间、派车人、派车

部门、车牌号、司机。该申请可以发给车辆管理人员，也可保存起来。车辆管理员在浏览完用车申请后，可以选择派车，也可选择无法派车，无法派车的信息返回给用车申请人，如果同意派车，就通知司机，司机出车后，把出车记录交给车辆管理员。另外，车辆管理员可以直接派车，但也要填写用车申请。私车公用需要保存用车信息，这些信息包括：车主、车牌号、开始时间、返还时间、使用部门、使用人、前往地，原因、公里数、过路费、总计、备注。对私车公用可以修改、删除和查询。车辆维修需要记录的信息包括：车牌号、维修时间、维修人、维修内容。

可以对车辆管理中的车辆历史维修进行查询、车辆申请进行查询、车辆使用进行查询。车辆历史维修查询的显示信息包括：维修人、维修时间、维修内容。可以对车辆维修记录进行修改、删除。用车申请查询显示的信息包括：用车人、用车时间、申请时间、用车事由、所用车辆、状态、派车时间，对用车申请可以删除和修改，对车辆使用可以进行查询，需要输入车号、时间段，系统将显示该时间段内使用情况。所有的查询都可分页显示。

第四节　用活动图描述业务流程

4.4.1 活动图的基本概念

（1）活动图是用来描述一系列顺序动作、结果及其它们之间关系的图，主要用来表示系统控制流程和业务处理流程，它重点关注业务过程中的动作和结果。在软件开发中，活动图主要用来和用户交流，以辅助需求采集。有时候，可以把活动图看成一种特殊的状态图。活动可以绑定到任何建模元素，用来反映该元素的行为，该元素提供该活动的语境，活动绑定的元素有;用例、类、接口、组件、协作、操作，但在早期建模时不需要明确是哪个元素。活动图在使用时要注意目标用户，力求简单。活动图元素包括活动、转移、分支、分叉、合并、汇合、泳道、对象流。活动图的典型应用包括：用例中控制流和用例间的控制流，操作和算法的细节以及业务流程

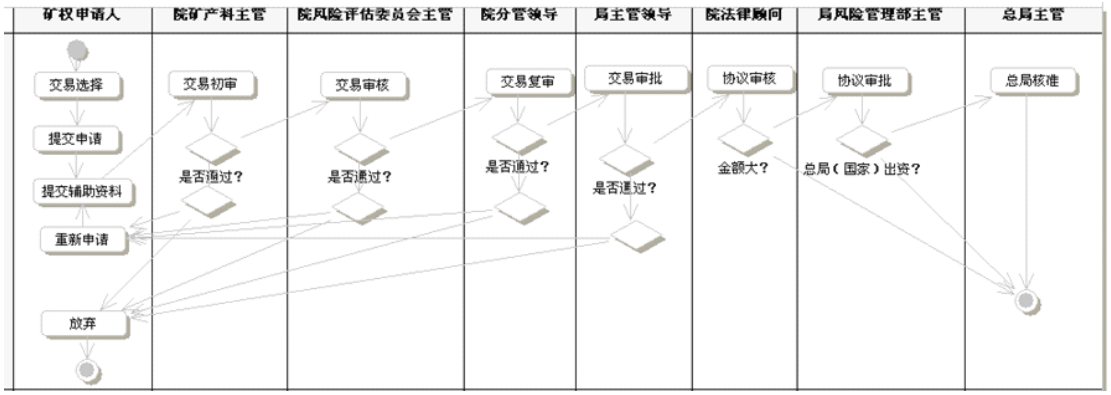


图 4.1　活动图表示实例

（2）活动表示业务过程中的原子动作或操作步骤片段，具有可分解性。活动又叫活动节点，主要分为三类：表示具体工作单元的动作节点、表示控制流的控制节点以及活动中使用的对象节点。在模型图中用一个圆角矩形来表示活动节点（和状态图中状态表示相同）。另外，活动图有起点和结束点，起点用实心圆表示，终点用一个圆圈中包含一个实心圆（俗称公牛眼）表示。

（a）活动起始点表示法 （b）活动起始点表示法 （c）活动节点表示法

图 4.2 起始节点、结束节点以及活动节点的表示法

（3）系统状态由代表控制流（或数据流）的令牌位置来确定，令牌沿着源节点向目标节点转移常常是有约束的。转移表示活动节点之间的过渡，转移用一个带箭头的直线表示，箭头指向后一个活动节点，转移发生可以是有条件的（称为监护条件）和无条件的，无条件的转移是指活动节点中的每一个动作都执行完成以后自动向后转移；有条件转移指当条件为真时才发生转移。

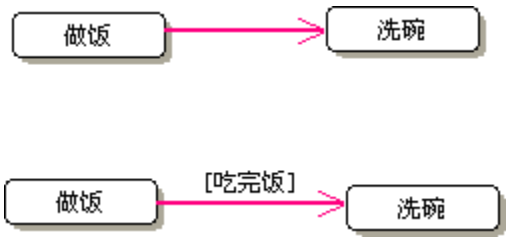


图 4.3 无条件转移和有条件转移

（4）比如你从你家出来去钟楼，既可以坐车走大街，也可以步行走小巷，你不能同时选两条路径，离开你家就发生了分支，到了钟楼就发生合并。分支表示在活动图中的决策节点，用一个空心菱形表示，有一个输入，多个输出，在每个输出上标有转移发生的条件，这些条件之间是互斥的，并且是由活动本身提供的，只有条件为真时（必须有一个为真）才发生转移。在分支结束时可以合并，，也用一个空心菱形表示，有两个或多个输入，一个输出，没有监护条件。

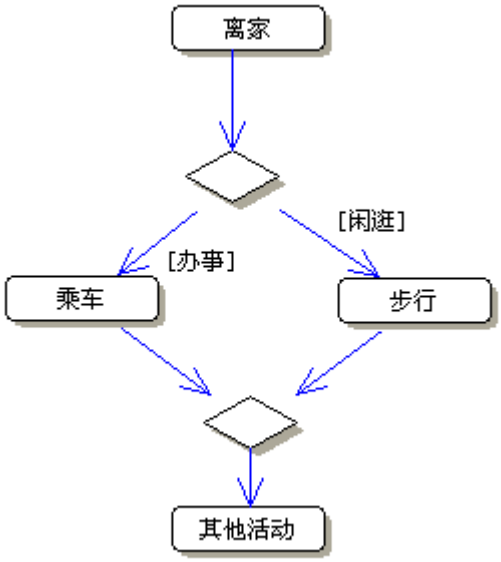


图 4.4 分支和合并的表示法

（5）分叉表示在活动图中并发执行的动作，一般表示单个过程启发了多个并发线程或进程，用一条粗线表示转移转换为并行动作，这条粗线可以垂直放置也可水平放置，有一个输入多个输出，每个输出独立执行。多个并发分支需要同步时，用汇合表示，汇合也是用一条粗线表示，它有多个输入一个输出，只有当多个输入都到达同步时，才能发生汇合，并输出一个控制流。

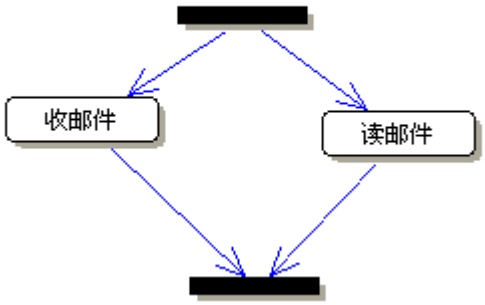


图 4.5 分叉和合并的表示法

（6）泳道又叫活动分区，它的作用是把活动和它的执行者联系起来，一般是针对跨越两个以上的执行者的业务活动（如果不这样做，也可以在活动节点中用小括号注明执行者），负责该泳道的对象放在该泳道的顶部，该对象下面放置该对象的活动小组。

（7）对象流用来表示动作和对象之间的依赖关系，用一条带箭头的虚线来表示，对象指受活动节点影响的事物，主要用来表示数据存储，对象用一个矩形来表示，最上面为构造型，中间为对象的名称，下面为对象的状态。

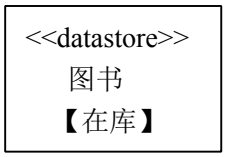


图 4.6 对象流的表示法

4.4.2 活动图建模的步骤

（1） 在采集的原始需求中选择重点流程
（2） 首先要确定要设计的活动图是针对业务流程还是用例。
（3） 其次要设计活动过程的起点和终点。
（4） 确定活动图所有执行对象。
（5） 确定活动节点，并根据执行对象进行活动分组。
　（a） 如果对用例建活动图，则把角色所发出的每一个动作变为活动节点。
　（b） 如果对业务流程建活动图，则把每一个流程步骤(或片段)变为活动节点。
（6） 确定活动节点之间转移。
（7） 处理在活动节点之间的分支和合并。
（8） 处理在活动节点之间的分叉和汇合
（9） 用 UML 建模工具进行活动图建模。
（10） 编写必要的补充文档。

4.4.3 活动图建模分析过程

当你拿到一个项目时，首先面临的问题是对那些问题进行活动图建模，根据前面的原则我们知道要选择重点流程，同时，活动图建模的对象可以是流程级（需要两个以上岗位相互配合才能完成，上一个岗位的输出是下一个岗位的输入，并且两者之间具有逻辑关系，属于多个用例结合），也可以是用例级（只要一个角色使用系统就可以完成），对于会议管理子系统来讲，我们选择能够贯穿日常业务全过程，有多个角色参加，在会议管理中必须执行的业务，作为我们的重点流程，会议申请流程作为我们的首选流程，这是因为会议管理的主要对象是“会议”，我们关心的是会议对象的生成、获得、使用、消亡的全过程，会议对象的生成就是由会议申请人进行的会议申请，会议对象的获得就是经过办公室主任审批通过的会议对象，会议对象的使用包括会议通知、会议执行、会议纪要等活动，由其他角色去执行，会议对象的消亡就是会议执行完后的归档。在这里要注意，我们的会议管理流程不是到申请被批准以后就结束，而是关注会议的全过程。

活动的起点选在那里呢，由会议申请人起草会议申请作为起点是最好的选择，当然有些临时会议的举行可能有另外的发起点，但这不是我们重点关注的目标，会议结束点可能有多个，如会议正常归档后结束、会议审批未通过而放弃结束、会议申请暂时保存而结束，会议因其他原因取消使会议结束等等，是不是活动图把这些结束点都要画出来呢，回答是否定的，活动图只画基本的、正常流程活动。对无数个异常活动没有必要，也无法完全画出来。

接着我们要确定会议的执行对象，在会议管理中，有多个对象要参与会议的执行，他们是：会议起草人要起草会议申请，办公室主任要审批会议申请，会议办理人要通知会议、相关人员要布置会议室、领导要主持会议、参会人员要接收会议通知并且发言，会议纪要人员要记录会议内容，清洁人员在会后要清洁会议室……。是不是把所有人员的工作都画进流程，显然不是，这里有一个原则，就是要确定那些人员未来使用该软件系统完成前面这些工作，并且软件公司打算为该工作开发软件，那么这些人就是会议流程的执行对象。

在确定了该活动的执行对象以后，就需要对每个执行对象逐个确定在本活动图中所从事的活动，活动可以是一个步骤片段，也可以是一个动作，这两种形式可以在一张图中并存，比如起草会议申请就是一个操作步骤片段，它包括输入、编辑、修改、预览等多个动作，而提交会议申请，只是点击提交这一个动作。在这里要注意，所有的活动都要和未来实现的软件系统有关，负责就会变成普通的业务流程图。

前面我们提到，活动和活动之间的关系是有逻辑性的，有的是无条件顺序执行的，即在上一个活动执行完后，就自动执行下一个活动，如办公室主任在阅读完会议申请后，正常情况下，就会签署审批意见，就是从上一个活动自动转移到下一个活动；有的活动是在满足一定条件后，才从上一个活动转移到下一个活动，比如办公室主任在审批后，如果认为需要修改会议申请，就转移到会议申请人下的修改会议申请活动，否则，就转移到会议办理人下的发会议通知活动；有些活动后，会出现两个同时执行的活动，比如在会议申请审批通过后，在会议办理人通知会议的同时，其他人员要布置会议室、主持人同时要起草会议提纲，由于后两个活动不打算为其设计软件，所以在活动图就没有被画出。

4.4.4 元素识别结果

（1） 重点流程选择：会议申请流程

（2） 用例还是业务流程：业务流程

（3） 活动图的执行对象：会议申请人、办公室主任、会议办理人，纪要起草人

（4） 会议申请人的活动节点包括：起草会议申请、修改会议申请、完成会议申请拟稿、提交会议申请、暂存会议申请、会议纪要归档。办公室主任的活动节点包括：阅读会议申请、签署修改意见。会议办理人的活动节点包括：发会议通知、发会议通知完成。纪要起草人的活动节点包括：起草会议纪要、修改会议纪要、会议纪要完成、发送会议纪要、暂存会议纪要。

（5） 识别的分支合并包括：会议申请人拟稿完成后选择是暂存还是立即提交申请；办公室主任审核后选择是交会议办理人办理还是退回给会议拟稿人；会议纪要起草人在起草完会议纪要以后，是暂存还是发给会议申请人，由会议申请人归档，另外会议申请人选择起草会议申请还是修改会议申请，以及纪要起草人的活动起草会议纪要还是修改会议纪要，都是分支和合并节点。并且以上分支的监护条件都是互斥的。

4.4.5 活动图

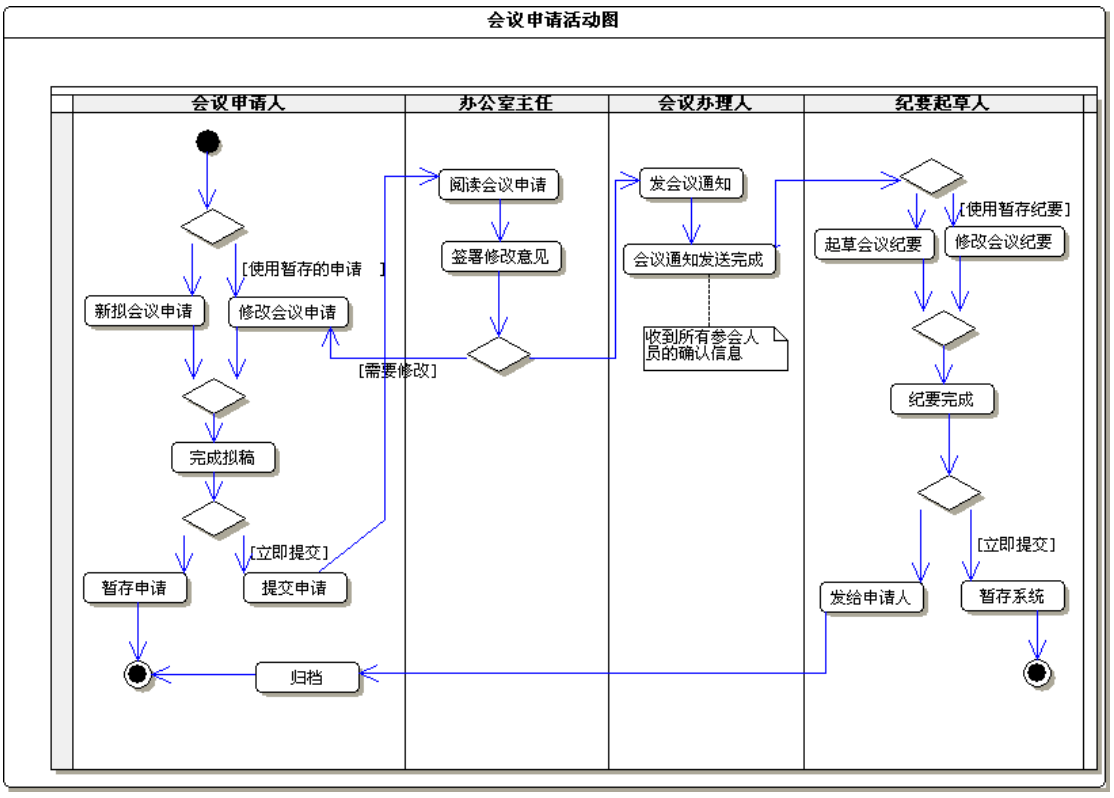


图 4.7 会议申请活动图

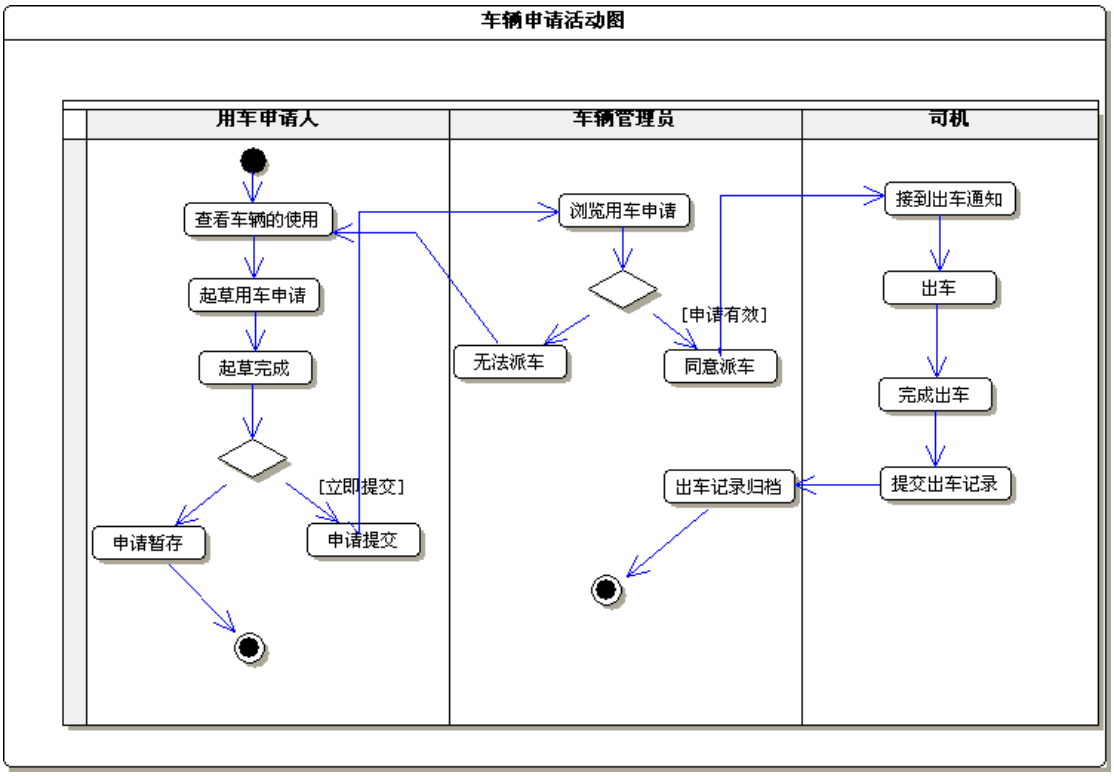


图 4.8 车辆申请活动图

4.4.6 操作步骤

为了用 trufun 的 UML 建模工具进行具体建模，首先我们要建立项目，其次要建立包和子包，以便对模型进行分类管理，最后在每个子包下，建立每个具体的模型图。下面我们以会议申请活动图为例说明活动图的画法：

一、综合行政管理项目创建步骤

1. 双击击桌面的 trufun 建模工具快捷方式图标，系统显示 trufun 主界面。

图 4.9 trufun 建模工具快捷方式图标

2. 点击“文件“菜单下的”新“子菜单，再在弹出菜单中选择”项目“，系统显示新建项目界面。

图 4.10 新建项目操作菜单

3. 在向导列表框中选择“新建 UML2.x 项目“，然后点击”下一步“，系统显示选择项目类型界面。

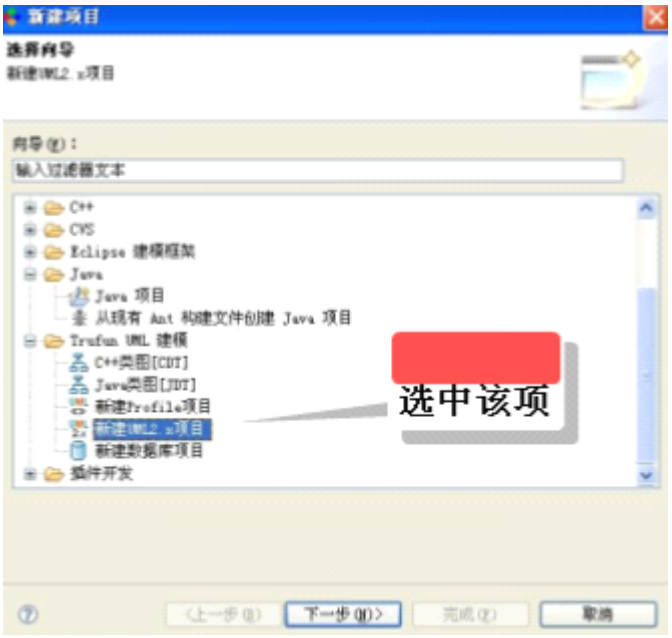


图 4.11 新建项目导航

4. 在项目类型的下拉列表框中选择“UML“，然后点击”下一步“，系统显示创建 UML 项目界面。

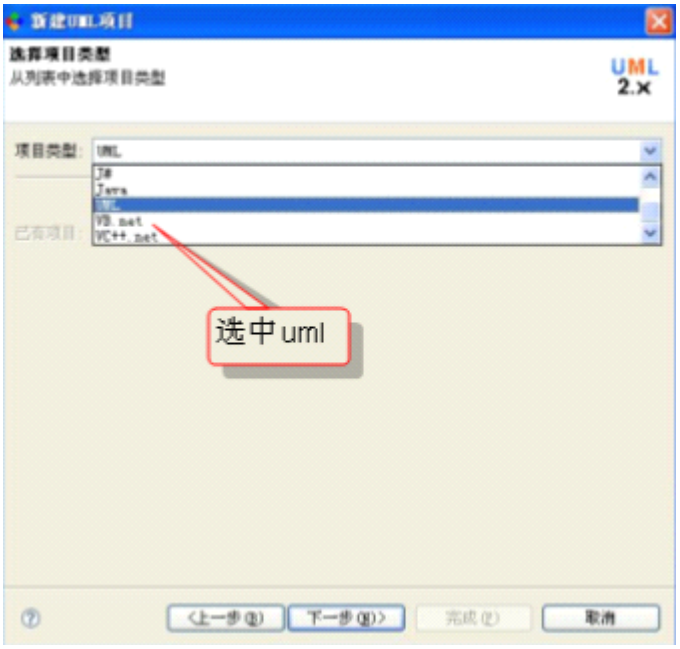


图 4.12 选择项目类型操作界面

5. 在项目名称文本框中输入“综合行政管理“，存储位置选择”使用缺省位置“，系统返回到主界面。

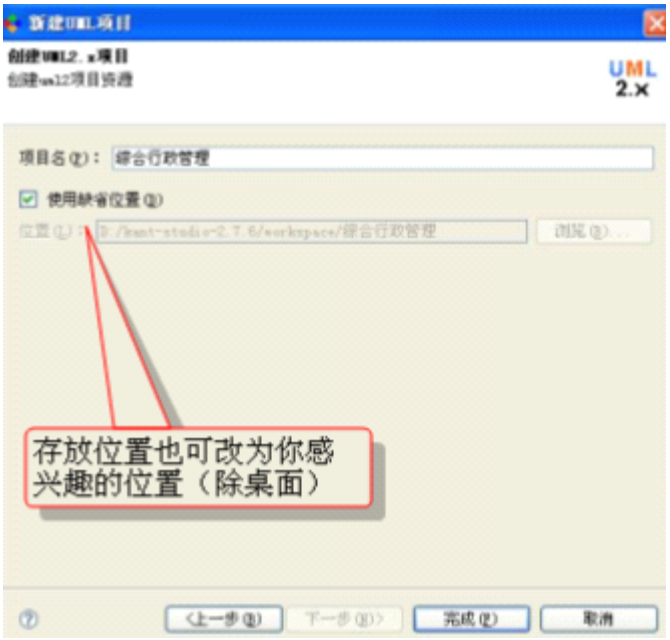


图 4.13 项目的存储位置选择

二、包的创建步骤

1. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。

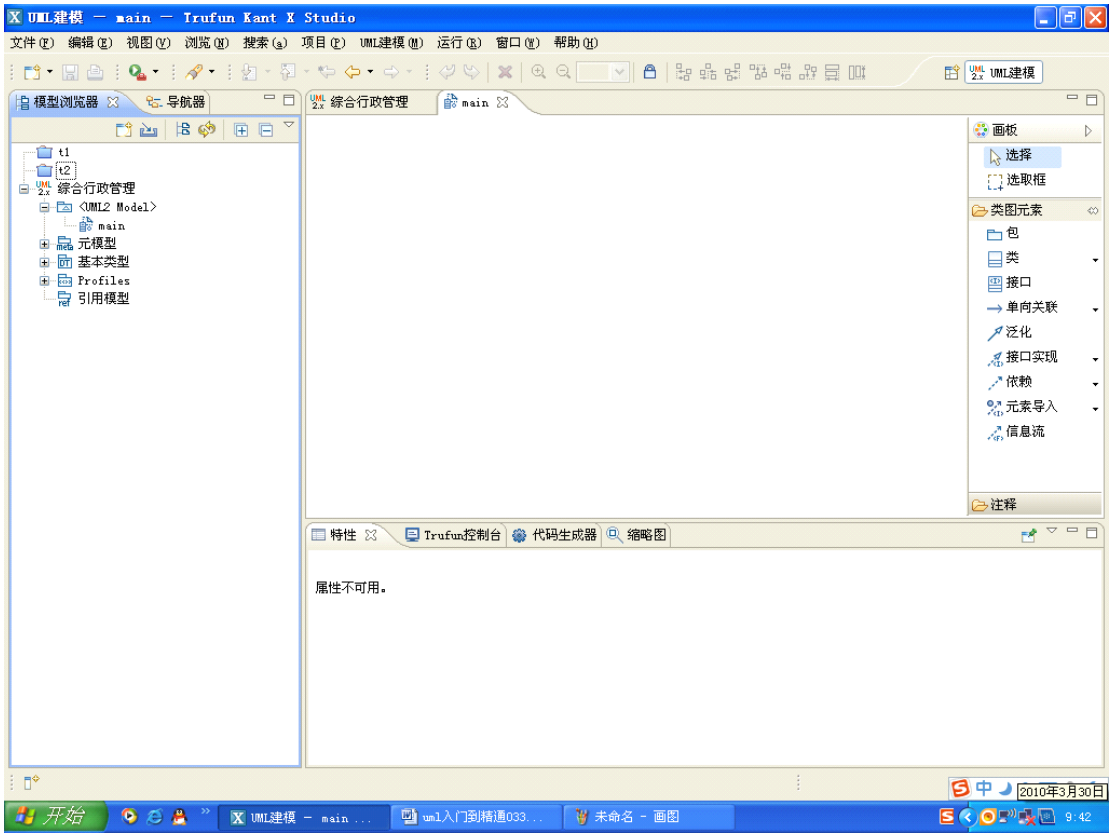


图 4.14 系统主界面

2. 在“<UML2 Model>“节点上右击，从弹出菜单中选择”新建“，再在”新建“菜单下选择”包“，系统在”<UML2 Model>“节点下新增一个以 package

开头包。

图 4.15 新建包操作界面

3. 选中该以 package 开头，在界面下面的属性选项卡中修改别名（注意已选择后面的”显示别名“，以后不再说明），输入“会议管理”。

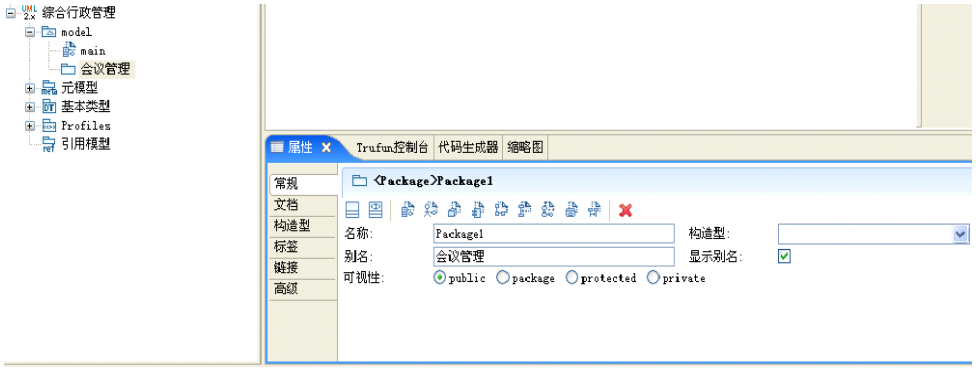


图 4.16 会议管理包的属性界面

4. 重复步骤 2、3。完成车辆管理、图书管理、办公用品管理、固定资产管理、考勤管理包的创建。
5. 右击“会议管理”包，在弹出菜单中选择”新建“，再在”新建“菜单下选择”包“，系统在”<UML2 Model>“节点下新增一个以 package 开头包。

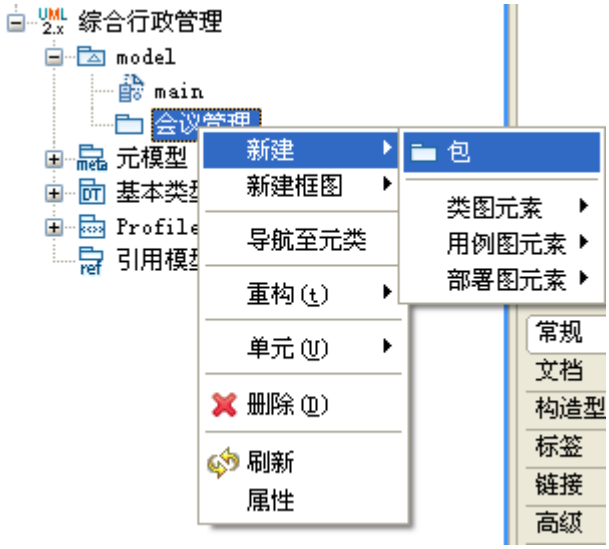


图 4.17 在包下面再建子包的操作界面

6. 选中该以 package 开头，在界面下面的属性选项卡中修改别名（注意已选择后面的”显示别名），输入“需求定义”。

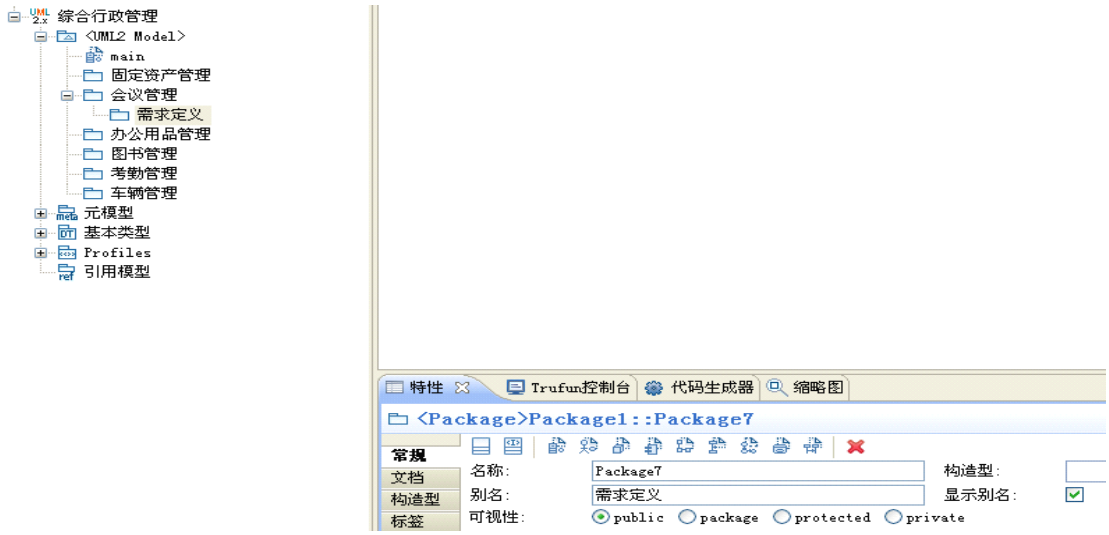


图 4.18 子包的属性定义界面

7. 重复步骤 5、6。完成业务建模、动态建模、设计建模包的创建。
8. 重复步骤 5、6、7。完成车辆管理、图书管理、办公用品管理、固定资产管理 、考勤管理包下的需求定义、业务建模、动态建模、设计建模包的创建。

三、活动图建模

1. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
2. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“会议管理”前面的“+”符号。

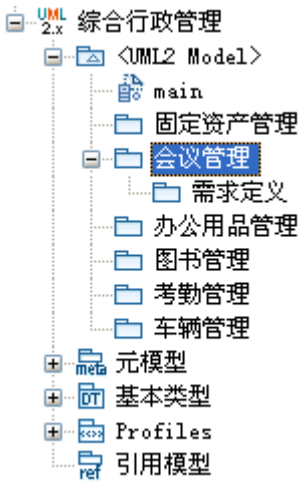


图 4.19 模型导航栏操作界面

3. 右击展开导航树上的“需求定义“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”活动图“。在活动图上增加了两个节点，一个是以”actividiagram“开头的节点，代表这个活动图全局属性，并可在属性选项卡中修改该节点名称为“活动图”。另一个节点以“activity”开头，在导航树中选择该节点，在属性选项卡中输入别名“会议申请活动图”。

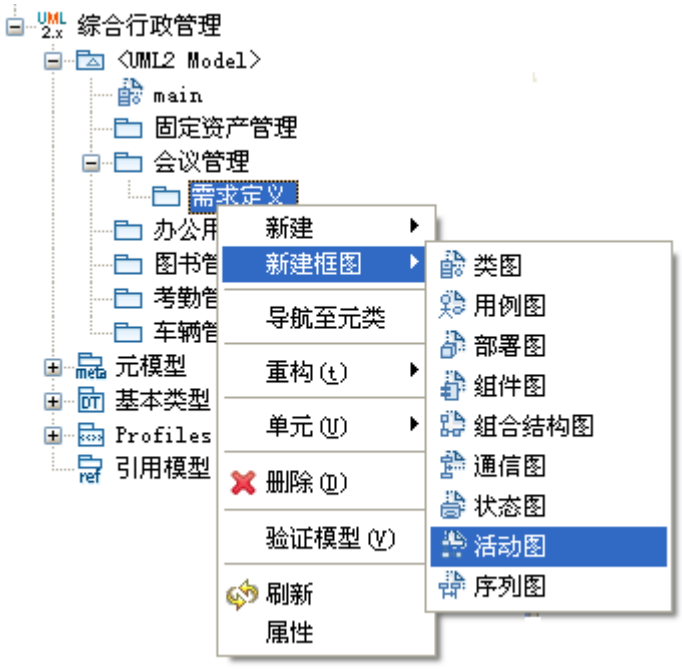


图 4.20 建立活动图操作界面

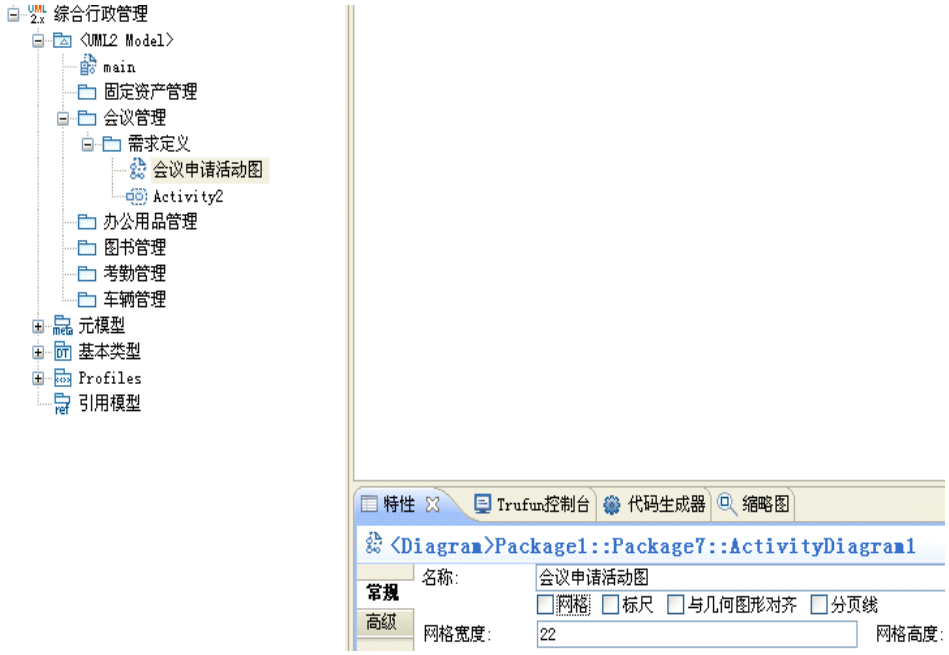


图 4.21 活动图的属性界面

4. 在建模绘图区，选中缺省的活动图，拖动四周实心矩形锚点，把绘图区域调整到适当大小。

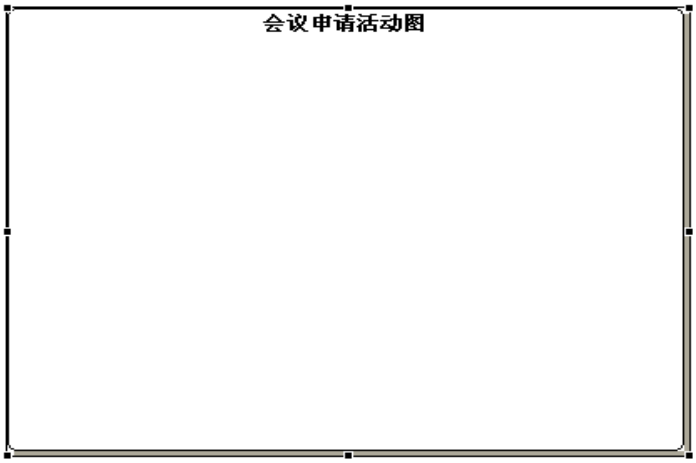


图 4.22 绘图区域大小调整

5. 从绘图工具面板选中分区元素，并将其拖入建模绘图区，并适当调整大小。

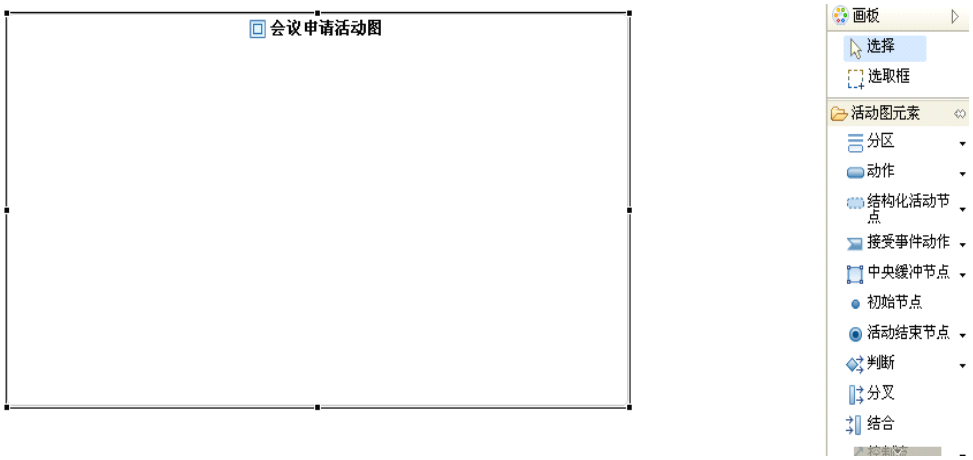


图 4.23 分区元素以及调整分区大小示意图

6. 在分区元素上右击，在弹出菜单中选择“布局”,并在”布局”菜单下选择“列表“，然后在分区上右击，在弹出菜单中选择”表格“，并在”表格“菜单下选择“新增列”，如此反复新增四列，适当调整各列的宽度，新增的列即为泳道。

图 4.24 增加新分区操作界面

7. 选择第一泳道顶部的矩形区域，在属性选项卡中的别名中输入“会议申请人”，同理可为办公室主任、会议办理人、纪要起草人设置相应的泳道。

图 4.25 泳道负责人设置

8. 从绘图工具面板选中起始节点和结束节点元素，将其拖入会议申请人所负责的泳道。

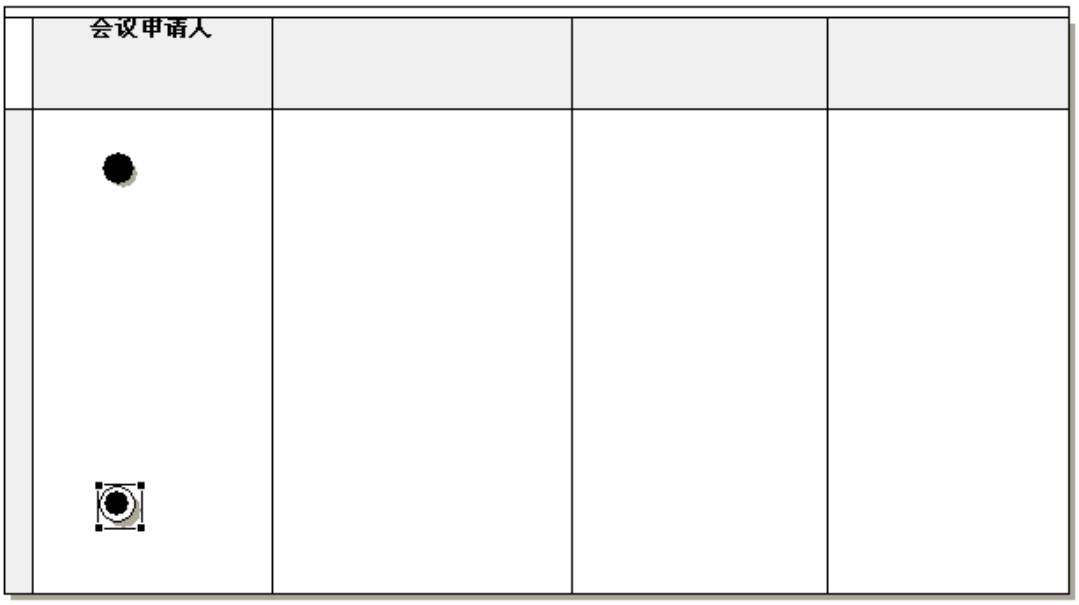


图 4.26 起始活动和结束活动操作界面

9. 从绘图工具面板选中动作元素，将其拖入会议申请人所负责的泳道，在属性选项卡中输入别名“起草会议申请”，同理可创建会议申请人所负责的其他活

动节点。

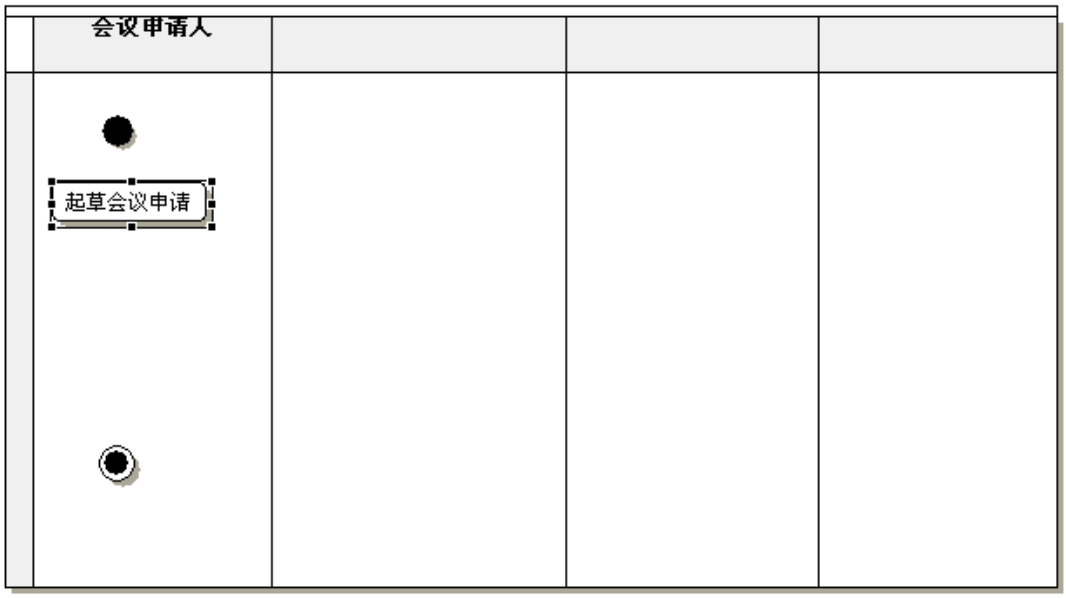


图 4.27 状态元素示例

10. 对办公室主任、会议办理人、纪要起草人也采用同样的方法创建活动节点。
11. 从绘图工具面板选中分支以及合并元素，将其拖入会议申请人所负责的泳道，从绘图工具面板选中转移元素，将其拖入会议申请人所负责的泳道，连接分支、合并元素和活动节点，同时选中分支转移元素，在属性选项卡中的监护条件下加入不同的监护条件。以此类推完成其他泳道的分支、合并元素的建模。
12. 从绘图工具面板选中转移元素，将其拖入会议申请人所负责的泳道，进行剩余活动节点连接。
13. 从绘图工具面板选中注释元素，将其拖入会议办理人所负责的泳道，再选择活动连接元素把注释元素和完成会议通知发送活动节点连接，然后填写注释信息。
14. 审核无误后保存。

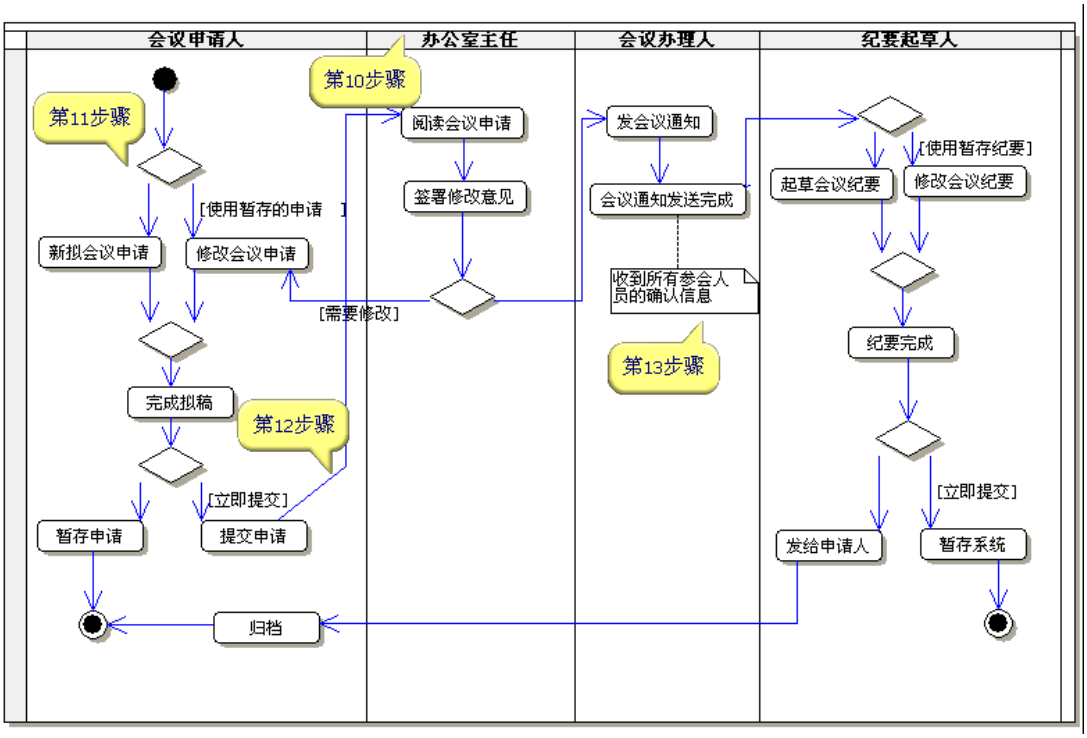


图 4.28 活动图建模 10-13 步骤

第五节 系统性能

1. 系统响应时间一般小于 3 秒，复杂数据查询小于 5 秒
2. 系统最大并发用户数 250，同时在线用户 1000.
3. 系统总体可用率 99.8%
4. 界面友好，易于操作。
5. 对敏感信息加密，并且可监控系统的运行。

第六节 建模过程

本书建模过程是依据 trufun 的 tup 建模过程进行(相见第 18 章)，在需求定义阶段主要建立主要业务流程活动图模型（在本书的第四章完成），系统用例图模型（在本书的第五章前半部分）；业务建模主要建立静态类图模型（在本书的第六章完成），静态类关系模型（在本书第七章完成），动态分析阶段首先要建立低级用例模型（在本书的第五章后半部分完成），顺序模型和通信模型（在本书的第八章完成），状态模型（在本书的第九章完成），实现阶段完成组件图和部署图建模（在本书第十章完成），为了避免案例单一，在不同的章节选用不同的案例进行操作描述。

第七节 小结

需求采集阶段获取的原始需求资料，是进行 UML 建模的基础和前提条件，本章一开始对本书采用的案例的原始需求资料，从用户高层需求、系统功能、系统性能几个方面进行了介绍。依据原始需求，我们进行了活动图的建模，在需求采集阶段使用活动图，主要是为了下一步和用户交流之用（需求采集的结果最少也要和用户交流一次）。在活动图建模中，首先介绍了活动图的模型元素和语义，活动图建模步骤，最后对会议申请活动图的建模元素、建模操作过程进行了详细介绍，读者只要按步骤操作，就可完成活动图建模。

第八节 习题

1. 简述活动图的建模步骤。
2. 活动图建模的主要元素有哪些？，它们的语义和表示图符是什么？
3. 对图书管理系统的图书借阅过程建立活动图模型。

第四章 建立用例模型

上一章我们介绍了本书案例的需求，并建立了活动图模型，本章我们将继续进行需求定义阶段建模，首先是系统用例，它主要用来帮助我们确定未来系统范围（边界），哪些岗位（角色）将使用系统，他们希望系统提供哪些服务，他们需要为系统提供哪些服务，他们用此系统完成哪些方面的用户需求（顶级用例）。用例主要由 Ivar Jacobson 创造的，1996 被并入 UML，用例的表示方法包括用例图、用例的文本描述及脚本，文本描述是强调需求的细节，脚本是表示用例执行中的选项，一般用活动图表示。用例图主要用来描述企业中的系统是如何组合的；某个子系统或构件是怎样实现的；需求的范围在哪里等问题。为了保证建模过程的逻辑连贯性，下面我们对分析阶段的业务用例建模进行了介绍。最后对用例的描述文档格式进行了介绍。

第一节 用例模型概述

软件开发的最终产品是由用户（不同于上一章的客户）来使用。从用户角度观察需求，能够确保最终开发的软件产品能给用户提供所要求的服务。用例模型主要用来描述系统和系统外部环境的关系，直接影响着其他模型。用例是一组系统使用场景的集合，每个场景又是由一些事件序列构成，发起这个事件的用户就是系统使用的参与者。用例图是系统的高层描述，角色和用例，在实现阶段则变成了对象和接口这样的底层描述。用例可以帮助每一用户知道他们未来怎样使用系统。对开发者来讲，在不关注细节的情况下，可以快速搜集系统需求，形成总体样式。除了用文档描述用例以外，用上一章学过的活动图也可描述用例。

5.1.1 用例的模型元素

用例模型元素包括：系统（或主题）、角色（或活动者）、用例以及各种关系，下面我将逐一介绍它们的语义和表示法。

（1） 系统：概括地讲，系统是指分析设计阶段所涉及，实现阶段要实现的所有元素的集合。从用例建模角度来看，系统就是提供用例实现的“黑匣子”。系统的边界，就是软件开发的范围。但系统不单指“软件系统”。系统的确定，关键是确定需求资料中那些工作由未来的计算机系统完成，那些还是沿用手工方式。系统用一个矩形盒子表示，顶部有系统的名称。

图 5.1 系统元素表示法

（2） 角色：是指和未来所建系统交互的人和物，也叫参与者，通俗地讲，就是和系统打交道的人、系统、设备、企业。角色可以是人，也可以是和系统交互的其他系统，也可以是时间(定时发生)。角色向系统发消息，系统向角色返回消息，角色指人时是指一类人，而不是一个，一般用一个小人符号表示，角色是其他类型时，一般使用带有《actor》原型的矩形表示。在实际问题中，一个人可以有多种角色，一个角色可以有多个人。

图 5.2 角色的表示法

（3） 用例：系统执行的一组动作序列，并为执行者产生一个可供观察的结果，这个结果对系统的一个或多个参与者是有价值的，从系统角度看，用例定义了系统（或者企业、子系统等）的行为特征，缺少这些特征，系统就不能工作，用例侧重于目标，而不是内部处理。用例总有一个启动者，用例只有得到最终结果（或返回结果）才能结束。用例用动名词短语的业务术语命名。用例是系统功能的总体描述。用例的实例就是场景，场景是用例的一条特定执行路径。路径一般有基本路径、可选路径、异常路径。用例的表示符号为一个椭圆。下面是用例的名称（也可以放在椭圆中间）。

图 5.3 用例的表示法

（4） 关系：关系反应了参与者和用例之间,用例和用例之间以及参与者和参与者之间的相互作用，用例和参与者之间是关联关系，一般角色为用例的启动者，用单向关联，否则为双向关联。角色和角色之间如果存在一般和部分的关系，可以用泛化来表示。

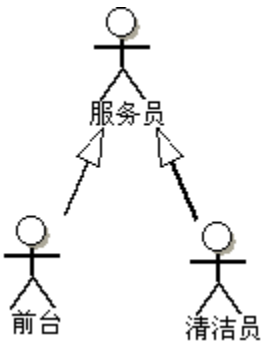


图 5.4 角色的泛化关系示例

用例之间的关系有三种：

（a） 如果一个用例包含另一个用例的行为，则这两个用例之间存在包含关系，或者说一个用例使用了另一个用例的行为，被调用用例可以是事先定义好的，也可以是在各种环境下使用的公共行为，调用是无条件的。包含关系用一个虚箭头表示，从调用用例指向被调用用例。

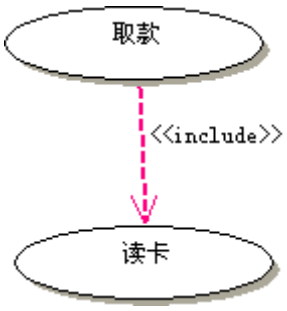


图 5.5 用例之间的包含关系示例

（b） 如果对一个用例的行为添加某些额外的行为，包含此额外行为的用例和原来的用例之间就构成了扩展关系，扩展用例本身提供了一个离散的行为，可以把自己添加到执行用例的环境中去，增强了执行用例的行为，等于在不改变执行用例的情况下，给系统添加新的行为。扩展用例的执行是有条件的，这个条件叫扩展点，可以用注释在图中表示，也可以在用例描述中加以说明。扩展关系用虚箭头表示，箭尾在扩展用例上，箭头在执行用例上。

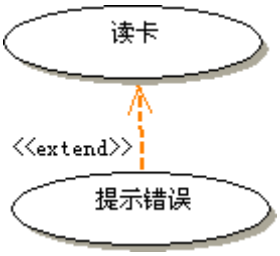


图 5.5 用例之间的扩展关系示例

（c） 如果一个用例是另一个用例的特例，这两个用例之间就是泛化关系，父用例不具备完整的事件流，不具备执行能力，而子用例实现其高级行为。这种关系表示在语义上比较困难，UML2.0 对此又没有明确的描述定义，在实际中这种关系使用非常少，甚至很多书连提都不提这种关系。

5.1.2 用例图建模分析步骤

（1）首先要确定将要设计的系统和它的边界。
（2）其次要确定系统外的活动者。
（3）从活动者（用户）和系统对话的角度继续寻找一下两方面的特征：
（a）寻找活动者怎样使用系统。
（b）系统向活动者提供什么样的功能。
（4）把离用户最近（接口）的用例作为顶级用例。

（5）对复杂的用例做进一步分解，并确定低级用例以及用例间的关系。
（6）对每一用例做进一步细化。
（7）寻找每一个用例发生的前提条件和发生后对系统产生的结果。
（8）寻找每一个用例在正常条件下的执行过程。
（9）寻找每一个用例在非正常条件下的执行过程。
（10）用 UML 建模工具画出用例模型图。
（11）编写用例模型图的补充说明文档

5.1.3 寻找用例的方法

（1） 从原始需求中所包含的功能中寻找。
（2） 未来的系统，需要和那些系统要发生信息交互？
（3） 那些人将操作未来的软件系统？
（4） 系统有那些受限的条件？
（5） 未来的软件界面怎样组织？
（6） 系统的响应有哪些去向？

第二节 系统用例模型

系统用例建模主要在需求采集阶段进行，其目的为了获取需求的范围和层次，以确定系统支持什么，不支持什么。如果分阶段开发，可以帮助确定每个阶段的目标。另外，系统用例模型还可以确定未来系统与其他企业、系统、构件或用户接口，并为未来软件测试和验收提供框架。系统用例建模是以确定未来系统目标为主要任务，这样做就可以避开在项目的开始阶段讨论具体实现方法的问题，有利于以后考虑多种实现方法，并从中选取最合适的方法。

5.2.1 会议管理系统用例图分析过程

对于一个新项目，系统需求中的每一个名词，在未来系统中，可能为一个系统、一个子系统、一个类、一个对象、一个属性，甚至不作处理，有没有一个标准用来确定一个名词在未来系统中属于哪一个范畴呢？回答是没有，它的选择完全取决于具体业务，取决于未来软件系统的范围和边界，在确定这个问题前，需要回答一下三个问题，第一是那些人或系统要使用未来的软件系统（系统外角色），第二是这些人用此软件系统能够完成哪些有经济意义的活动（用例），第三是未来的软件系统在多大程度上满足这些人的要求（处理粒度）。对于会议管理子系统来讲，会议申请人、办公室主任、会议办理人、纪要起草人员、参会人员、办公室其他人员、会议室布置人员都是未来可能使用软件系统的人员，其次，这些人中那些人必须使用系统，才能完成工作，才能获得有经济意义的结果呢？ 显然，办公室其他人员、会议室布置人员不满足此条件，不使用该系统，他们也能完成自己有意义的工作，这就好比街上发生了一件交通事件，交警在现场看，围观者也在现场看，但对交警来说是有经济意义的，但对围观者来说，看上一天也没有经济意义。只有使用有经济意义的用例的角色，才是有效角色。最后是要确定系统处理此问题的详细程度， 如果对该名

词没有其他动词作用于上，使用的概念范畴就小，反之，动词的作用层次越多使用的概念范畴就越大。在会议管理中，管理会议作为子系统，会议作为类、会议主持人作为属性，会议布置人员不作处理，一般来讲，对概念范畴大的名称施加操作，就是顶级用例，在图中离用户近，反之为低级用例，在图中它离用户远。

在明确了以上几点后，就能很快找到系统的边界，边界外的角色、以及顶级用例和低级用例，要注意，这几个对象在寻找时具有概念上的相关性。在进行顶级用例建模时，我们还需要寻找角色和角色之间的关系，以及角色和顶级用例之间的关系。会议申请者，办公室主任，会议办理者，纪要起草人，参会者这些会议管理对象，从表面上看，可以抽象出一个会议人员角色，他们之间构成泛化关系，但由于这几个角色对未来系统的操作差异太大，在系统实现后映射到系统内部，使用泛化关系带来的收益较小，所以使用几个独立的角色，没有建立泛化关系。角色和顶级用用例之间的关系。是根据信息流动的方向不同而有所不同，分为单向关联和双向关联，比如会议申请人和编辑会议申请用例之间就是单向关联，会议申请对象中包含了会议申请人对象的信息，而会议申请人对象中没有记录所申请的会议信息。双向关联是指角色和用例中都包含了对方发来的信息，比如办公室主任角色和分配会议室资源用例之间就是双向关联，办公室主任在会议室分配时，系统能显示他所能分配的会议室信息，被分配的会议室对象中也记录了分配人的信息。

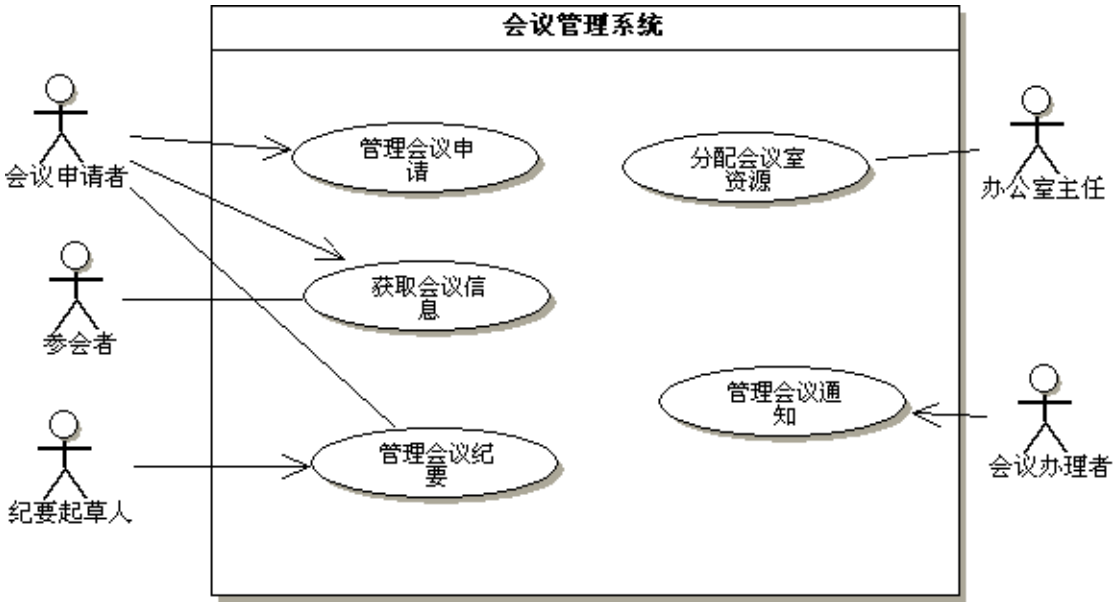
在进行顶级用例建模画图完成以后，还要进行描述文档的编写，文档主要用来弥补图形表示的不足和增强系统的完整性而编写的，该文档是一个格式文本，通常包括的内容有:用例名称、参与者、前置条件、后置条件、基本事件流、备用事件流、异常事件流。比如起草会议申请用例的用例名称为起草会议申请。参与者也就是角色是会议申请人，前置条件是指执行该用例应该具备的条件，如会议申请人有条件通过网络访问系统并已成功地登录系统，如果无法上网或无法登录系统，都说明不具备条件；后置条件是系统执行完用例后的结果或变化，起草会议申请结果就是系统保存一份新的会议申请，该申请可能在暂存,或者被发到了办公室主任可以浏览的位置。基本事件流是正常情况下的操作流程，系统分析人员在分析后根据以往项目的经验进行编写，和操作手册的写法相同。可选事件流也是一种正常的业务流程，只是该流程的执行是有条件，比如到商场买东西，正常情况下是付现金，但有位顾客拿了张支票，系统就要执行支票购买可选事件流。异常事件流是由于错误而执行的错误处理流程，我们知道错误的路径有无数条，没有必要和可能把每条路径都写出来，以主要错误处理流程为主，当然，如果有可能，可以编写一个通用的错误流程，以避免系统进入一个无法预知的流程，当然，异常流程的编写也是根据以往的项目经验。

5.2.2 会议管理系统用例图模型元素识别结果

1. 角色:角色识别是整个用例建模的第一步，那些人和事物能成为角色，首先要看它是否要使用未来的系统，和系统发生交互行为，再者要看它使用未来的系统是否对它来说具有经济价值，最后还要确定未来的系统是否要实现此需求特性。经过识别，确定一下系统角色：会议申请者，办公室主任，会议办理者，纪要起草人，参会者。
2. 用例：在确定了系统角色以后，每一角色使用系统完成什么样的业务，就是用例，系统用例具有概括性和目标性，经过识别，确认一下系统用例：管理会议申请，获取会议纪要，管理会议纪要，分配会议室资源，发送会议信息,获取会议信息。

3. 关系：在系统用例图中，主要识别角色和系统用例间的关系以及角色与角色之间的关系，根据用例的发起者不同，把角色和用例间的关联（通信）关系分为单向管理和双向关联，单向关联有：会议申请人和编辑会议申请，会议纪要起草人和编辑会议纪要，会议办理者和发送会议通知；双向关联有：办公室主任和分配会议室资源，参会者和获取会议信息。
4. 系统：经过前面分析，未来系统将要实现的需求特征包含：编辑会议申请、编辑会议纪要、获取会议通知、分配会议室资源、发送会议通知，这些元素属于系统内，其余在系统外，属于系统环境。

5.2.3 系统用例图



5.6 会议管理系统用例图

5.2.4 操作步骤

（1） 在第四章第四节项目和包建模的基础上进行系统用例建模，如果要直接建系统用例模型，项目和包的建模操作步骤和第四章第四节相同。
（2） 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
（3） 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“会议管理”前面的“+”符号。
（4） 右击展开导航树上的“需求定义“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”用例图“。则在该包上增加了一个节点，一个名称以”usecasediagram“开头的节点，它代表这个用例图全局属性，可在属性选项卡中修改该节点名称为“会议管理系统用例图”。

图 5.7 建立用例图步骤

（5） 从绘图工具面板选中“主题“元素，将其拖入建模绘图区，并通过四周锚点适当调整绘图区的大小。在界面下面的属性选项卡中修改别名，输入“会议管理系统"。

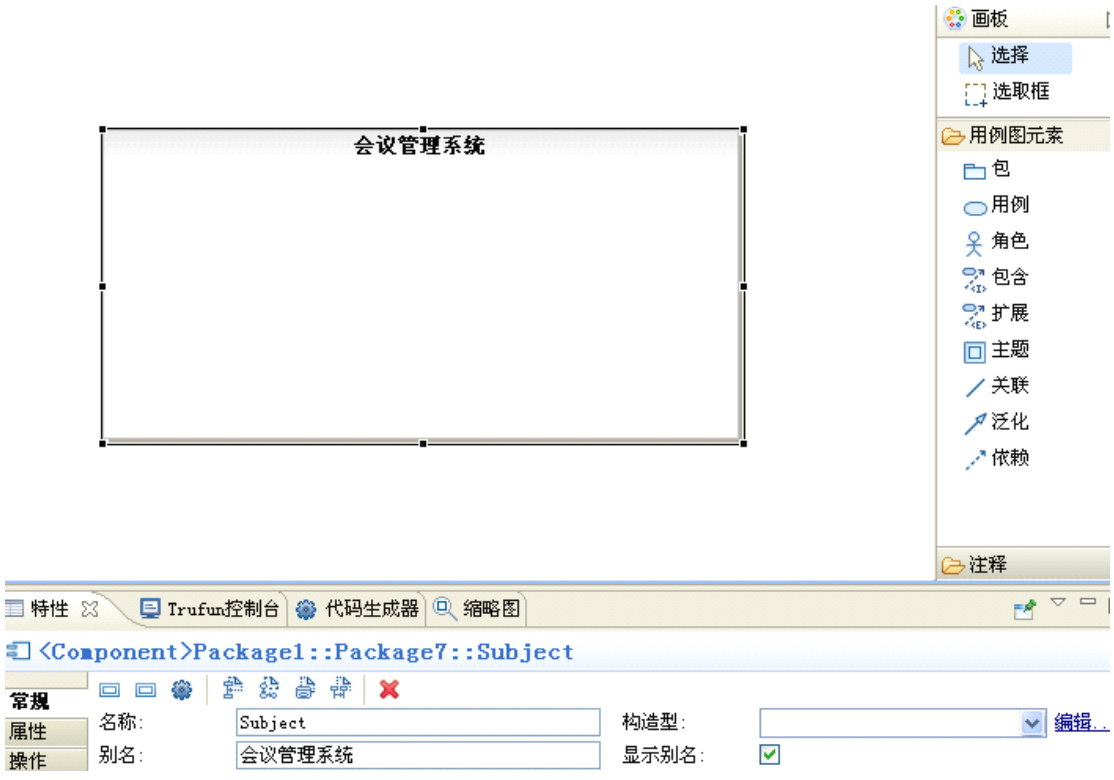


图 5.8 系统元素的添加及属性设置

（6） 再从绘图工具面板选中角色元素，把其拖入到绘图区，在适当调整大小和位置后，在界面下面的属性选项卡中修改别名，输入“会议申请人"。同理可以完成对办公室主任、会议办理人、参会人员、会议纪要起草人员布局。

图 5.9 角色的属性设置

（7） 如果角色之间存在着泛化关系，就从绘图面板选取泛化元素，连接不同的角色。

（8） 从绘图工具面板选中用例元素，将其拖入建模绘图区在适当调整大小和位置后，在界面下面的属性选项卡中修改别名，输入“管理会议申请”。同理可以完成对其他用例的布局。

图 5.10 用例元素及属性设置

（9） 从从绘图工具面板选中单向关联元素，将其拖入建模绘图区，连接系统内的“管理会议申请”和系统外的“会议申请人”。以此类推完成其他用例和角色之间的单向关联的建模。

（10） 从从绘图工具面板选中双向关联元素，将其拖入建模绘图区，连接系统内的“管理会议资源”和系统外的“办公室主任”。以此类推完成其他用例和角色之间的双向关联的建模。

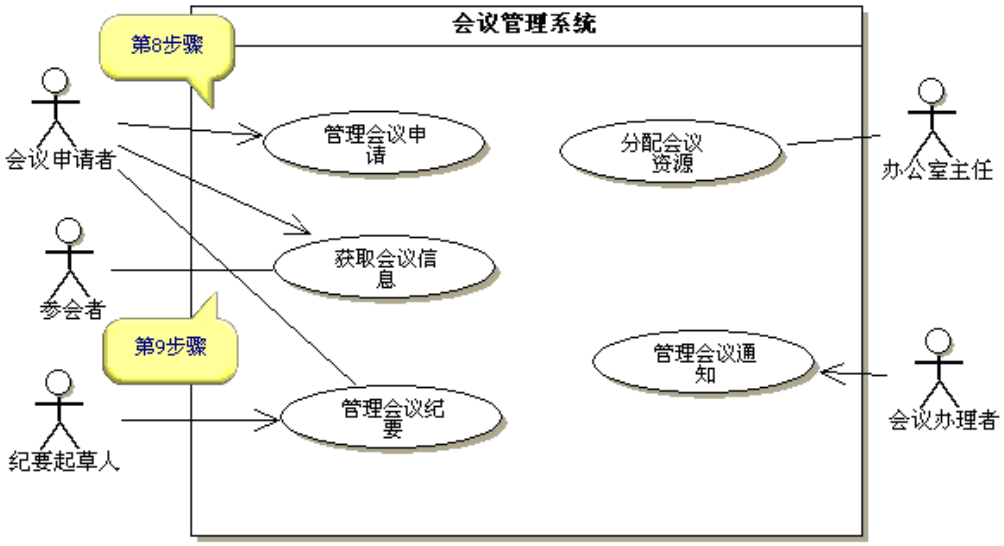


图 5.11 单向关联和双向关联的绘图

第三节 业务用例模型

5.3.1 会议管理系统业务用例分析过程

业务用例建模是在前面系统用例的基础上进行，它更关心某一具体对象的生命周期，需要对每一个顶级用例进行细化，寻找在顶级用例下包含的主要业务对象，比如办公室主任的管理会议室资源这个用例，它下面包含了会议申请、会议室、会议、会议室设备，办公室人员等多个对象，是不是把这些对象的管理统统画成低级用例呢，回答显然不是，如果这样做，就成了功能分解，就成了面向过程分析，低级用例是顶级用例的细化，它任然围绕着系统的主要流程，所以我们以会议申请对象作为重点，关心会议对象的生成和获得，因此，管理会议室资源这个用例包括浏览会议申请信息和审批会议申请两个低级用例，在审批会议申请时，需要知道可用的会议资源，会议室设备管理成为另外一个低级用例，会议室的分配、删除可以放在系统管理顶级用例（未画）中，会议的管理可以放在管理会议申请这个顶级用例中，办公人员的管理可以放在人事管理中（如果需要）。对每个低级用例都细化后，就可以得到系统的主要低级用例。

在找到低级用例后，需要寻找低级用例和顶级用例之间的关系，顶级用例和低级用例之间主要有三种关系，即包含、扩展、泛化关系，分配会议资源用例主要是针对有会议申请才进行分配，所以它和浏览会议申请信息和审批会议申请两个低级用例是包含关系，每次进行会议申请审批并不一定要进行会议室设备的管理，会议室设备的管理作为分配会议资源用例的一个补充，在需要时进行操作。所以他们之间是扩展关系。

最后是编写用例描述文档，其分析过程和系统用例的编写过程相同，不在赘述。

5.3.2 会议管理系统业务用例模型元素识别

一个解决方案的好坏不但要看它是否满足了既定的目标，还要看它是否很好地满足了项目的约束，为了避免按传统的功能分解方法分析问题，而应将每一个用例分解成更小的过程，直到完成所有内部机制的描述。业务用例建模是对前面系统用例建模的一种细化，需要找出每个顶级用例所包含的资源，需要从资源的获取、使用、及其释放等细节行为中寻找低级用例，增删改查（CRUD）是最常见的低级用例，对于不同方式的使用 CRUD 和特定的显示、计算、存储、通信都可以作为低级用例。业务用例建模主要要识别低级用例和顶级用例之间的关系。

1. 会议办理者低级用例:对于管理会议通知顶级用例，可以细分为发送会议通知和收到会议通知确认两个用例，两个用例和顶级用例之间是包含关系。
2. 会议申请者低级用例：对于管理会议申请顶级用例，可以细分为起草会议申请、修改暂存的会议申请、暂存会议申请，其中起草会议申请和顶级用例之间是包含关系，其余两个用例和顶级用例之间是扩展关系；对于管理会议纪要顶级用例，可以细分为归档会议纪要，它们之间是包含关系；对于获取会议信息顶级用例，可以细分为浏览会议信息，它们之间是包含关系。
3. 会议纪要起草者低级用例：对于管理会议纪要，可以细分为起草会议纪要、修改会议纪要、发送会议纪要，其中起草会议纪要和顶级用例之间是包含关系，其余两个低级用例和顶级用例之间是扩展关系。
4. 办公室主任低级用例：对于管理会议资源顶级用例再细分，可以分为浏览会议申请、审核会议申请、管理会议室设备，其中管理会议室设备和顶级用例之间是扩展关系，其余两个用例和顶级用例之间是包含关系。

5.3.2 会议管理业务用例图

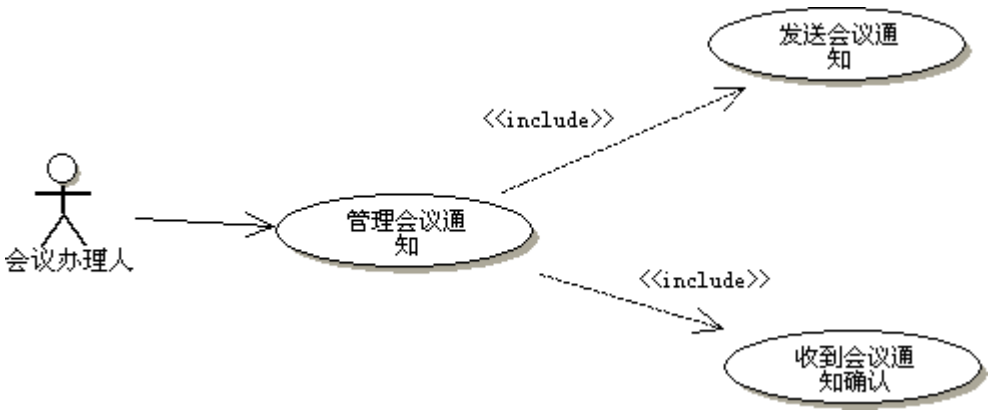


图 5.12 会议办理人业务用例模型

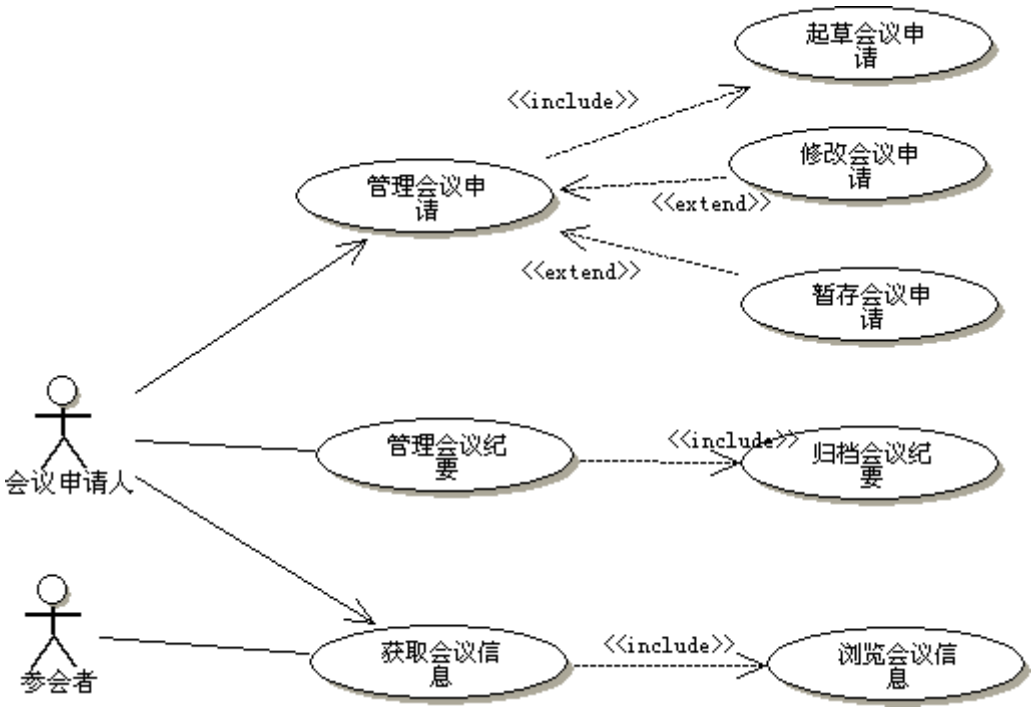


图 5.13 会议申请人业务用例模型

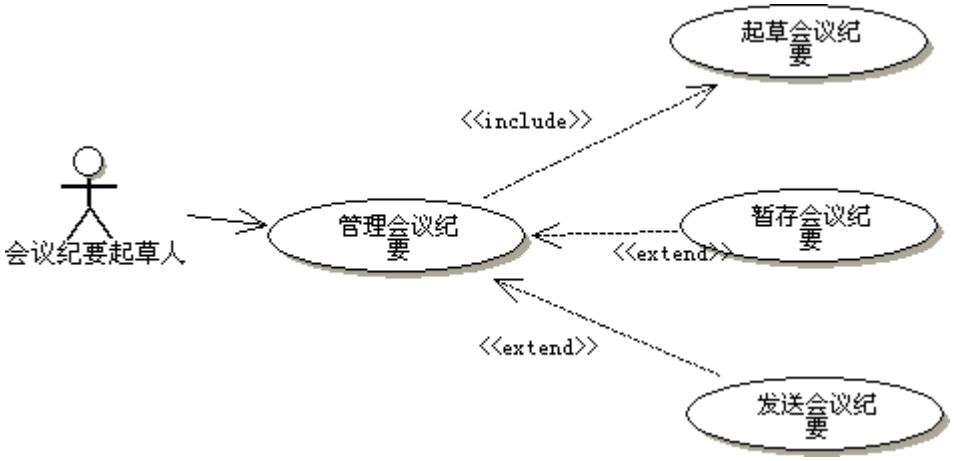


图 5.14 会议纪要起草人业务用例模型

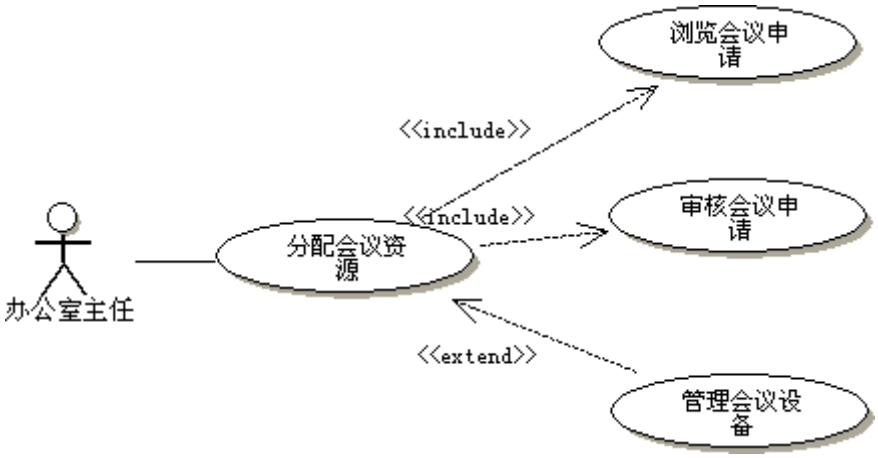


图 5.15 办公室主任业务用例模型

5.3.3 操作步骤

1. 在第四章第四节项目和包建模的基础上进行系统用例建模，如果要直接建业务用例模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“会议管理”前面的“+”符号。
4. 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”用例图“。则在该包下增加了一个节点，一个名称以”usecasediagram“开头的节点，它代表这个用例图全局属性，并可在属性选项卡中修改该节点名称为“会议申请人业务用例图”。
5. 首先从绘图工具面板选中角色元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的属性选项卡中修改别名，输入“会议申请人”。
6. 其次从绘图工具面板选中用例元素，将其拖入建模绘图区在适当调整大小和位置后，在界面下面的属性选项卡中修改别名，输入“管理会议申请”。再从绘图工具面板选中用例元素，将其拖入建模绘图区在适当调整大小和位置后，在界面下面的属性选项卡中修改别名，输入“起草会议申请”。以此类推可以完成管理会议顶级用例下的“暂存会议申请”等所有低级用例的布局。
7. 最后从绘图工具面板选中单向（或双向）关联元素，连接角色和顶级用例；再最后从绘图工具面板选中包含元素，把其拖入绘图区，连接顶级用例“管理会议申请”和低级用例“起草会议申请”，再从绘图工具面板选中扩展元素，把其拖入绘图区，连接顶级用例“管理会议申请”和低级用例“暂存会议申请”，以此类推，完成该顶级用例和低级用例之间关系的业务建模。
8. 重复 6.7 完成该角色的其他顶级用例的业务建模。
9. 重复 5、6、7、8 完成其他角色的业务建模。

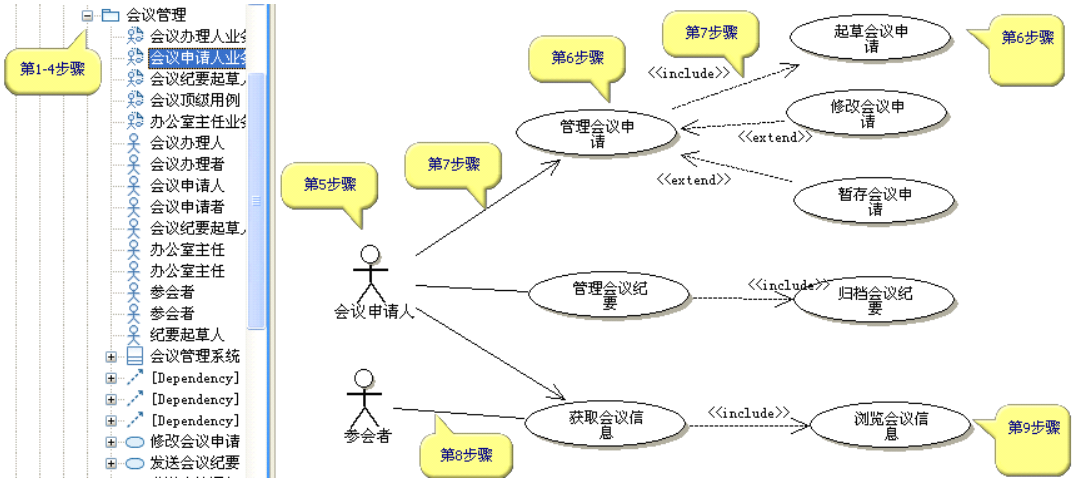


图 5.16 业务用例建模操作步骤

5.3.4 车辆管理业务用例图

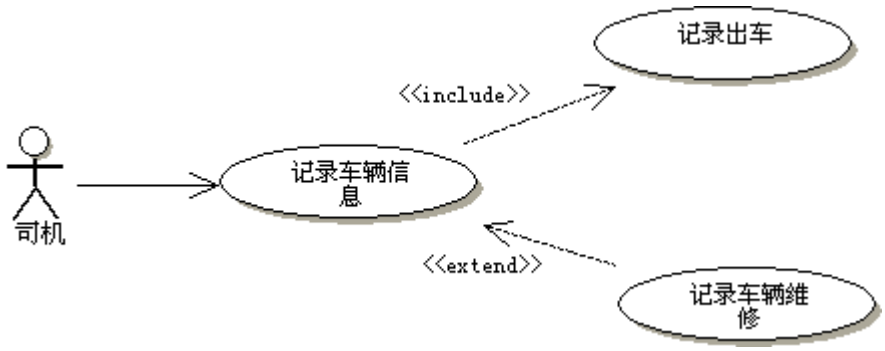


图 5.17 司机业务用例模型

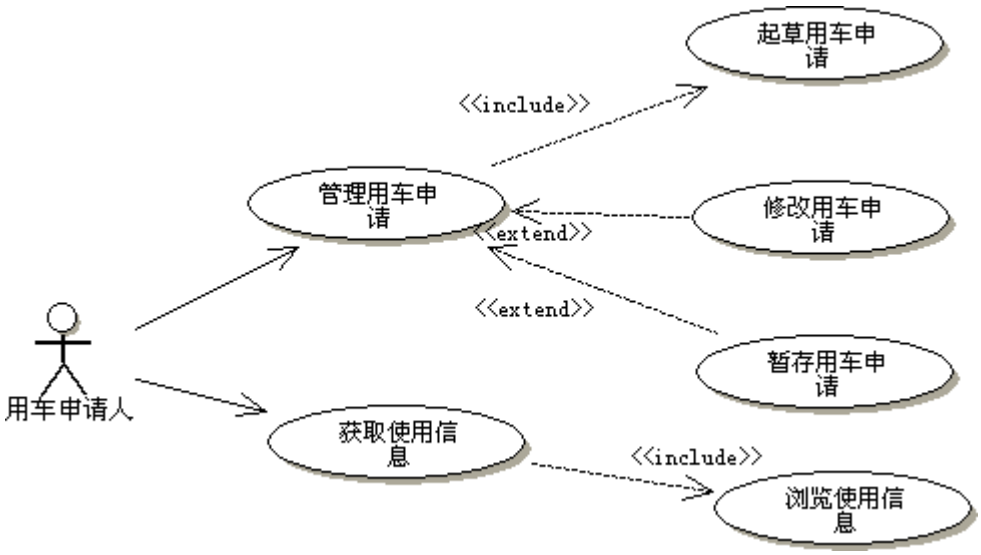


图 5.18 用车申请人业务用例图

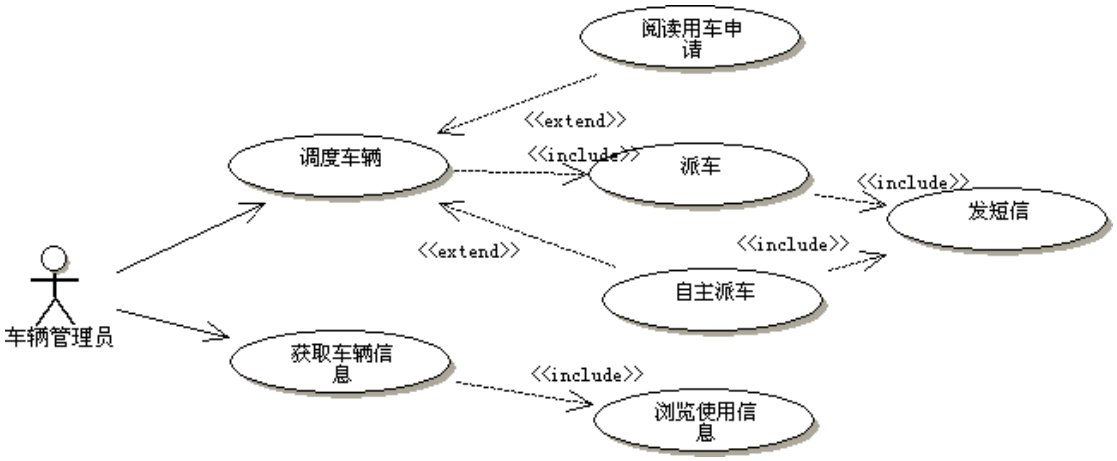


图 5.19 车辆管理员业务用例图

第四节 用例描述文档规范

5.4.1 用例描述模板

用例模型充分反映了角色和系统之间的关系，作为角色和系统之间交互的模型，显然缺乏行为细节描述，因此，需要一份书面的说明文档，象写契约一样，把双方的都要遵守的规则都写进去，用来描述用例的模板很多，没有统一规范，下面就一些常用描述元素做以概括介绍：

（1） 前置条件：用例执行前必须满足的条件，如果不满足条件，用例不被执行，一般与系统的上下文环境和参数限定有关。

（2） 后置条件：在用例结束时，系统必须所处的状态，以防止用例执行的结果不确定，影响后续用例的执行。

（3） 基本事件流：路径是从用户角度来描述系统的使用，它关注系统干什么，而不关心怎样干。基本事件流是用例中最常发生的路径。

（4） 可选事件流：是系统合法的路径，但不是经常发生的路径，

（5） 异常事件流：指未来系统要捕获错误的路径。

（6） 参与者:触发用例的角色、系统、时间。

（7） 用例名称：就是用例图中用例名。

5.4.2 会议管理系统用例描述

（1）用例名称：起草会议申请。

参与者：会议申请人。

前置条件：会议申请人有条件通过网络访问系统并已成功地登录系统。

后置条件：系统保存一份新的会议申请。

基本事件流：1.用户通过网络登录后成功访问系统。
2.用户选择会议管理后，再选择浏览会议信息。
3.浏览结束后用户选择查看暂存会议申请。
4.在确认无合适的会议申请后，用户选择起草会议申请。
5.用户输入会议申请的相关信息。
6.会议申请经过校验后提交办公室主任。

可选事件流：1.用户发现有可用的暂存申请可以修改，系统进入修改会议用例界面。
2.新起草的会议申请被暂存。

异常事件流：1.会议室已被预定，给出错误信息提示。
2.会议信息校验失败，给出错误信息提示。

（2）用例名称：修改会议申请。

参与者：会议申请人。

前置条件：会议申请人有条件通过网络访问系统并已成功地登录系统。

后置条件：系统保存一份修改过的会议申请。

基本事件流：1.用户通过网络登录后成功访问系统。
2.用户选择会议管理后，再选择浏览会议信息。

3.浏览结束后用户选择查看有无暂存的会议申请。或浏览发现有办公室主任退回需要修改的会议申请。
4.在确认有合适的会议申请后，用户选择修改会议申请。
5.用户输入会议申请的相关信息。
6.会议申请经过校验后提交办公室主任。

可选事件流; 修改后的会议申请被暂存。

异常事件流：1.会议室已被预定，给出错误信息提示。
2.会议信息校验失败，给出错误信息提示。

（3）用例名称：暂存会议申请。

参与者：会议申请人。

前置条件：会议申请人有条件通过网络访问系统并已成功地登录系统。

后置条件：系统保存一份会议申请。

基本事件流：1.用户通过网络登录后成功访问系统。
2.用户选择会议管理后，再选择浏览会议信息。
3.用户在选择起草或修改会议申请后，选择暂存。
6.会议申请经过校验后保存。

可选事件流; 用户选择放弃本次操作。

异常事件流：1.是否覆盖以前的相同会议申请，给出错误信息提示。
2.会议信息校验失败，给出错误信息提示。

（4）用例名称：浏览会议申请。

参与者：会议申请人、办公室主任。

前置条件：用户通过网络访问系统

后置条件：正确显示用户所要浏览的信息。

基本事件流：1.用户通过网络成功登录系统
2.用户输入自己要浏览的信息关键词。
3.选择查询。
4.系统显示查询结果。

可选事件流：1.未查到相关信息，给出信息提示。
2.信息量大时分页显示。

异常事件流：1.查询条件有误，给出错误信息提示。
2.查询超时，给出错误信息提示。

（5）用例名称：归档会议纪要。

参与者：会议申请人。

前置条件：用户通过网络访问系统。

后置条件：成功归档会议纪要。

基本事件流：1.用户通过网络成功登录系统。
2.用户选择要浏览会议信息。
3.在确认有需要归档的会议纪要后选择归档。
4.系统显示归档成功。

可选事件流：1.选择打印，用纸质归档。
2.选择暂存。

异常事件流：1.权限不够，给出错误信息提示。
2.附件太大，给出错误信息提示。

（6）用例名称：审核会议申请。

参与者：办公室主任。
前置条件：用户通过网络访问系统。
后置条件：会议申请发给会议办理人或退回给会议起草人。
基本事件流：1.用户通过网络成功登录系统。
2.用户选择要浏览会议信息。
3.在确认有需要审核的会议申请后选择审核。
4.用户输入审核意见。
5.用户选择把会议申请发给会议办理人。
可选事件流：1.选择退回给会议申请人。
2.选择放弃本次操作。
异常事件流：1.权限不够，给出错误信息提示。
2.审核意见为空，给出错误信息提示。

（7）用例名称：发送会议通知。
参与者：会议办理人。
前置条件：用户通过网络访问系统。
后置条件：会议通知发送到所有参会人员。
基本事件流：1.用户通过网络成功登录系统。
2.用户选择要浏览会议信息。
3.在确认有需要办理的会议通知后选择发送会议通知。
4.用户输入会议通知的内容。
5.用户选择把会议通知发给参会者。
可选事件流：无
异常事件流：1.发送超时，给出错误信息提示。
2.会议通知内容为空，给出错误信息提示。

（8）用例名称：确认会议通知
参与者：会议办理人，参会者
前置条件：用户通过网络访问系统，
后置条件：确认会议通知发送到参会人，所有参会人都收到会议通知。
基本事件流：1.用户通过网络成功登录系统。
2.用户选择要浏览会议信息。
3.在确认有需要确认的会议通知后选择确认会议通知。
4.会议办理人在收到所有参会者确认后，确认会议通知发送完毕。
5.用户选择确认会议通知发送完成。
可选事件流：1.有些参会者未确认，选择对部分人重发会议通知。
2.采用系统外的其他方式确认。
异常事件流：1.确认人和参会人不符，给出错误信息提示。
2.超过一定次数仍未确认，给出错误信息提示。

（9）用例名称：起草会议纪要。
参与者：会议纪要起草人。
前置条件：会议纪要起草人有条件通过网络访问系统并已成功地登录系统。
后置条件：系统保存一份新的会议纪要。
基本事件流：1.用户通过网络登录后成功访问系统。
2.用户选择会议管理后，再选择浏览会议信息。
3.浏览结束后用户选择查看暂存会议纪要。

4.在确认无合适的会议纪要后，用户选择起草会议纪要。
5.用户输入会议纪要的相关信息。
6.会议纪要经过校验后提交给会议申请人。

可选事件流：1.用户发现有可用的暂存纪要可以修改，系统进入修改会议纪要用例。
2.新起草的会议纪要被暂存。

异常事件流：1.会议纪要发送失败，给出错误信息提示。
2.会议纪要信息校验失败，给出错误信息提示。

（10）用例名称：修改会议纪要。

参与者：会议纪要起草人。

前置条件：会议纪要起草人有条件通过网络访问系统并已成功地登录系统。

后置条件：系统保存一份修改过的会议纪要。

基本事件流：1.用户通过网络登录后成功访问系统。
2.用户选择会议管理后，再选择浏览会议信息。
3.浏览结束后用户选择查看有无暂存的会议纪要。
4.在确认有合适的会议纪要后，用户选择修改会议纪要。
5.用户输入会议纪要的相关信息。
6.会议纪要经过校验后提交会议申请人。

可选事件流; 修改后的会议纪要被暂存。

异常事件流：1.会议发送失败，给出错误信息提示。
2.会议纪要信息校验失败，给出错误信息提示。

（11）用例名称：暂存会议纪要。

参与者：会议纪要起草人。

前置条件：会议纪要起草人有条件通过网络访问系统并已成功地登录系统。

后置条件：系统保存一份会议纪要。

基本事件流：1.用户通过网络登录后成功访问系统。
2.用户选择会议管理后，再选择浏览会议信息。
3.用户在起草或修改会议纪要后，选择暂存。
6.会议纪要经过校验后保存。

可选事件流; 用户选择放弃本次操作。

异常事件流：1.是否覆盖以前的相同会议纪要，给出错误信息提示。
2.会议纪要信息校验失败，给出错误信息提示。

第五节 小结

用例建模是整个建模的基础，它上联需求的原始文字资料，下联动态分析建模。用例建模的好坏直接关系到项目的成败。用例是需求中功能的模型化描述，本章从用例模型的原理出发，首先介绍了寻找用例的方法，然后对会议管理寻找模型元素，并建立了系统用例模型和业务用例模型，最后介绍了用例描述模板，并对会议管理编写了用例描述。

第六节 习题

1. 简述用例与原始需求的关系。
2. 系统用例和业务用例有什么区别？
3. 为什么要写用例描述文档，用例描述文档主要包括哪些内容？
4. 对车辆管理系统找出不同的系统边界。
5. 对车辆管理系统编写用例描述文档。
6. 画出固定资产管理系统系统用例图和业务用例图。

第五章 创建类图

软件开发已经有几十年的历史，在这个阶段产生了很多开发语言，但其支持的理论基础却比较少，主要有面向过程开发和面向对象开发，面向过程开发主要从功能出发来建立系统及系统与系统之间的关系。由于功能具有不稳定性，软件开发逐渐转向了面向对象开发，类是面向对象开发的核心。本章将在业务领域，分析支撑业务实现的静态类。它与后几章讲到的支持用例实现的动态类有一些本质的区别。如何用面向对象的思想去观察世界，去分析问题和解决问题，是本章的重点。

第一节 定义类

6.1.1 类的基本概念

面向对象的思想来自人们对客观世界的认识，希望把对客观世界的认识，映射到计算机世界。对象是真实世界中某一具体事务，可以是有形的，也可以是无形的。如果把真实世界的每一个对象都搬进计算机，其开发和管理的复杂程度及效率可想而知，因此，面向对象思想提出了一种类的概念。即把满足一组规则的所有对象的集合称为类。这组规则就是类属性和操作。类是一种模板，它的实例化就是对象。

既然面向对象的核心是类，而客观世界的真实对象映射到软件领域任然是一个对象，怎样把对象转化为类呢？回想一下我们在第一、二章讲过的面向对象分析方法，其中有一种方法就是抽象，也就是提取维护和操作对象必要的信息，而丢弃大量不必要信息。由于业务不同，同一个对象，可以从不同方面进行抽象。

由于描述类的规则分为属性和操作，如果把这些属性和方法分散到软件的各个地方，则抽象的类就名存实亡。在面向对象分析方法中有一种叫封装的方法，它可以把类的属性和方法组织起来，也就是所谓的’信息隐藏“，一方面，实现这些方法，对象可以直接使用这些属性，另一方面，这些属性是受到保护的，没有权限是不可访问这些信息的。同时，只有通过这些操作，才能访问被封装的属性。

如果一个类中有一个可执行线程，就是主动类，如果该类最后要保存到数据库，就是永久类。

6.1.2 类的识别方法

对软件开发进行分析建模，依据的是原始需求资料，而原始需求是由文字描述而成，说的更白一点，就是由名词、动词、形容词等等按照一定的规则组合而成，名词一般被识别成类或属性，动词一般被识别成操作，形容词一般被识别成性能属性。

一个名词被识别成属性还是类，与该软件业务有很大的关系，通常如果一个名词有另外的名词作为附属，或有一个动词受此名词的支配，那么该名词就是类，其次要寻找隐含

在字里行间的名词，合并含义相同的名词。

一般来讲，参与业务活动的人、组织机构、系统管理的设备都是类，需要长期保存的事件也是类，业务运转的表单、票据都是类，另外还有一些为了业务运转而附加的类。

这些寻找出来的待定类，经过反复整理、筛选，最后形成候选类。

6.1.3 类图的模型元素

（1） 类： 类是一种系统资源，除了属性和操作外，还有一个唯一的名称，名称一般用名词或名词短语。在 UML 中，类用一个矩形来表示，分为长格式和短格式，长格式在矩形中有分为三栏，从上到下分别是名称、属性、操作；短格式在矩形中只有类名称。

图 6.1 类元素的短格式和长格式表示法

（2） 可见性：类的可见性是指类的某个成员能够被访问的范围，有的资料中称为类的范围，类和包都包含了元素的可见性，可以限制对这些元素的访问。private 表示在一个类内可以访问，用“- “表示；package 表示在相同的包内可以访问，用”~“表示，public 表示在这个系统中可以访问，用”+”表示，protected 表示在一个继承树之内可以访问，用“#”表示。楚凡科技在自己的建模工具中定义了一套新的可见性表示法，如果有兴趣，可查看有关手册。

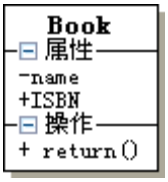


图 6.2 可见性的表示示例

（3） 代码名称：为了保证类模型图生成代码后具有可利用价值，类、方法、属性、参数除了名称外，还有代码，该代码受两方面因素影响，其一是受各公司软件开发规范中命名规范影响，其二是受所选择的开发语言对命名的要求影响，如果在分析阶段描述代码，则陷入了分析结果与实现有关的困境。一般在分析阶段不关心代码名称，而留到设计阶段完善，由于类图最终要生成代码，可在分析阶段之用别名，在设计阶段根据所选择的开发工具和开发规范，对类名称、属性、操作，再添加规范的代码名称。

（4） 外部数据类型：是指在分析阶段给用户所看的数据类型，其数据类型一部分是 UML 定义的，如整数：integer，字符串：string，静态属性：isStatic（或用下划线），另外一些是借用 java 的数据类型，如：float、long、short 等，外部数据类型只用来表示输入输出中的数据类型，而不是软件运行时在系统内的存储数据类型，同样，需要在设计阶段根据语言的不同，将外部数据类型转化为内部数据类型，类一般用“：”

隔开代码名称和外部数据类型。

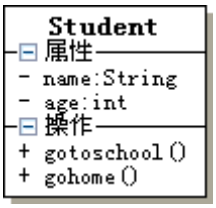


图 6.3 类图的代码名称和数据类型表示示例

（5） 多重性：多重性是指类中的某些属性，特别是某一关联方充当属性时，对于某个具体对象，往往对应着多个该属性取值，该属性代码通常用数组“[]”或集合”set、list“来表示。对于连续范围，用上限和下限之间加两点表示，如[1..5],如果无法确定某一边界，可用”*“表示，如果上下边界相同，可用一个数字表示。

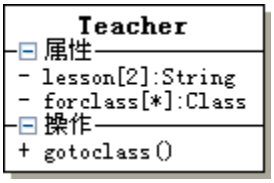


图 6.4 多重性的表示示例

（6） 默认值：为了保证系统的完整性和用户的可理解性，属性往往具有默认值，默认值一般用“=“表示，后面可以是任何形式的文本或数字，在设计阶段，必须转换为该语言所能支持的表达式。

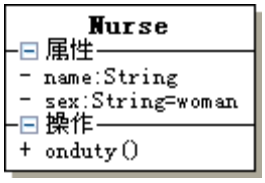


图 6.5 默认值的表示示例

（7） 特定字符串：为了使属性满足某些限制规则，在属性定义后面加一个大括号，大括号里的内容是限制字符串，如:{ readonly}

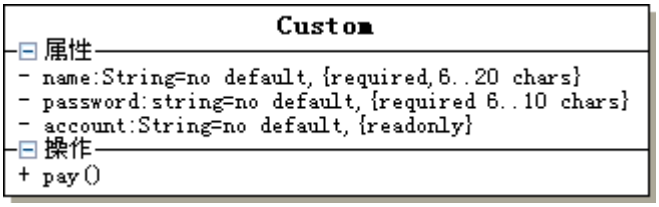


图 6.6 特定字符串的表示示例

（8） 对象图：对象图是类图的一种实例，表示系统运行时对象之间的关系，除非有特殊的需要，否则不进行建模。

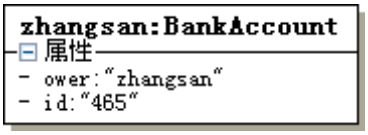


图 6.7 对象图的表示示例

（9） 参数列表：参数列表是操作中输入的按顺序排列的属性，可以有多个，参数列表是可选的。一般放在小括号里，参数带数据类型时，用“:”和参数隔开。在设计阶段，要根据语言进行修改参数类型。

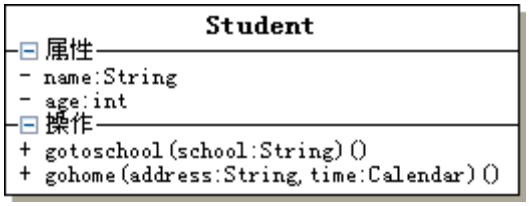


图 6.8 参数列表表示示例

（10） 返回值：返回值指操作的输出，返回值只规定了返回值的类型，没有名称，且只有一个返回值（可以是对象）。返回值用“：”和操作隔开，在设计阶段，要根据语言进行修改返回值的数据类型。

图 6.9 返回值的表示示例

6.1.4 类图的建模分析步骤

（1） 寻找出需求中的名词(候选对象)。
（2） 合并含义相同的名词，排除范围以外的名词，并寻找隐含的名词。
（3） 去掉只能作为类属性的名词。
（4） 剩下的名词就是要找的分析类（候选类)。
（5） 根据常识、问题域、系统责任确定该类有那些属性。
（6） 补充该类动态属性，如状态、对象间联系（如聚合、关联）等属性
（7） 从需求中的动词、功能或系统责任中寻找类的操作（候选操作）。
（8） 从状态转换，流程跟踪、系统管理等方面补充类的操作。
（9） 对所寻找的操作进行合并、筛选。
（10） 对所寻找的操作在类间进行合理分配（职责分配）。形成每个类候选操作。
（11） 补充每个类的的分析文档，为类的进一步设计打下基础。

6.1.5 会议管理中类图中类的分析过程

从前面的分析我们知道，每一系统都有一个或几个主要分析类，并且有一些其他类围绕在这个类的周围，辅助主要分析类完成它的生命周期，对于会议管理子系统来讲，显然会议是主要的分析类，主要分析类是在需求采集阶段就要确定的类，在画类图时只画主要分析类呢？当然不是，一个孤零零的类什么作用都没有，只有类和类之间发生关系才能形成业务，所以第一步是要找到其他的支持类，从哪儿找呢？回答是原始需求文档。

首先我们从原始需求文档中找到名词和隐含的名词，在软件开发以后，这些名词可以演化为类或类的属性，对于会议管理系统来讲，前半部分文档所包含的名词有：会议 、行政管理、手段、会议类别、会议室、会议申请、会议通知、会议纪要、基础、信息、会议性质名称、备注、会议室名称、容纳人数、会议室资源、说明、使用情况、会议申请人、会议安排、会议性质、会议议题、预算、会议附件、主持人、记录人员、参加人

员、会议地点、会议室、会议开始时间、会议结束时间、会议内容、审批人。

其次要合并含义相同的名词，如把备注和说明合并为说明、把会议性质和会议性质名称合并为会议性质。还要排除范围以外的名词，行政管理、手段、基础、信息都是一些概括性的名词，只能作为抽象类或子系统、系统名称，可以暂时不考虑，最后还需要找到一些隐含的名词，由于在会议管理中，作某些事情（如提交会议申请）的人，往往不是一个特定的员工，而是有一群这样的人，而且这群人中的具体员工在随时发生变化，在这里事实上隐含了一个岗位或职责的概念，我们用角色来代替。员工这个隐含的名词也必须出现在系统，才能使整个系统具有完整性。

最后去掉只能作为类属性的名词。一个名词是作为类还是作为类的属性，关键是看在需求中有没有动词作用在该名词上，对会议管理子系统来讲，会议性质名称、备注、会议室名称、容纳人数、会议室资源、说明、使用情况、会议申请人、会议安排、会议性质、会议议题、预算、会议附件、主持人、记录人员、参加人员、会议地点、会议室、会议开始时间、会议结束时间、会议内容、审批人，都没有动词作用于该名词，只能作为属性使用，具体是那个类的属性，我们在后面分析。

剩下的名词有：会议、会议类别、会议室、会议申请、会议通知、会议纪要、容会议附件。再加上隐含的名词：角色、员工，它们构成了类图的候选类。

6.1.6 会议管理识别的类

会议管理静态分析识别的类有：会议类别，会议室，会议，会议申请，会议通知，会议纪要。另外还有人员，附件，角色几个附加类。下面是这些类的详细说明：

序号	类别名	类名称	主动性	永久性	解释
1	会议类别	meetingType	无	有	指企业会议的性质分类
2	会议	meeting	无	有	记录会议信息的实体
3	会议室	meetingRoom	无	有	能够举行会议的房间
4	会议申请	mettingApplication	有	有	预开会申请实体
5	会议通知	meetingNotice	有	有	记录会议通知的实体
6	会议纪要	meetingArchive	有	有	记录会议信息的实体
7	人员	person	无	有	参与会议活动的人
8	附件	attachment	无	有	会议辅助信息
9	角色	role	无	有	按权限完成一定操作的人

第二节 定义类的属性

6.2.1 属性的基本概念

类的属性是类的一个描述特征，如果一个类没有这个属性，就不能保持语义的完整性，这也是属性取舍的依据，系统分析时，人们往往只关心与特定系统有关的属性。

和类的来源一样，属性也是来源于原始需求中的名词。其次属性具有另外一个重要特征就是原子性，即属性不能再分解。同时也不能从其他属性推出这个属性。寻找属性时，一般要从类的实例-对象来着手，并从以下几个方面进行考虑：

1、按一般常识这个对象有那些属性。
2、在当前问题域该对象有那些属性。
3、根据系统责任，这个对象应该有那些属性。
4、为了实现某些功能，对象需要增加那些属性。
5、对象有那些区别的状态。
6、对象整体和部分属性设置

6.2.2 会议管理类图中属性的分析过程

在静态分析前期阶段找到的类，一般都是可以持久化的实体类，因此都有属性，下面我们给前面找到的候选类寻找属性，首先我们对前面赛选下来只能作为属性的名词进行归类，它的归类是按照一般的常识进行，会议性质名称这个名词放在会议对象和放在会议类别对象里好像都可以，其实不然，会议和会议类别这两个对象作用的动词都是设置，但从生命周期上来讲，应该是先有会议类别，然后才有会议，同时如果会议分类这个对象没有这个属性就失去存在的价值，因此放在会议类别里面更合适；会议室名称、容纳人数、会议室资源、使用情况都和会议室对象有关，按照常识可以作为它的属性。会议申请人、会议安排、会议议题、预算、会议附件、主持人、记录人员、参加人员、会议地点、会议室、会议开始时间、会议结束时间、会议内容，都与会议对象有关，按照常识可以作为它 的属性，审批人与会议申请对象有关，可以作为它的属性。说明做为对象信息的补充，可以根据需要添加到任何对象里。

根据业务需要，我们还必须对对象添加需求文档中没有，但完成此业务必须有的属性，对会议申请对象需要添加申请时间、审批时间、审批意见、办理人、办理时间；会议通知对象需要添加发送人、发送时间、接收人、接收时间；会议纪要对象需要添加纪要内容和纪要附件。

用计算机进行管理，关键是能提高工作效率，对对象进行编号，是提高效率的常见做法，为此需要添加会议分类编号、会议编号、会议室编号、会议申请编号、会议通知编号、通知编号、人员编号、附件编号、角色编号。当然，如果需要也可以给这些对象增加对象产生、对象修改、对象删除、对象归档等属性以及操作人的属性。以便进行责任追查和工作量统计，会议管理子系统没有类似的要求，所以不需要添加相关属性。

对于某些功能的实现，还必须添加相应的属性，才能满足为该对象方法提供数据的要求，在附件对象中，有一个上传功能，而这一功能的实现，需要知道上传附件的数量、附件的大小、类型、格式，所以必须对附件这个对象添加这些属性。

我们知道，很多对象只有在一定状态时，才能完成相应的操作，同时状态是提高系统效率的必要信息。因此，需要对会议室、会议、会议申请添加状态属性。

对于对象之间的一些关系属性的分析，我们将在下一章进行。

6.2.2 会议管理系统属性识别

1. 类名： 会议分类

属性别名	属性名	可见性	数据类型	静态	默认值	解释
会议性质	nature	private	String	false	no default	会议管理分类依据
会议分类编号	id	public	String	false	no default	唯一标识
说明	explain	private	String	false	empty	补充信息

2. 类名： 会议室

属性别名	属性名	可见性	数据类型	静态	默认值	解释
会议室名称	roomName	public	String	false	no default	会议室房间代号
会议室资源	resource	private	String	false	no default	指桌椅、投影仪等
容纳人数	sum	private	int	false	10	设计人数
状态	status	public	String	false	idel	使用状态
说明	explain	private	String	false	empty	补充信息
会议室编号	id	public	String	false	no default	唯一标识

3. 类名： 会议

属性别名	属性名	可见性	数据类型	静态	默认值	解释
会议议题	title	private	String	false	no default	会议讨论的问题
预算	budget	private	String	false	0	会议费用计划
附件名称	attachName	private	String	false	no default	附件标题
主持人	host	private	String	false	no default	控制会议进度的人
记录员	writer	private	String	false	no default	记录会议讨论信息的人
地点	place	private	String	false	no default	开会的地点
参会人员	attendPerson	private	String	false	no default	到指定会议室开会的人
会议室	meetingRoom	private	String	false	no default	举行会议的地方
开始时间	beginTime	private	Datetime	false	no default	会议计划开始时间
结束时间	endTime	private	Datetime	false	no default	会议计划结束时间
内容	context	private	String	false	Empty	详细信息
状态	status	public	String	false	no default	目前所处的状态
会议编	id	public	String	false	no default	唯一标识

号						
会 议 类 别	type	public	String	false	no default	会议性质

4. 类名： 会议申请

属性别名	属性名	可见性	数据类型	静态	默认值	解释
申请人	applicant	public	String	false	no default	发起会议申请的人
申请时间	offerTime	private	String	false	no default	提交申请的时间
会议	meeting	private	String	false	no default	会议类属性
申请状态	status	public	String	false	no default	申请所处的阶段
审批人	vetting	public	String	false	no default	批准是否举行会议的人
审批时间	vettingTime	private	Datetime	false	no default	签署审批意见的时间
审批意见	oppinion	private	String	false	Empty	签署的意见内容
办理人	handle	private	String	false	no default	办理开会相关事宜的人
办理时间	handleTime	private	Datetime	false	no default	办理完成时间
申请编号	id	public	String	false	no default	管理编号

5. 类名： 会议通知

属性别名	属性名	可见性	数据类型	静态	默认值	解释
会议	metting	public	String	false	no default	会议类属性
发送人	sender	private	String	false	no default	发送会议通知的人
发送时间	sendTime	private	Datetime	false	no default	通知发送完成时间
通知编号	id	public	String	false	no default	管理编号
接收人	receiver	public	String	false	no default	参加会议的人
接收时间	recTime	private	Datetime	false	no default	收到会议通知的时间

6. 类名： 会议纪要

属性别名	属性名	可见性	数据类型	静态	默认值	解释
会议	metting	private	String	false	no default	会议类属性
纪要编号	id	public	String	false	no default	唯一标识
纪要内容	context	private	String	false	empty	详细信息
附件	attachment	public	String	false	Empty	其他信息
记录员	writer	private	String	false	no default	记录会议内容的人
管理员	manager	private	String	false	no default	接收归档的人

7. 类名： 附件

属性别名	属性名	可见性	数据类型	静态	默认值	解释
附件名称	title	private	String	false	no default	辅助资料名称
附件数量	sum	private	String	false	1	辅助资料的数量
附件类型	type	private	String	false	no default	是何种信息的资料
附件格式	format	private	String	false	no default	文档格式
附件大小	size	private	float	false	no default	文档大小
上传时间	transfer	private	Datetime	false	no default	上传到服务器时间
附件编号	id	public	String	false	no default	唯一标识
所属文档	whosid	public	String	false	no default	所属文档编号

编号						

8. 类名： 人员

属性别名	属性名	可见性	数据类型	静态	默认值	解释
姓名	name	private	String	false	no default	在册姓名
部门	depart	private	String	false	no default	当前所在部门
岗位	position	private	String	false	no default	当前所在岗位
角色	role	public	String	false	no default	在会议管理系统的角色
人员编号	id	public	String	false	no default	在册编号

9. 类名： 角色

属性别名	属性名	可见性	数据类型	静态	默认值	解释
角色编号	id	public	String	false	no default	管理编号
角色	role	public	String	false	no default	完成某类工作的一类人
说明	explain	private	String	false	empty	补充信息

第三节 定义类的操作

6.3.1 操作的基本概念

类的操作位于类内部，用来操作属性或进行其他动作，只是适合这个类的所有对象。操作签名是指操作的返回类型、名称、参数。它是操作完成所必要的全部信息，类的操作是通过类的对象调用来实现的。

接口是一种声明而不是类，它只含有操作特征，接口定义的操作不能有任何实现。接口没有属性，接口的实现依赖于和接口相连的类，一个接口可以有多个实现。在 UML 中，接口用原型<<interface>>来表示，或用一条直线和一个空心圆来表示提供接口；用一条直线和半个空心圆来表示需求接口。

寻找操作要从类的实例-对象来着手，并从以下几个方面进行考虑：

1、从需求中的功能，寻找对象的操作。
2、系统行为推迟到设计阶段。
3、暂不考虑对象中读、写属性这样的操作。
4、根据系统责任，这个对象应该有那些操作。
5、分析对象的状态转换，寻找操作。
6、追踪流程，寻找操作。
7、类本身要使用那些操作来维护信息更新以及信息的一致性和完整性。

6.3.2 会议管理类图中操作的分析过程

对类的操作的识别，也是在类识别分析结束以后进行，前面我们已经找到了候选类，那是通过原始需求文档中的名称发现的，对原始需求中的动词，我们还没有作处理，下面让我们看看会议管理前半部分中主要动词：（会议类别）设置、（会议室）设置、（会议）申请、（会议）审核、（会议）通知、（会议）查询、（会议）归档、保存、修改、删除、查看

（使用情况）、草拟（会议安排），输入（会议信息）、（附件）上传、（会议申请）暂存、发给（审批人），这些动词中，有些在括号里附加了它所作用的名词，主要是为了阅读方便，而有些动词没有附加作用的名词，这是因为它可以作用于多个名词。很显然，我们能够很容易地把这些动词并成操作并分配类中，如动词“设置”分配到会议类别对象变为增加会议类别操作，动词“申请”分配到会议申请对象变为起草会议申请操作。如此反复，首先完成原始需求中的动词分配。也就是从需求中功能找出的操作。

对于对象的普通属性的获取和设置，在静态分析阶段不进行添加，留到设计阶段添加。对于系统行为如备份、恢复、日志等等，也暂不作设置，留到设计阶段添加。

信息系统对对象的管理往往是对对象整个生命周期的管理，除了增加操作外，根据系统责任，往往还必须有删除，否则系统就会垃圾成山，所有对象都有删除操作（与增加相对），修改和查询操作根据对象的作用进行合理设置，一般情况下都是要设置的。

状态属性在对象中的作用我们在前面已经讨论过，对象属性的改变往往是有条件的，受触发事件的影响，因此在有状态属性的对象，需要增加状态改变操作。

从业务流程中找操作，如会议申请对象在申请过程中随时都有可能放弃，所以给会议申请对象增加一个放弃操作。

为了保证系统的完整性，很多对象在收到输入信息以后，首先要进行业务规则检验，如新增人员信息输入到对象后，系统要作重复性检验，附件信息输入后，系统要作与上传信息有关的限制条件检验。会议申请信息输入后，要进行会议室、会议时间有效性业务规则检验。同样会议纪要也要进行类似的有效性检验。所以对人员对象、附件对象、会议申请对象、纪要对象需要增加业务有效性检验，当然，从界面来的所有信息，都要在界面类中作输入有效性检验，对它的分析将在后面动态分析中进行。

6.3.3 会议管理系统操作识别

1. 类名：会议类型

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
删除	delete()	public	boolean	会议类型编号	false	false	删除无用的会议类型
增加	add()	public	boolean	会议类型编号	false	false	增加新的会议类型
修改	update()	public	boolean	会议类型编号	false	false	修改原来的会议类型
查询	query()	public	String[]	会议类型编号	false	false	通过关键词查询类型

2. 类名：会议室

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
删除	delete()	public	boolean	会议室编号	false	false	删除无用的会议室
增加	add()	public	boolean	会议室编号	false	false	增加新的会议室
修改	update()	public	boolean	会议室编号	false	false	修改原来的会议室
查询	query()	public	String[]	会议室编号	false	false	通过关键词查询
改变状态	Change()	Public	Boolean	会议室编号	False	false	改变使用状态

3. 类名：会议

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
删除	delete()	public	boolean	会议编号	false	false	删除无用的会议
增加	add()	public	boolean	会议编号	false	false	增加新的会议
修改	update()	public	boolean	会议编号	false	false	修改原来的会议

查询	query()	public	String[]	会议编号	false	false	通过关键词查询
改变状态	Change()	Public	Boolean	会议编号	False	false	改变会议状态

4. 类名：会议申请

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
暂存	delete()	public	boolean	会议申请编号	false	false	保存会议申请
起草	add()	public	boolean	会议申请编号	false	false	编写会议申请
修改	update()	public	boolean	会议申请编号	false	false	修改会议申请
查询	query()	public	String[]	会议申请编号	false	false	通过关键词查询
审核	sign（）	public	boolean	会议申请编号	false	false	签署审核意见
改变状态	Change()	public	boolean	会议申请编号	false	false	改变申请状态
办理	Handle()	public	boolean	会议申请编号	false	false	办理会议申请
放弃	giveup（）	public	boolean	会议申请编号	false	false	丢弃编写的会议申请
校验	Check()	public	boolean	会议申请编号	false	false	检验输入信息的有效性

5. 类名：会议通知

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
删除	delete()	public	boolean	会议通知编号	false	false	删除无用的会议通知
增加	add()	public	boolean	会议通知编号	false	false	增加新的会议通知
修改	update()	public	boolean	会议通知编号	false	false	修改原来的会议通知
查询	query()	public	String[]	会议通知编号	false	false	通过关键词查询

6. 类名：会议纪要

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
暂存	delete()	public	boolean	会议纪要编号	false	false	保存会议纪要
起草	add()	public	boolean	会议纪要编号	false	false	编写会议纪要
修改	update()	public	boolean	会议纪要编号	false	false	修改会议纪要
查询	query()	public	String[]	会议纪要编号	false	false	通过关键词查询
改变状态	Change()	public	boolean	会议纪要编号	false	false	改变纪要状态
归档	filing()	public	boolean	会议纪要编号	false	false	办理会议纪要
放弃	giveup（）	public	boolean	会议纪要编号	false	false	丢弃编写的会议纪要
校验	Check()	public	boolean	会议纪要编号	false	false	检验输入信息的有效性

7. 类名：人员

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
删除	delete()	public	boolean	会议人员编号	false	false	删除无用的人员
增加	add()	public	boolean	会议人员编号	false	false	增加新的人员
修改	update()	public	boolean	会议人员编号	false	false	修改原来的人员信息
查询	query()	public	String[]	会议人员编号	false	false	通过关键词查询
校验	Check()	public	boolean	人员编号	false	false	检验输入信息的有效性

8. 类名：附件

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述

删除	delete()	public	boolean	会议附件名称	false	false	删除无用的会议附件
增加	add()	public	boolean	会议附件名称	false	false	增加新的会议附件
修改	update()	public	boolean	会议附件名称	false	false	修改原来的会议附件
查询	query()	public	String[]	会议附件名称	false	false	通过关键词查询
校验	Check()	public	boolean	附件编号	false	false	检验输入信息的有效性

9. 类名：角色

操作别名	操作名	可见性	返回值	参数	抽象	静态	描述
删除	delete()	public	boolean	会议角色编号	false	false	删除无用的会议角色
增加	add()	public	boolean	会议角色编号	false	false	增加新的会议角色
修改	update()	public	boolean	会议角色编号	false	false	修改原来的会议角色
查询	query()	public	String[]	会议角色编号	false	false	通过关键词查询

第四节 会议管理类图

图 6.10 会议管理系统类结构图

第五节 操作步骤

(1) 在第四章第四节项目和包建模的基础上进行类图建模，如果要直接建立类图模型，项目和包的建模操作步骤和第四章第四节相同。

(2) 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。

(3) 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“会议管理”前面的“+”符号。

(4) 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”类图“。则在该包下增加了一个节点，一个是以”classdiagram“开头的节点，代表这个用例图全局属性，并可在“常规”选项卡中修改该节点名称为“会议管理类图”。

图 6.11 创建类图操作界面

(5) 首先从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“会议室”。在名称栏目输入“meetingRoom”，可见性栏目选择“public”，以完成类的属性设置。

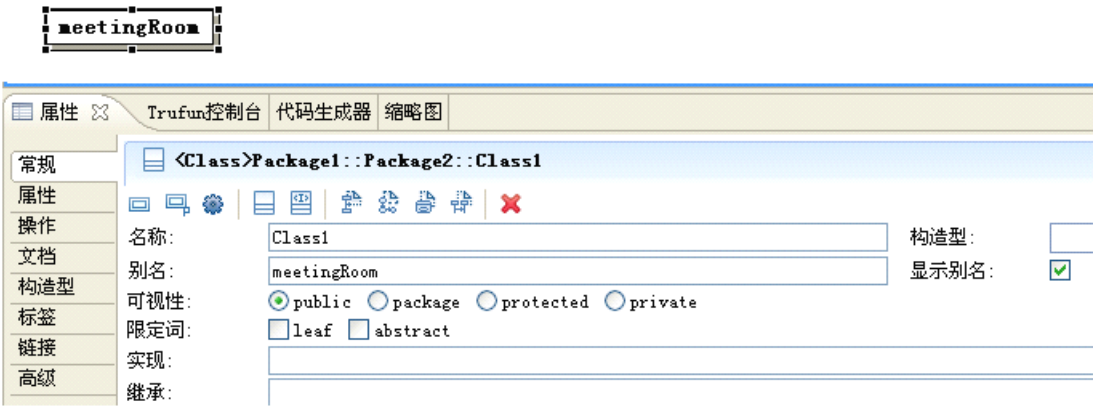


图 6.12 类元素的属性设置

(6) 右击绘图区的“会议室“类，从弹出菜单中选择“创建字段”，类图就多了一个属性项，以“field”开头，选中该属性项，在界面下部的“常规”选项卡中，别名栏输入“会议名称”，在名称栏目输入“meetingName”，可见性栏目选择“private”，选择”文档“选项卡，写入该字段的注释信息。如

果还有该字段的其他元素要输入(没有可以不做)，可在绘图区选择“会议名称”类，在界面下部的“属性“选项卡中的“mettingName“行的类型列下输入类型，默认值下输入默认值 ，以及是否是静态、只读，final 等。

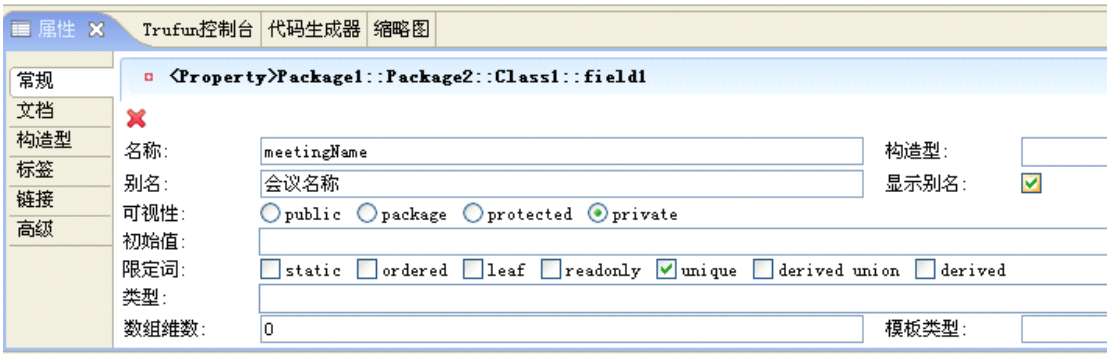


图 6.13 字段的属性设置

(7) 右击绘图区的“会议室“类，从弹出菜单中选择“创建操作"，类图就多了一个操作项，以“operation”开头，选中该操作项，在界面下部的“常规"选项卡中，别名栏输入“增加（）"，在名称栏目输入“add（）"，可见性栏目选择“public”，再选择”文档“选项卡，写入该操作的注释信息。如果还有操作的其他元素要输入(没有可以不做)，可在绘图区选择“会议名称"类，在界面下部的“操作“选项卡中的“add()“行的返回值列下输入返回值类型，参数下输入操作的参数 ，以及是否是静态、抽象、final 等。

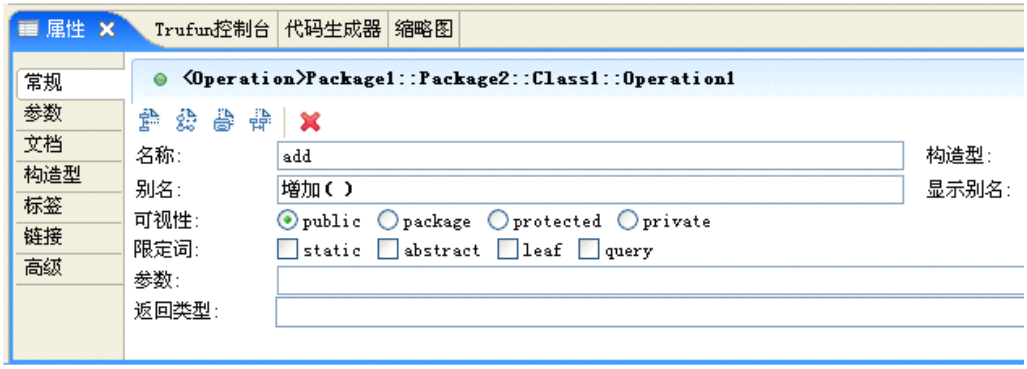


图 6.14 操作的属性设置

(8) 重复 5、6、7 步骤完成其他类的建模。

第六节 车辆管理系统类图

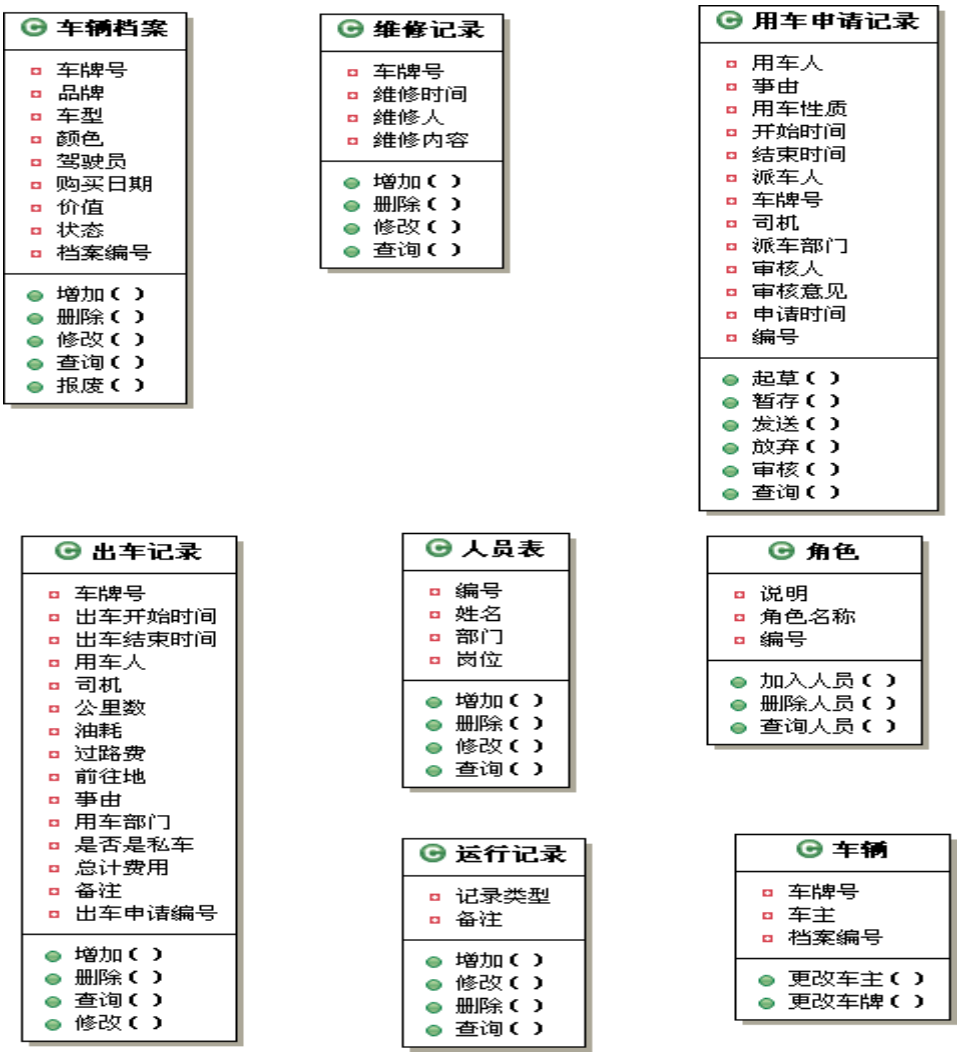


图 6.15 车辆管理系统类图

第八节 小结

类是面向对象建模的核心元素，在建模时，首先是要对原始需求进行分析，找到对象，然后再找到候选类，在这个过程中，除了前面讲过的方法外，利用以前的项目经验也是很重要，只有多做多看，才能触类旁通，游刃有余。找到类以后，系统就多了一个资源，但想让类发挥应有的作用，还需要填充类的属性和操作这些类的特征，在这个过程中，经验实践的作用也不可低估。

画图建模操作是大家学习本章必须掌握的内容，对于类其他扩展原型，如接口、实现、抽象、实体等等可在分析阶段不予考虑（与实现有关），在设计阶段，再把相应的类转

化为相应的扩展原型。

第九节 习题

1. 试说明为什么不能在分析阶段定义属性的存储类型？
2. 简述接口和类的区别？
3. 试对车辆管理系统类图建模编写操作步骤。
4. 您认为还可以从哪些方面寻找类、属性、操作。
5. 试对考勤管理系统进行类图建模。

第六章 定义类之间的关系

在上一章，我们首先用面向对象的思想寻找了对象，然后又由对象抽象出了类，为了反映这些对象的共同特征，我们从类中找到属性和操作，填充和完善了类，使我们获得了面向对象的一个重要资源——类。

面向对象分析的思想来自客观世界，在客观世界里，每个对象并不是孤立存在，要完成一件有意义的工作，对象之间就必须发生联系，也就是对象之间要发生关系，这种对象和对象之间关系组成结构被称为业务式样，它是对象结构的精髓。在分析对象和对象之间关系同时，我们还可以发现新的对象、类，新的特征（属性和方法）。这就需要对上一章的分析结果进行调整。进一步完善对象（类）的特征。从这一点看，面向对象分析也是一个迭代完善的过程。这一章我们重点讨论类和类之间的关系。

第一节 关系

如果对象之间存在着关系称为链接，那么反映这两个对象的类之间必然存在着某种语义上的联系，最基本的要求两个类之间要彼此了解，才能使一个对象通过链接向另一个对象间发送消息，若一对象获得消息，就可以调用它了解的方法，满足另一方的请求。这种传递信息的方向性被称为导航性，这种对象间链接如果上升到类这个层次就被称为关联，在有些资料把这种链接称为永久链接或实例链接（用以区别由类间的依赖关系构成的暂时链接或消息链接）。关联在我们生活中几乎无处不在，比如你和你的鞋子，为了走路的需要，你购买它，你和鞋子之间发生了购买关联；为了美观，你需要清洗鞋子，你又和鞋子发生了关联；为了….。从导航性出发，把类间关系也因此被分成了单向关联和双向关联。另外为了更精细地反映类之间的关联关系，关联又被定义为聚合和组合两种特殊情形。但在 UML 建模中，关联和聚合、组合之间是一种并列的关系，它们都有自己的建模元素。当一对对象在任意时刻存在一个唯一链接，就可以使用关联类，关联类既是关联又是类。

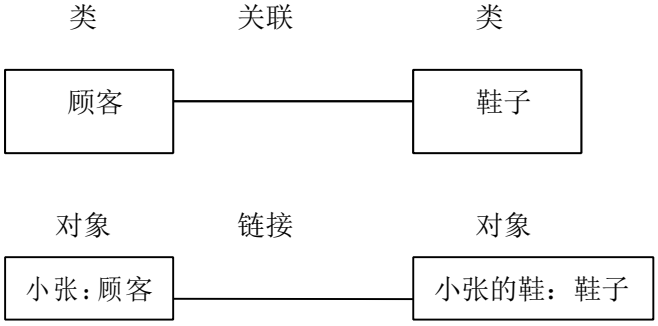


图 7.1 关联的语义示例

在多个建模元素之间，如果对一个建模元素（提供方）的改变，可能会影响接受该元素（客户方）提供信息的其他元素，这两个元素之间就存在依赖关系。依赖不仅发生在类之间，在包与包以及类与对象之间也能发生依赖关系。尽管依赖有多种类型，但在实际使用中，常用的类型有使用和实例两种。如果类 a 操作类 b 的参数；或者类 a 的操作返回类 b 的值；或者类 a 的操作在实现中使用了类 b 的对象，它们之间就是使用依赖。如果一个元素是另一个元素的实例，它们之间就是实例依赖。

如果一个类（一般类）是另一类（特殊类）的特殊情形，在任何期望一般元素的地方，都可以使用特殊元素。特殊类的每一个实例也是一般类的一个实例，特殊类也具有一般类的所有特征。这两个类之间就是一种泛化关系。泛化只针对类元素，在软件实现中用继承来实现。子类可以多种不同的方式和父类类似，根据不同的特征抽象，可以得到不同的子类组，又叫做泛化组。

第二节 关联

7.2.1 关联的模型元素

为了表示两个类之间具有关联关系，在 UML 建模中，主要定义了一下模型元素，下面我们将从语义和表示法两个方面加以介绍：

1. 关联的名称：关联的名称表示源对象正在目标对象上执行动作，是关联关系的标志和目的，采用动词来表示，一般用一条连接两个类的直线表示关联，关联的名称放在直线上的中间位置，缺省情况下关联的方向是从左到右，有时为了阅读方便，给从右向左的关联名称加一些方向性的后缀（一个指方向的小三角）。尽量不要把关联和后面讲到的角色同时都画在模型图中。关联的画图主要有正交和倾斜两种风格，这在后面的会议管理建模实例中都有反映。

图 7.2 关联命名示例

2. 端点：为了反映对象在结构中所起的作用以及参与关联的对象所遵循的规则，直线的两个端点被定义为新的模型元素：端点，端点主要反映一下信息：角色、接口说明、可见性、多重性、定序、约束、限定、导航性。下面就分别讨论这些信息的语义和表示法：
   1) 角色反映对象是如何参与关联的，和关联名称不同，角色的名称可以在后面反映在生成的代码中，每个对象都要保存一个参考值，它是对象的属性，指向所关联的对象。

图 7.3 角色表示法示例

   2) 接口说明：关联定义了类的特征的一种使用方法，一般用“角色：接口说明”来表示。
   3) 可见性：可见性的定义和前面可见性的定义相同，但它现在表示那个对象可以访问角色的名称（角色将成为类的属性），在 UML2.0 中，该属性全部为 private。
   用一个减号（-）表示，如上图教授前面的减号。
   4) 多重性：多重性事指该端点有多少个对象可以与另一个端点的一个对象关联。可以是一个给定值、离散值、或一个范围，和前面讲的属性的多重性含义相同，只是在表示时不用中括号，直接表示就行。

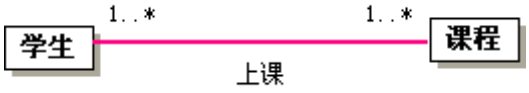


图 7.4 端点的多重性表示示例

5) 定序：指关联中一组对象按一定的顺序排列，定序用标记值 ordered 表示，只要把定序放置在相应的端点就可以了。

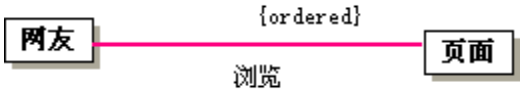


图 7.5 端点的定序表示示例

6) 约束：是为了保证系统的完整性，而对某些属性、操作添加的限制条件，可以放在定序中，用逗号隔开，也可用 OCL 表示。

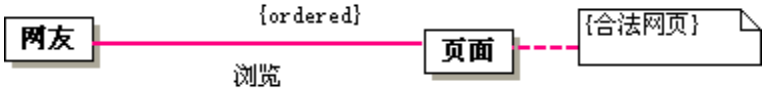


图 7.6 约束的表示示例

7) 限定：有时又叫属性关联，它不是象角色名称一样拿整个对象作为对方的属性，而是拿出对象的一个属性作为对方的属性值，但要求通过该属性值可以找到原来的对象。与数据库中的主键在含义上有些类似。

8) 导航性：导航性表示一个对象访问另一个对象的能力，用箭头来表示，如果某端点存在箭头，表示该端点是可访问的。如果不可访问，则用一个叉来表示。

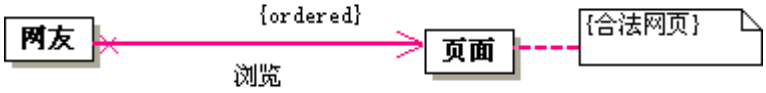


图 7.7 导航性的表示示例

9) 关联类：关联类描述的是关系，而不是象一般类那样描述的是实体，但它不是关联本身，它也可以和其他类关联，一般用虚线连接到它所描述的关联。关联和关联类的主要区别是链接有唯一标识时使用关联类。

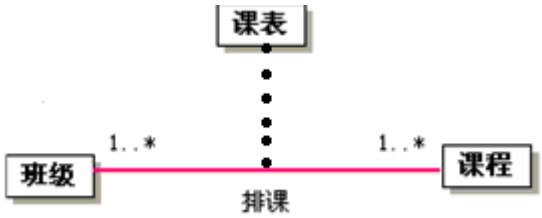


图 7.8 关联类的表示示例

10) 多元关联：如果一个关联有多个对象参与，就是多元关联，一般多元关联都用二元关联处理。

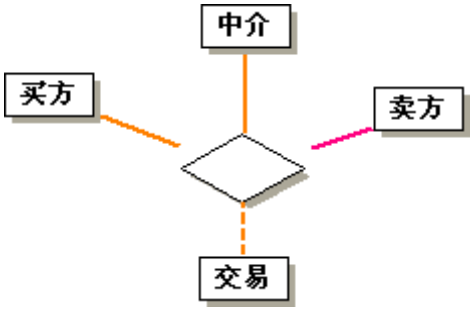


图 7.9 多元关联的表示法

7.2.2 会议管理中的关联关系建模

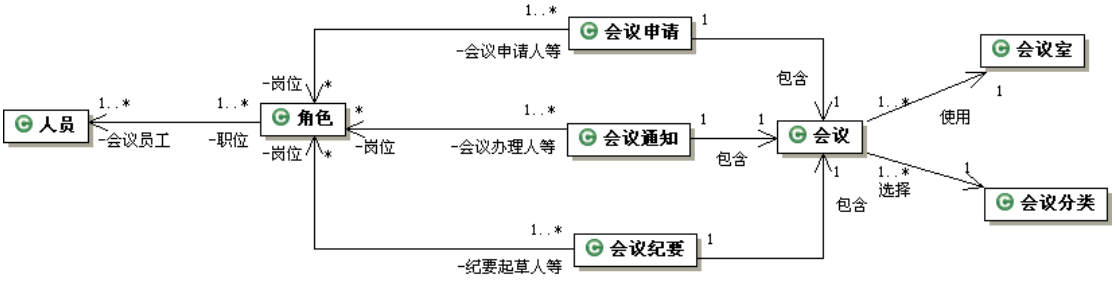


图 7.10 会议管理中部分类关联图

第三节 聚合和组合

7.3.1 聚合和组合的模型元素

（1） 聚合：

在一般的关联中，参与关联的类是独立的，它们之间除了通信关系外，不存在任何依赖关系，而聚合却不满足此要求，聚合之间有一种明显的等级关系，在聚合中，一组元素组成一个更大的元素，整体大于部分永远成立。同时，聚合关系中的对象有一个控制点，负责聚合的行为协调。聚合定义了一种构造关系，用以保护对象配置的完整性。聚合既可以是物理集合，又可以是逻辑集合。但其语义相同，由于这种特殊的聚合关系的存在，使得人们不能象使用关联一样使用这组对象。如果要对聚合类某一对象施加操作，必须通过对象控制点，外部的调用者不必关心聚合内部的情况，只需要调用对象的接口，接下来的工作聚合可以自己处理。另外，根据聚合元素之间的关系不同，聚合又分为组成聚合和共享聚合。

聚合也是表示类和类之间的关系，用于关联的建模元素，同样适用于聚合。如角色、多重性、限定、约束等。不在赘述。在画类之间的聚合关系时，首先画出类之间的关联关系，再在控制点所在的类的端点画一个空心菱形，最后加入多重性。

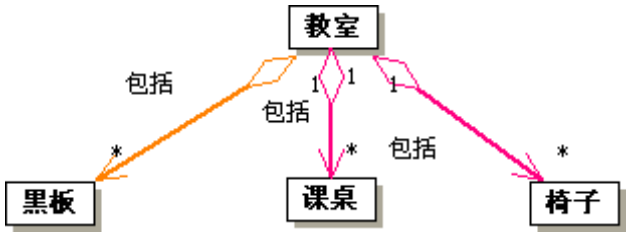


图 7.11 聚合关系表示示例

（2） 组合：

组合是聚合的一种特殊情形，在组合中，成员对象的生命周期取决于组合的生命周期，说白一点，组合管的更宽了，不仅管对象的行为，也管对象的构造和析构。成员对象不能脱离组合而独立存在。组合类不存在了，成员对象也就不存在。因此，组合又被称为强聚合，它同样具有聚合的语义，即：保护对象配置的完整性；作为一个单元在运作，通过一个控制对象，向下传递行为。在画类之间的组合关系时，首先画出类之间的关联关系，再在控制点所在的类的端点画一个实心菱形，最后加入多重性。

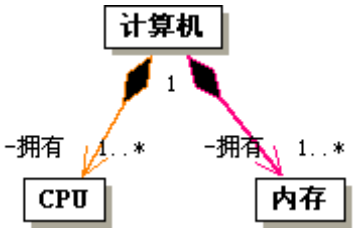


图 7.12 组合关系表示示例

7.3.2 会议管理中的聚合和组合关系建模

图 7.13 会议管理中的部分聚合组合关系建模

第四节 泛化

7.4.1 泛化的模型元素

和前面讨论的关联完全不同，它是描述类间是否有共同特征，是一般元素和特殊元素之间一种高层次依赖，它满足可替换性原则。所以泛化不能使用关联的任何模型元素。泛化是指由特殊元素，找到一般元素，与此相对的有特化，是指由一般元素找到特殊元素，这两种分析方法在面向对象分析中同时被使用。在泛化最底层的类，是不能被特化抽象。

1. 超类：包含它所有成员共同具有的特征，超类可以是抽象的，有时又叫基类、父类。

2. 子类：包含了某种类型对象独有的特征。子类不但继承超类的所有特征，而且有自己的特征。

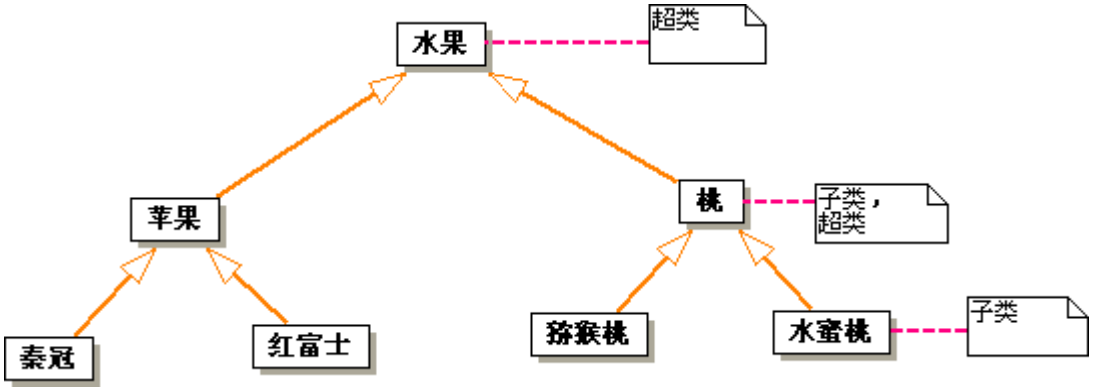


图 7.14 子类和超类的定义示例

3. 抽象类：缺乏完整定义类，不能被实例化，抽象类必须有子类，用以实现没有实现的操作。可以有不同的子类实现抽象类定义的方法，所以只有超类才能作为抽象类，抽象类的表示有两种方式，一种是把类名和操作用斜体字表示，另一种是给类名称或操作添加{abstractor}特性。

图 7.15 抽象类的表示示例

4. 多继承：UML 允许类具有多于一个的超类，子类直接继承所有它的直接超类。

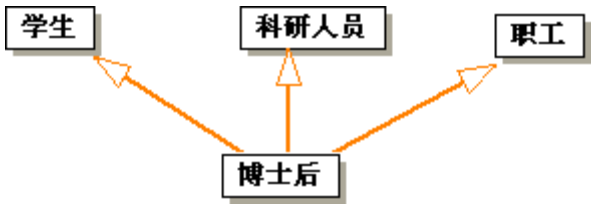


图 7.16 多继承表示示例

5. 多态：多态是建立在抽象类基础上的。其实质是不同类的对象具有相同签名的操作，但实现却不同，这是由泛化关系中抽象类所规定的契约。它使得给不同类的对象发送相同消息成为可能。

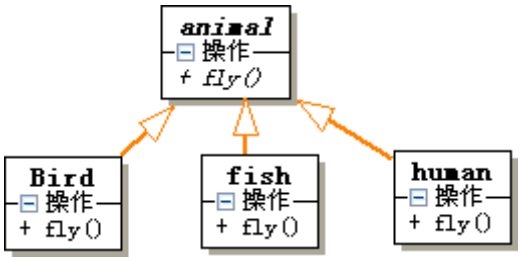


图 7.17 多态的表示示例

6. 覆盖：在子类中改写超类已经实现的操作，只要遵守超类定义的契约，在 UML 里，这种修改被认为是合理，但有些语言通过一些关键字限制覆盖。覆盖操作和被覆盖操作只有函数体 （函数名、参数列、返回值类型必须同父类中的相对

应被覆盖的操作完全一致）不同，当子类对象调用子类中该同名操作时会自动调用子类中的覆盖版本，而不是父类中的被覆盖函数版本。与覆盖容易混淆的概念是重载，它是指在同一可访问区内被声名的几个具有不同参数列的（参数的类型、个数、顺序不同）同名操作，程序会根据不同的参数列来确定具体调用哪个操作

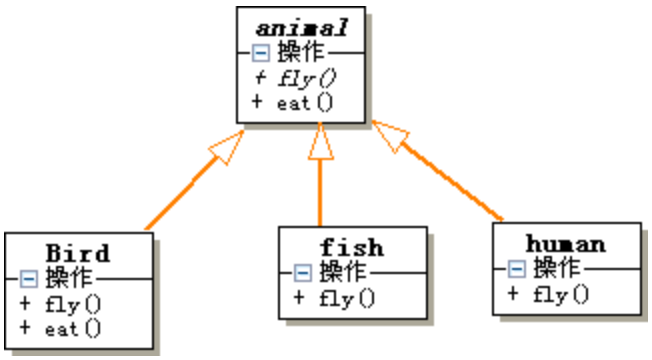


图 7.18 覆盖的语义示例

7. 泛化集合：根据特定的规则，对子类进行分组，这不同的子组被叫做泛化集合。泛化关系是用一条连接特殊类和一般类的带有空心三角形的实线来表示，箭头所指方向为该超类。空心三角形所在端点即为超类。

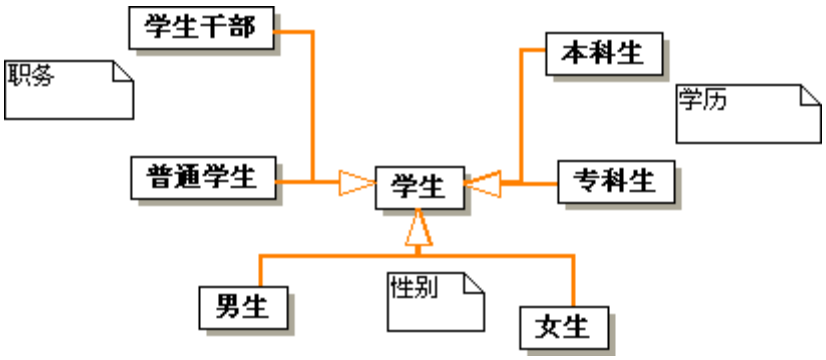


图 7.19 泛化集合的示例

8. 判别式：每个类都定义了一组特征值，这些特征值都有可能被用来定义判别式。用来描述超类挑选子类的属性或规则，常用的特征值包括属性类型、属性取值、操作、实现、关联，用它来判断不同类的对象，是否是属于同一个超类。
9. 强类型：强类型是元类，它的实例是类（而不是对象），强类型用构造形<<powertype>>来表示。强类型可以用在泛化集合中，泛化集合中的所有类，都是该强类型的实例，目前该概念在实际中使用比较少，只做一般了解就可以了。

7.4.2 会议管理中的泛化关系建模

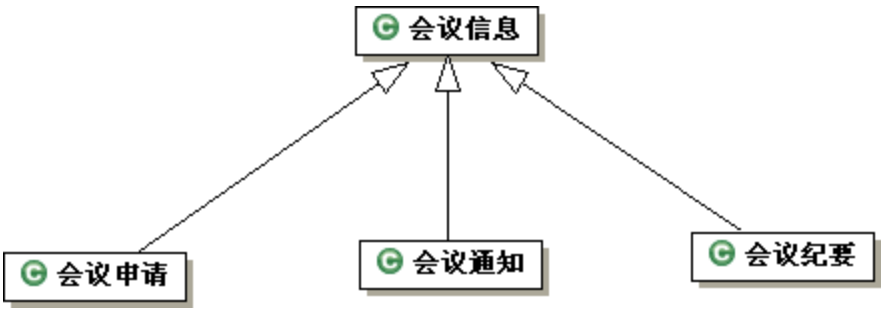


图 7.20 会议管理中的泛化关系建模

第五节 依赖性

7.5.1 依赖的建模元素

如果一个模型元素影响另外一个模型元素（这种影响不可逆），则这两个元素之间就存在着依赖关系。其中一个元素是独立的，另一个元素是非独立的，依赖于独立元素，如果独立元素发生变化，非独立元素必然要受到影响。与前面讨论的关联和泛化明显的语义差别。

在 UML 中，依赖用一条带箭头的虚线来表示，有时在虚线上有一个标签，箭头末端指向客户方，箭头所指提供方。UML 提供了一些标准的依赖类型。同时，建模时也可以根据需要自定义一些依赖类型，其表示方式采用构造形。

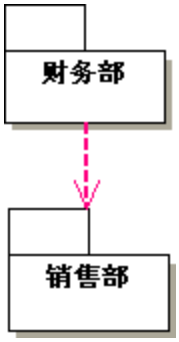


图 7.21 依赖关系表示示例

1. trace：反映不同模型历史关系依赖，如需求原始资料和模型之间就是这种关系，用<<trace>>表示。
2. refine：反映相同模型不同版本之间的关系，用<<refine>>表示。
3. derive：由提供方的值计算得到客户方的值得的关系，用<<derive>>表示。
4. bind：将模版参数绑定成实际值得以创建一个非参数化的元素，用<<bind>>表示 。
5. access：作为提供者的包，允许客户包访问，反映包间的关系，用<<access>>表示。
6. import：提供者的命名空间被整合到客户的命名空间，用<<import>>表示。
7. substitute: 客户方在运行时可以替代提供方，用<<substitute>>表示。

8. permit：客户方可以访问提供方的任何元素，用<<permit>>表示。
9. instantiate：客户方是提供方的实例，用<<instantate>>表示。
10. send：客户方把提供方发送到指定目标，用<<send>>表示。
11. parmeter：提供方是客户操作的参数，用<<parmeter>>表示。
12. call：客户方操作调用提供方的操作，用<<call>>表示。
13. use：客户方以某种方式使用提供方，用<<use>>表示。

7.5.2 会议管理中的依赖关系建模

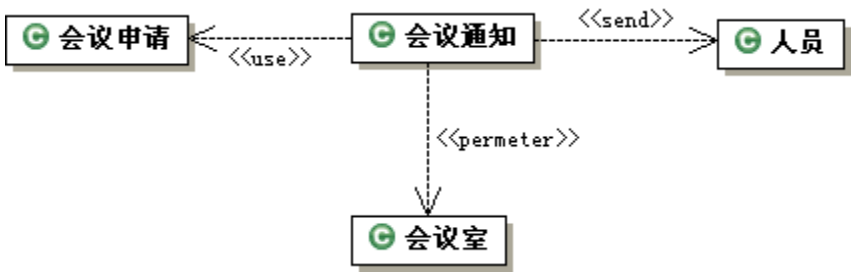


图 7.22 会议管理中的部分依赖关系建模

第六节 会议管理中的类关系图

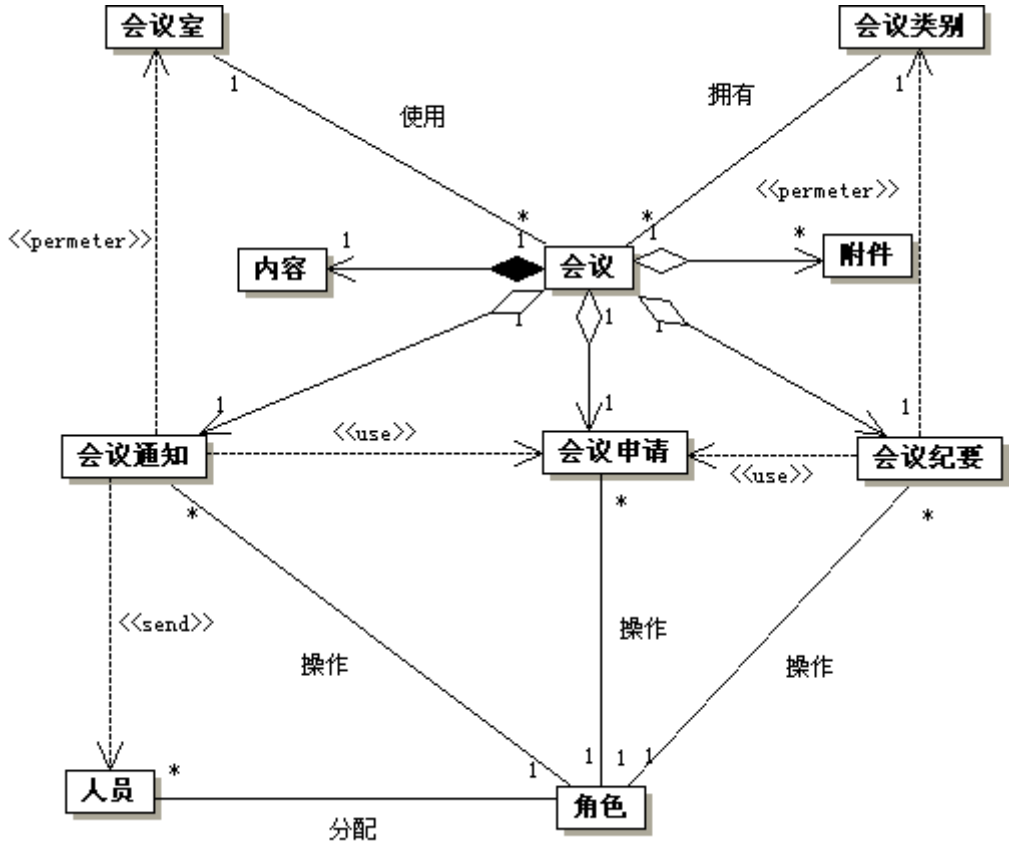


图 7.23 会议管理类关系图

7.6.1 建模分析步骤

1. 对所有候选类列表，寻找它们之间的实例连接关系，即首先找到关联关系。
2. 寻找候选类之间的整体和部分关系，确定聚合和组合关系。
3. 寻找候选类之间的特殊和一般关系。确定类之间的泛化关系。
4. 寻找候选类之间的提供和使用关系。确定类之间的依赖关系。
5. 用前面的分析完善类图。
6. 绘出类关系图

补充每个类的的分析文档，为类的进一步设计打下基础。

7.6.2 会议管理分析过程

在前面分析类图的基础上，下面我们分析类之间的关系，类和类之间的关系，是构成业务的基础，首先我们找类之间的静态关联关系，在会议管理子系统中，主要分析类需要得到其他类的支持，才能完成其生命周期，会议使用会议室，会议可以分类，这就可以确定会议对象和会议室对象、会议分类对象之间存在关联，一次会议按理来说可以使用一个或多个会议室（如电话会议、广播会议、视频会议等），由于系统未提出这样的需求，我们认为一次会议使用一个会议室符合会议管理需求，一个会议室可以给多个会议使用，所以，会议对象和会议室对象之间的多重性是一对多。会议对象拥有会议类别，一个会议对象拥有一个会议类别，而一个会议类别拥有多个会议对象，同理可以找出其他对象之间的关联属性和多重性。

一个会议对象可以包含附件，也可以不包含附件，它们之间是聚合关系，会议对象可以从另一个角度观察，是由会议申请对象、会议通知对象、会议纪要对象组成，它们之间也是聚合关系，一个会议对象不可能没有会议内容对象，所以它们之间是组合关系。

会议通知对象在执行发送操作时要使用会议申请对象的信息，同时要知道接收人员信息，以及开会的会议室信息，所以它和这几个对象之间分别存在着使用依赖、发送依赖以及参数依赖关系，会议纪要对象在执行归档操作时，要使用会议类别对象、会议申请对象的信息，所以它们之间存在着参数依赖和使用依赖关系。

7.6.3 会议管理的识别元素

关联关系:

会议对象和会议室对象、附件对象、会议类别对象、会议内容对象是关联关系；会议申请对象、会议纪要对象、会议通知对象和会议对象是关联关系。会议纪要对象、会议通知对象、会议申请对象和角色对象是关联关系。人员对象和角色对象是关联关系。

聚合组合关系：

会议对象和附件对象、会议申请对象、会议纪要对象、会议通知对象之间是聚合关系，会议对象和内容对象是组合关系。

依赖关系：

会议通知对象和会议室对象、会议申请对象、人员对象之间是依赖关系，会议纪要和会议分类之间是依赖关系。

第七节　车辆管理中的类关系图

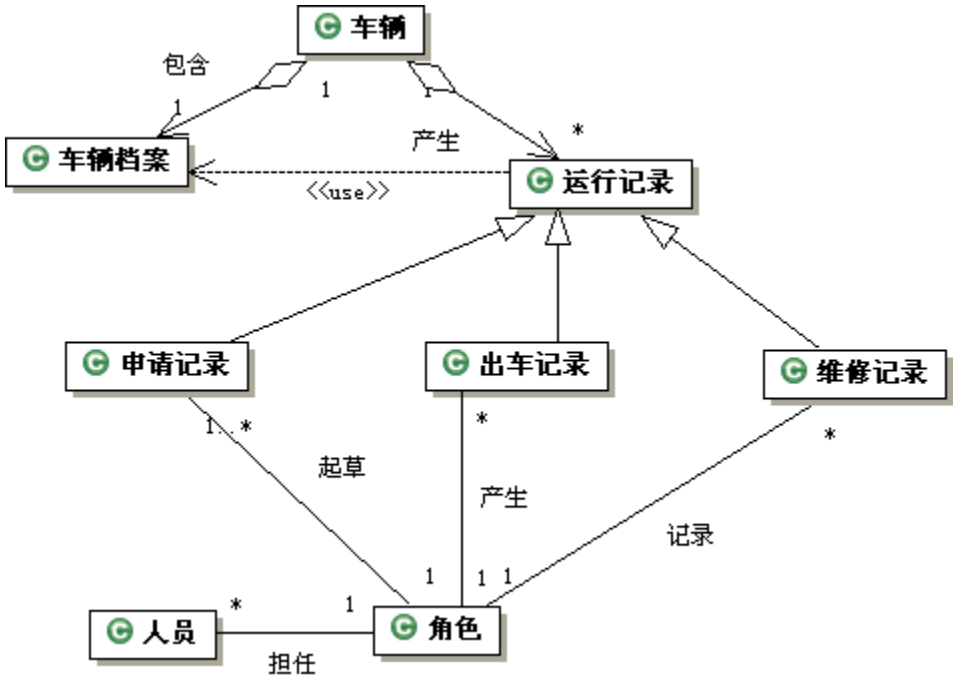


图 7.24 车辆管理类关系图

第八节　操作步骤

1. 在第四章第四节项目和包建模的基础上进行类关系建模，如果要直接建类关系模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“会议管理”前面的“+”符号。
4. 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”类图“。则在该包下增加了一个节点，一个是以”classdiagram“开头的节点，代表这个类关系图全局属性，并可在“常规”选项卡中修改该节点名称为“会议管理类关系图”。一般为了简介清晰，针对类之间的不同关系而分成不同的关系类图。

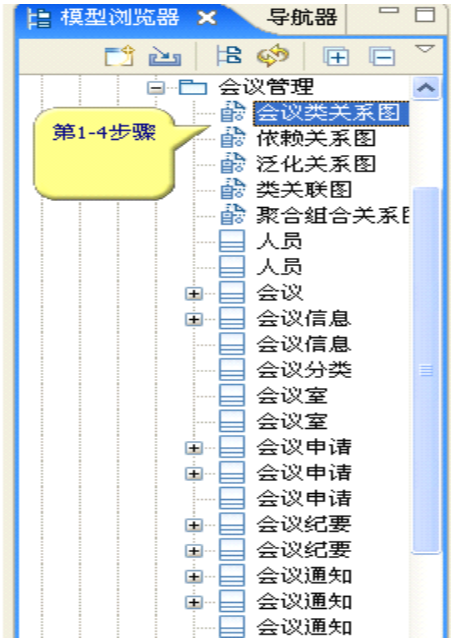


图 7.25 类关系建模操作步骤 1-4 界面

5. 首先从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“会议室”。在名称栏目输入“meetingRoom”，可见性栏目选择“public”，以完成类的属性设置。同理可完成类：会议、会议类别、员工、角色、会议申请、会议通知、会议纪要的布局。
6. 再从绘图工具面板选中关联元素，将其拖入绘图区倾斜连接会议和会议室两个类，在关联斜线选中的情况下，在下面的“常规”选项卡中的别名输入框输入“使用”关联名称。然后再点击角色 A 后面的会议室，在链接的新页面的多重性下拉框选择“1..*”，这样就完成了会议和会议室关联建模。同样操作，可完成会议和会议类型两个类关联建模。

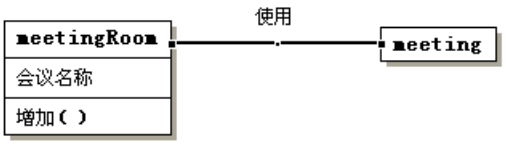


图 7.26 关联及其属性设置

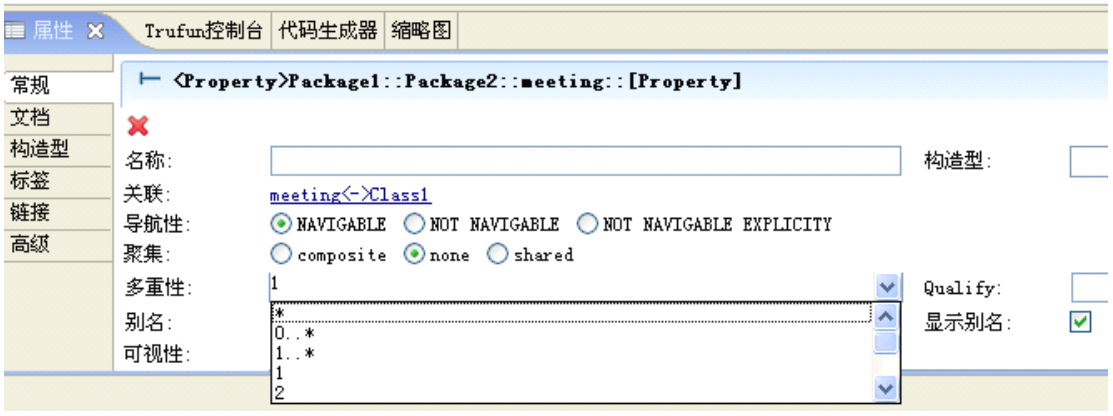


图 7.27 角色的多重性设置

7. 如果想用正交线和角色表示关联，其操作和上面基本相同，只是在用关联连接两个类时，需要把连线调节成水平和垂直线状态，同时还要在点击角色 A（B）后面的类名后，在链接的新页面的名称下输入角色名称。

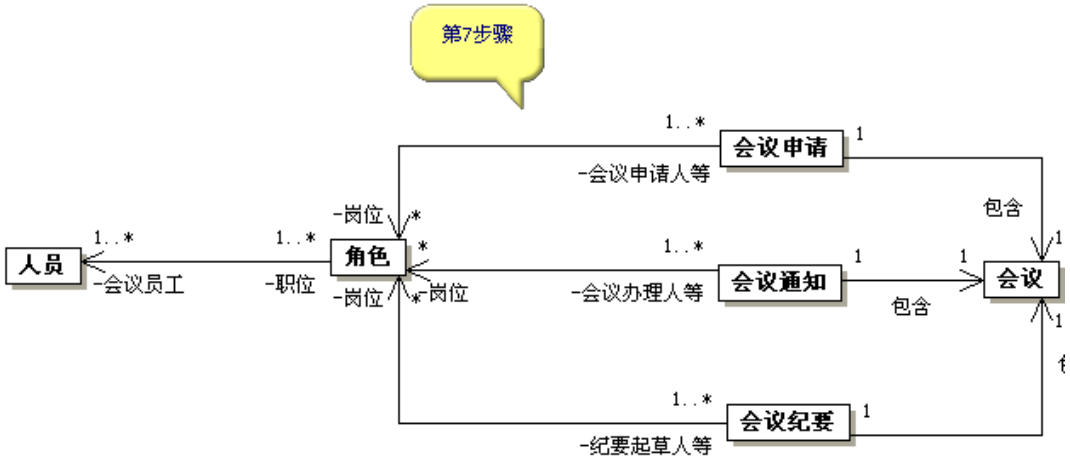


图 7.28 关联的正交线和关联的角色表示

8. 如果是关联类，在要建立关联类的两个类布局后，从绘图工具面板选中关联类，并把其拖到这两个类中间，用从绘图工具面板选中的关联端点连接左边的类，再用从绘图工具面板选中的关联端点连接右边的类。

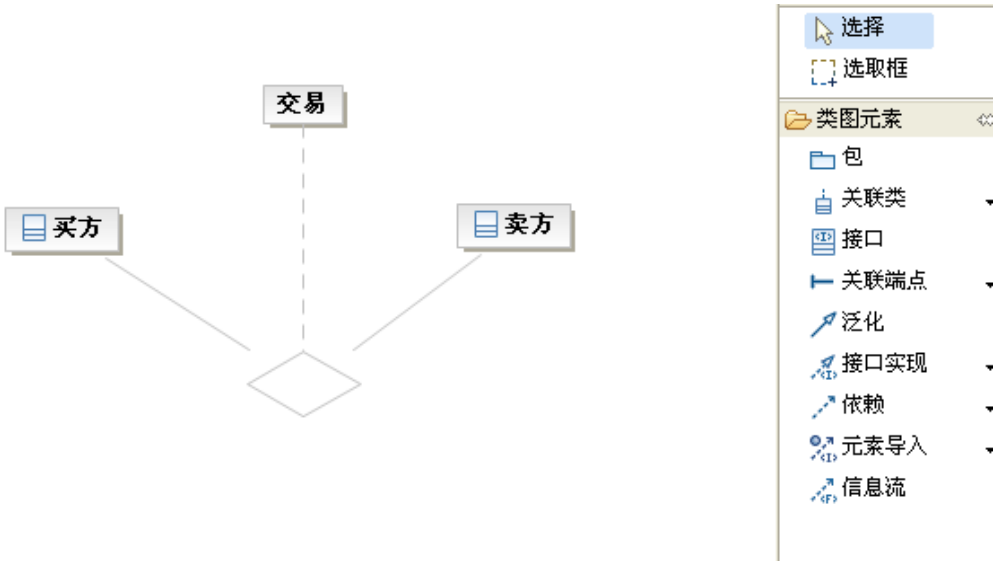


图 7.29 关联类创建操作界面

9. 对剩余的类关系，从 6、7、8 几个情况中选择合适的操作。相同操作完成关联建模。
10. 接着第 4 步完成类关系图定义之后，把分析新得的类和要建聚合组合关系的类进行布局。首先从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“会议信息”。在名称栏目输入“meetingInformation”，在可见性栏目选择“public”，以完成类的属性设置，同样操作可以完成其他类的布局。
11. 再从绘图工具面板选中组合元素，将其拖入绘图区倾斜连接会议信息和内容两个类，在组合斜线选中的情况下，在下面的“常规”选项卡中的别名输入框输入“包含”组合名称。然后再点击角色 A 后面的会议信息，在链接的新页面的多重性下拉框选择“1”，同样也可完成内容的多重性输入。这样就完成了会议信息和内容

组合关系建模。如果还有其他的类之间具备组合的关系，采用同样操作就可完成建模。

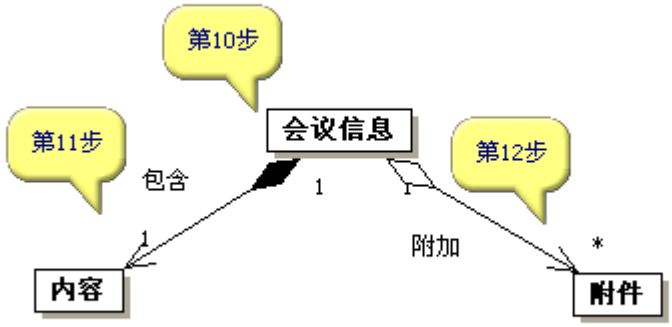


图 7.30 组合和聚合的绘图界面

12. 再从绘图工具面板选中聚合元素，将其拖入绘图区倾斜连接会议信息和附件两个类，在聚合斜线选中的情况下，在下面的“常规”选项卡中的别名输入框输入“附加”聚合名称。然后再点击角色 A 后面的会议信息，在链接的新页面的多重性下拉框选择“1”，同样也可完成附件的多重性输入，这样就完成了会议信息和内容聚合建模。如果还有其他的类之间具备聚合的关系，采用同样操作就可完成建模。
13. 接着第 4 步完成类关系图定义之后，把分析新得的类和要建泛化关系的类进行布局。首先从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“会议申请”。在名称栏目输入“application”，可见性栏目选择“public”，以完成类的属性设置，同样操作可以完成其他类的布局。
14. 再从绘图工具面板选中泛化元素，将其拖入绘图区连接会议信息和会议申请两个类。如果还有其他的类之间具备泛化关系，采用同样的操作就可完成建模。

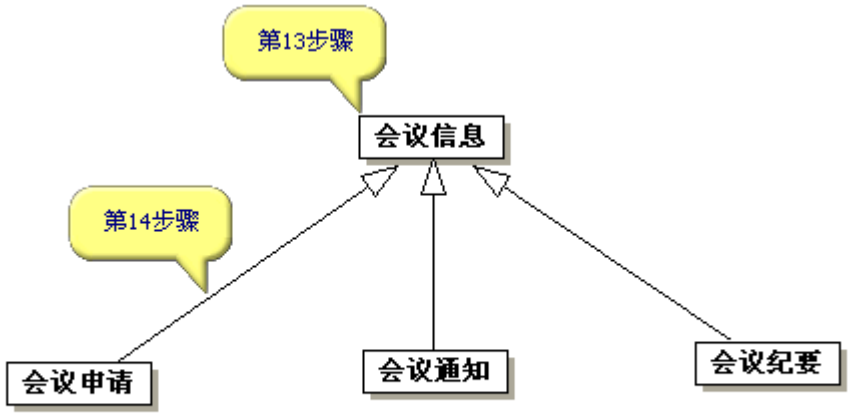


图 7.31 泛化的建模界面

15. 接着第 4 步完成类关系图定义之后，把分析新得的类和要建依赖关系的类进行布

局。首先从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“会议通知”。在名称栏目输入“notice”，可见性栏目选择“public”，以完成类的属性设置，同样操作可以完成其他类的布局。

16. 再从绘图工具面板选中依赖元素，将其拖入绘图区连接会议通知和人员两个类。在依赖虚线选中的情况下，在下面的“常规”选项卡中的构造型输入框输入“send”依赖类型。如果还有其他的类之间具备依赖关系，采用同样的操作就可完成建模。

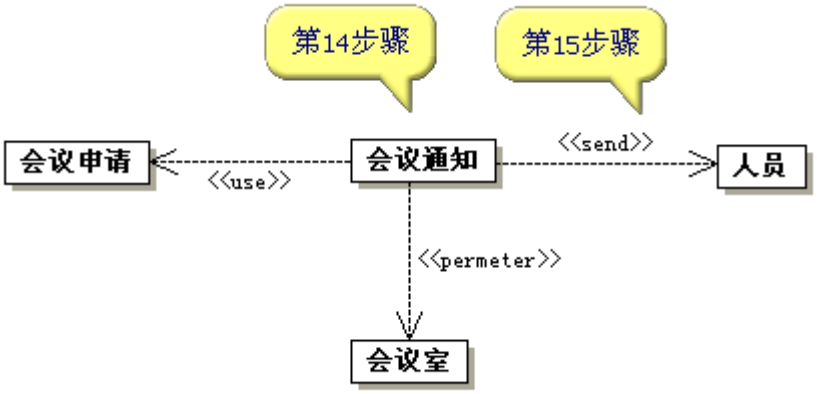


图 7.32 依赖的建模界面

第九节 小结

类间关系建模是上一章类图建模的延续，静态分析主要针对业务领域分析，类关系建模是静态分析的重点，对象间链接，是类间关系的具体体现。关联是从语义层面反映类间的关系，聚合和组合是在关联的基础上，再从组成关系和生命周期两个方面更进一步反应类间的关系。泛化是从类间是否具有共同特征这个层面来描述类间的关系，依赖则从完整性角度，反映类间的关系，有时在实际工作中，类间关系不止一个，可能构成一个关系环，这会增加后面分析的复杂度，应尽可能避免（通过增加约束分解成简单关系）。另外，对于类关系中的一些重要元素如：名称、角色、多重性、导航性、使用依赖、参数依赖要重点关注。

第十节 习题

1. 试举例说明链和关联的关系
2. 试从你的周围找出一些组合和聚合的例子。
3. 编写车辆管理系统类关系建模的操作步骤。
4. 给办公用品管理案例建立类关联关系图。
5. 对固定资产管理案例建立组合聚合关系图。
6. 对考勤管理建立泛化关系图。
7. 对图书管理系统建立依赖关系图。

第七章 对象交互

前两章我们从结构方面，静态地分析了对象以及它们之间关系，要完整地反映一个对象的本质，光有静态分析是不够的，还必须进行动态分析。动态分析主要关注对象在完成某些操作任务时所进行的通信，包括：对象如何响应用户动作，数据完整性、流程怎样控制，以及数据的存储等等，同时还涉及到对象的创建和销毁。

交互是一种行为，这种行为由语境中的一组对象为达到某一目的而交换的一组消息而构成，消息是对象之间进行通信的规约，语境指对象交互所在的系统、子系统、类、构件、节点或操作；既可以特定对象之间的交互，也可以描述扮演一定角色的原型化事物之间的交互，链是对象间的语义连接，指明了对象间发送消息的路径，该路径可以是关联可见、全局可见、局部可见和参数可见，原型化的链称为连接件。一个对象向另一个对象发送消息，这个对象又向下一个对象发送消息，一直传下去，这个消息流称为序列。

在 UML 中，常用顺序图、通信图、状态图以及前面讨论过的活动图来反映系统的行为，它们是运行系统的一个快照，在这个过程中，可以发现一些新的信息，这意味着需要修改前面的设计，比如前两章讨论的类图和类关系图，它的实例成为用例实现的基础，但光有这些类还不够，在分析中我们会发现新的类不断增加进来，需要我们及时修改前面的分析类图。

如果我们把一个动态运行的软件系统比作一个实际发生的戏剧故事，软件系统在运行时，每时每刻都有对象在发生变化，每时每刻都有对象在和其他对象进行消息通信，如果把每时每刻的信息都表示出来，就像戏剧故事在表演时把主人公的每个生活细节都表现出来一样。结果使那个观众也不知道导演要表现主人公什么信息。只有把有代表性的故事情节变为场景，并对主人公的个性、行为进行抽象，形成一个个观众感兴趣的模版，把这些不连贯的模版连在一起，观众就会在大脑中形成一个完整的戏剧故事。对动态运行的软件的描述也是采用同样的方法，需要寻找有代表性的业务场景，抽象对象以及对象之间的消息通信，通过交互建模，形成一个个有意义的业务模型，开发人员和用户通过对这些分离模型的观察，就会对系统形成一个完整的动态视图。

另外，在设计中，常常用中心机制来归类系统的行为，如：交互机制、持久化机制等，机制又决定系统的架构，从这个意义上说，动态分析与系统架构有关。

从前面的用例分析我们知道，用例是反映用户和系统之间的交互，现在又讲对象如何响应用户动作，它们之间有什么关系呢？事实上，我们本章所讨论的内容，是前面用例建模中低级用例的一个细化和精华，是用例建模的延伸，与前面所讨论的用例的定义不同，它反映是用例如何实现的。

第一节 健壮性分析

从前面的讨论我们知道，用例图反映的是用户和系统之间的交互，而对象交互反映的是系统内部是怎样支持系统和用户交互的，由于用例不是面向对象的概念，而对象

是面向对象的概念，它们之间是怎样转化的呢？很显然，如果突然冒出几个对象，并画出了它们之间的交互图，首先让人感到在分析逻辑上不够严谨，其次让使用者无法掌握，有没有好的解决办法呢？回答是肯定的。

事实上，作为 UML 三大创始人的 Ivar Jacobson 早在 1991 就提出了一种叫做健壮性分析的技术，来弥补这个空挡。该分析的主要目的就是寻找用例由那些对象支持，同时也能帮助分析人员验证以前的用例描述是否准确和完整，但由于它不是 UML2.0 的核心内容，尚未引起人们的足够重视。

8.1.1 健壮性分析的建模元素

1. 边界对象：一个系统要和系统外的其它事物进行信息交流，需要通过边界对象，这就象一个国家的海关和外交部一样，常见的边界对象比如窗口、页面，计算机对外接口等，主要分为二大类，即与用户交互的人机边界，和与其他系统（或子系统）交互的系统（或子系统）边界。边界对象一般为名词，可以用一个空心圆加一个手柄形状的直线来表示。

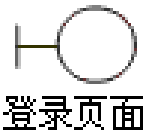


图 8.1 边界对象表示法示例

2. 控制对象：控制对象是为了满足面向对象设计的要求，而特意构造的一种对象，这种对象经常被用来封装一些复杂的控制行为，可能变化的行为以及需要事务管理的行为；或者作为一种桥梁，连接边界对象和实体对象。控制对象中一般包含一个关键动词，可以用一个空心圆并在圆边上标一个逆时针箭头来表示。

图 8.2 控制对象表示法示例

3. 实体对象：要在永久保存媒体上永远保存的“类”的快照或映像，从概念上严格地讲，该媒体指的是面向对象数据库。该对象的数据来自边界对象，该对象既可以来自动态模型，也可以来自静态模型，并不是所有的对象都要持久化，也不是对象的所有属性都要做映射并保存到数据库。实体对象又叫持久对象。实体对象一般为名词，可以用一个圆下面划一条直线来表示。

图 8.3 实体类的表示示例

4. 角色：就是该业务用例的使动者，和用例中该角色的定义和表示法一致。

图 8.4 角色的表示法示例

5. 通信：表示建模元素之间命令的传递方向，常用一个箭头来表示，可以单向也可以是双向。一般来讲， 参与者只同边界对象交互； 边界对象只能同控制对象和角色交互；实体对象只能同控制对象交互；控制对象可同边界对象，实体对象以及其他控制对象交互，但不能同角色交互。

图 8.5 单向通信和双向通信表示示例

8.1.2 健壮性分析建模步骤

1. 完成业务用例图的分析，并编写了用例描述文档。
2. 从用例描述文档中逐句找出边界对象。
3. 从用例描述文档中逐句找出控制对象。
4. 从用例描述文档中逐句找出实体对象。
5. 先画出业务用例角色。
6. 在按照边界对象、控制对象、实体对象的顺序依次画出这几类对象。
7. 根据用例描述的步骤，业务角色首先向边界对象发出通信命令。
8. 边界对象再将通信命令发给控制对象，其余命令的发送只能按照上面在通信定义中所讲的原则和用例描述中的顺序进行命令发送。
9. 用 UML 建模工具绘出健壮性图。
10. 给健壮性图补充必要的说明文档。

8.1.3 健壮性建模案例

1. 下面让我们以会议管理中的起草会议申请用例作为案例进行健壮性建模，首先让我们重温一下起草会议申请用例描述：

   用例名称：起草会议申请

   参与者：会议申请人

   前置条件：会议申请人有条件通过网络访问系统并已成功地登录系统

   后置条件：系统保存一份新的会议申请。

   基本事件流：1.用户通过网络登录后成功访问系统。

   2.用户选择会议管理后，再选择浏览会议信息。

   3.浏览结束后用户选择查看暂存会议申请。

   4.在确认无合适的会议申请后，用户选择起草会议申请。

   5.用户输入会议申请的相关信息。

   6.会议申请经过校验后提交办公室主任。

可选事件流：1.用户发现有可用的暂存申请可以修改，系统进入修改会议用例
2.新起草的会议申请被暂存
异常事件流：1.会议室已被预定，给出错误信息提示。
2.会议信息校验失败，给出错误信息提示。

2. 从上面的描述中，我们可以找到下面主要对象：
边界对象：登录页面，主页面，会议信息列表页面，暂存会议申请列表页面，会议申请起草页面。
控制对象：登录验证，会议申请校验、主页面导航。
实体对象：用户，会议信息，会议申请，暂存会议申请。

3. 会议申请健壮性分析建模

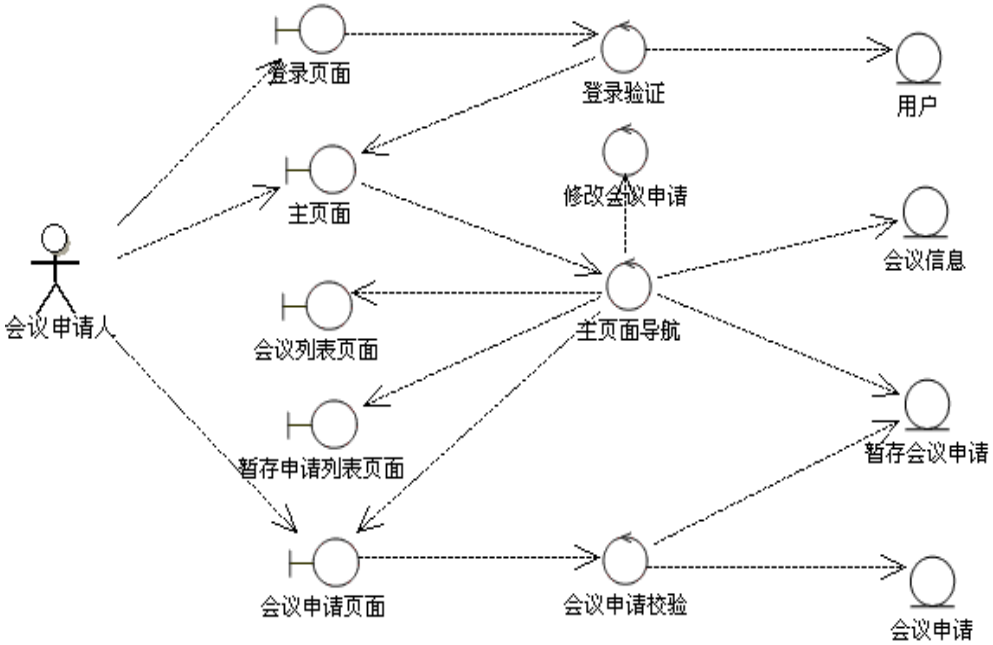


图 8.6 健壮性建模示例

4. 页面实例

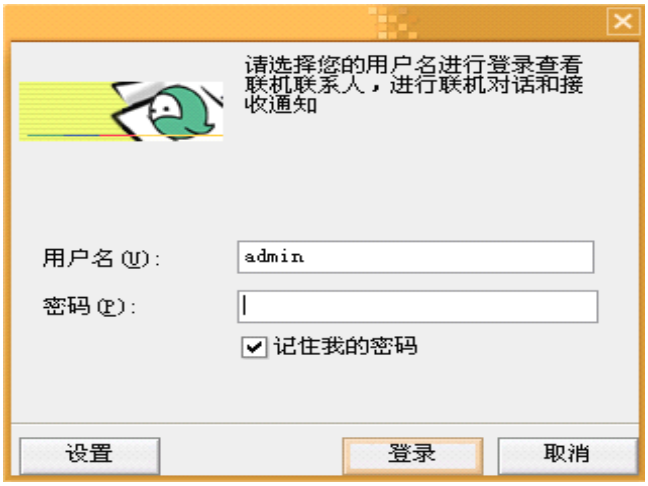


图 8.7 登录页面实例

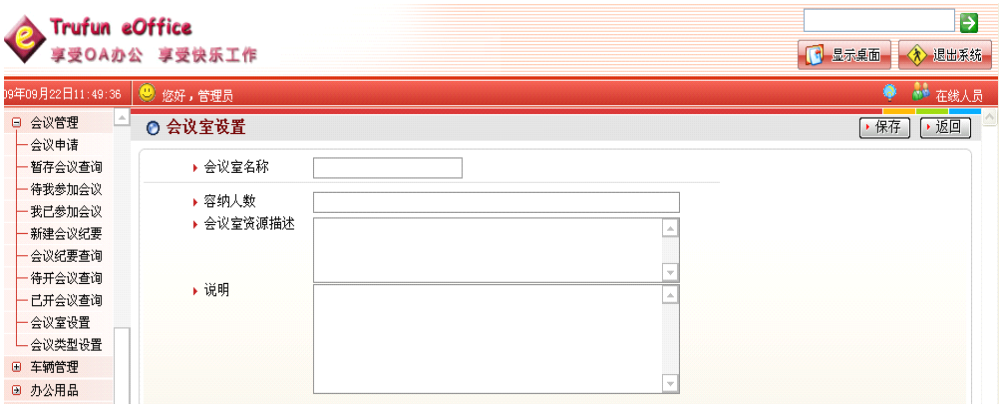


图 8.8 会议管理主页面

图 8.9 待开会议列表页面

图 8.10 暂存会议申请列表页面

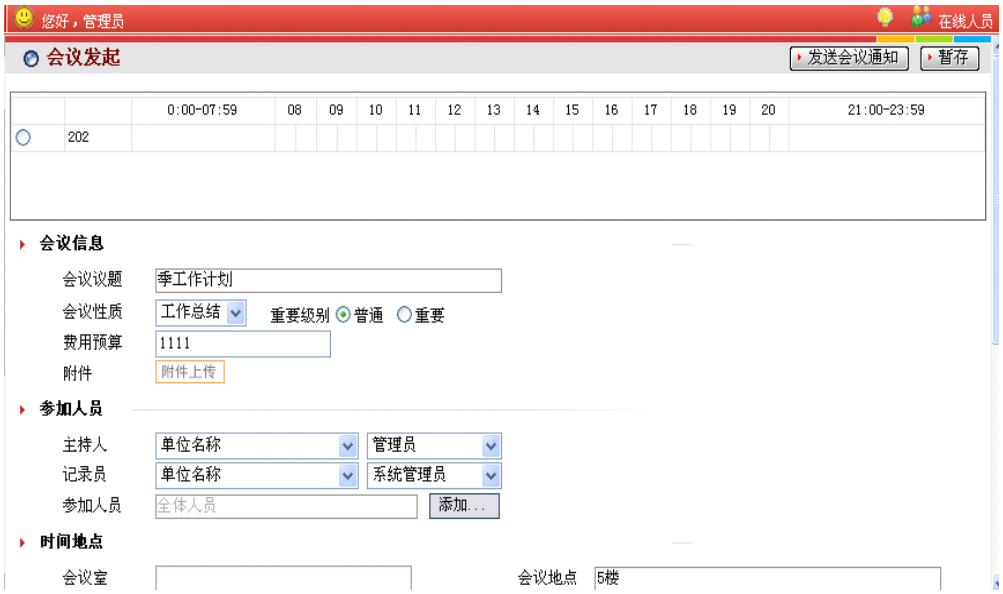


图 8.11 会议申请页面

8.1.4 操作步骤

1. 在第四章第四节项目和包建模的基础上进行类关系建模，如果要直接建健壮性分析模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“会议管理”前面的“+”符号。
4. 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”类图“。则在该包下增加了一个节点，一个是以”classdiagram“开头的节点，代表这个类关系图全局属性，并可在“常规”选项卡中修改该节点名称为“会议管理健壮性分析图”。一般为了简介清晰，它是通过动态分析发现的类图。

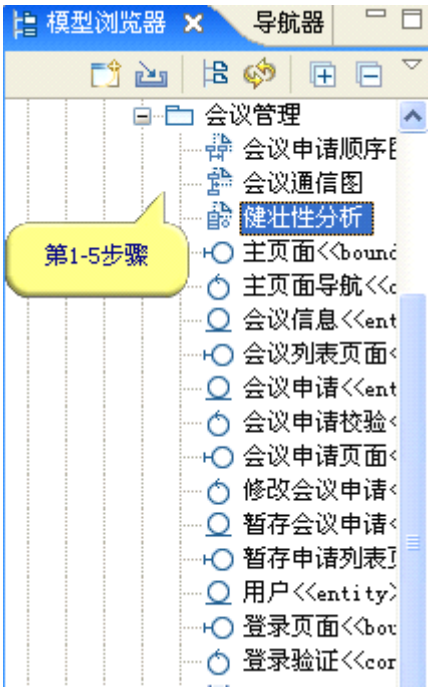


图 8.12 操作步骤 1-4 对应界面

5. 首先从导航树上选择“会议申请人”元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“会议申请人”。在名称栏目输入“applicateperson”。

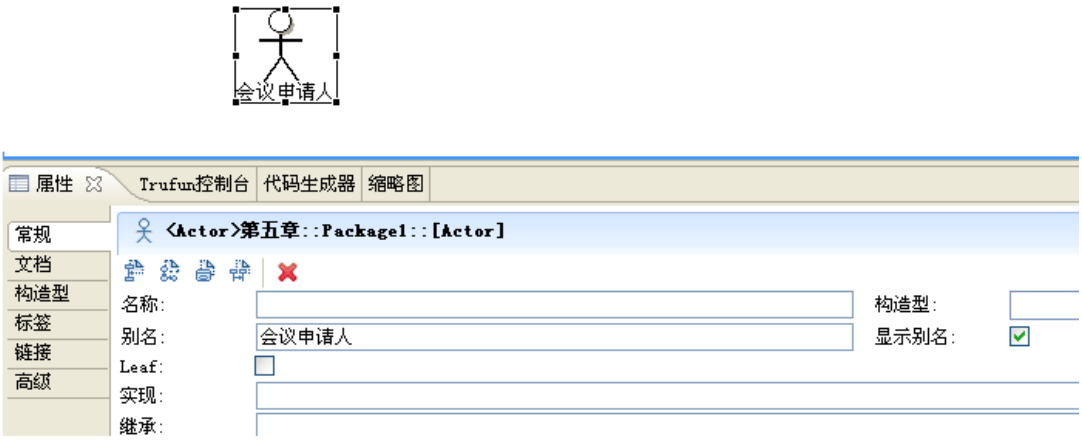


图 8.13 使动对象属性设置

6. 再从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“登录页面”。在名称栏目输入“loginpage”，在绘图区该元素处于选中的前提下，在构造型下拉框中选择“boundary”。同理可完成界面对象：主页面，会议信息列表页面，暂存会议申请列表页面，会议申请起草页面的布局。

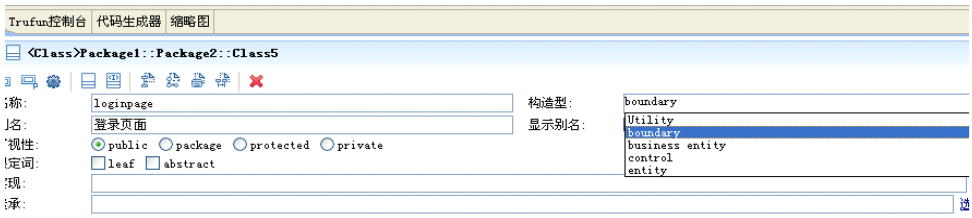


图 8.14 边界类的属性设置

7. 再从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“登录验证”。在名称栏目输入“check”，在绘图区该元素处于选中的前提下，在构造型下拉框中选择“control”。同理可完成控制对象：会议申请校验、主页面导航的布局。

图 8.15 控制类的属性设置

8. 再从绘图工具面板选中类元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“用户”。在名称栏目输入“user”，在绘图区选择该元素选中该元素的前提下，在构造型下拉框中选择“entity”。同理可完成实体对象：会议信息，会议申请，暂存会议申请的布局。

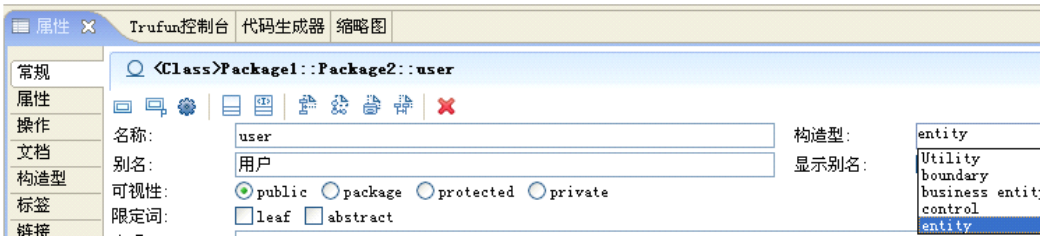


图 8.16 实体类的属性设置

9. 再从绘图工具面板选中依赖元素，将其拖入绘图区倾斜连接会议申请角色和登录页面对象，再用依赖元素连接登录页面和登录验证控制对象。最后根据用例的描述顺序和前面在理论中讲过的原则，连接其他对象，以此类推。

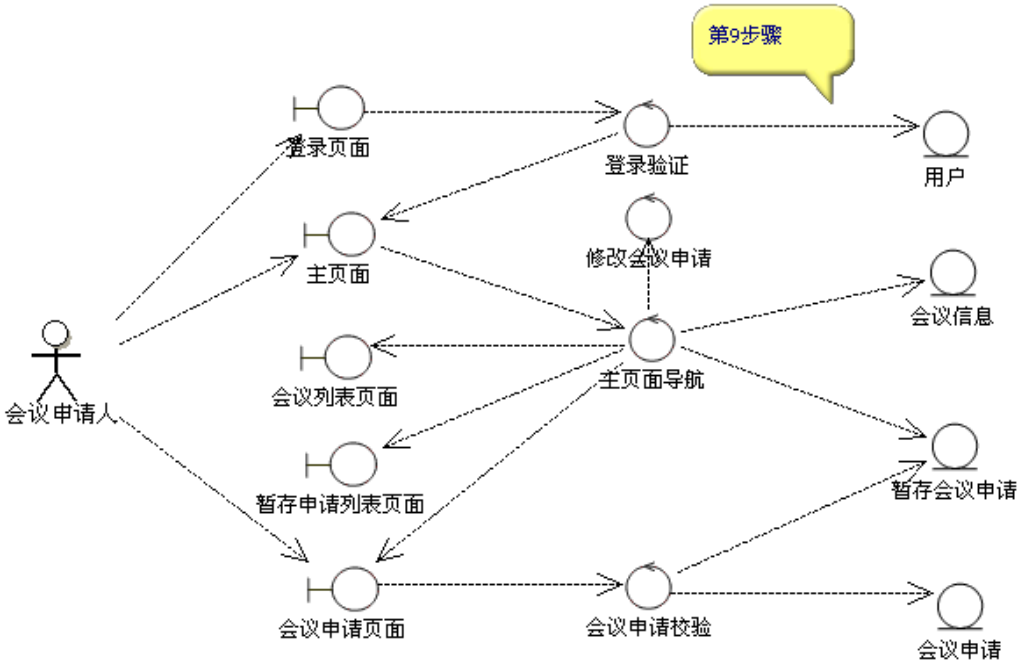


图 8.17 健壮性建模中的依赖元素示例

10. 整理在健壮性分析中新发现的对象，经过合并归类以后形成新的类，并把它添加到类结构图中。

8.1.5 车辆管理中车辆申请健壮性分析建模

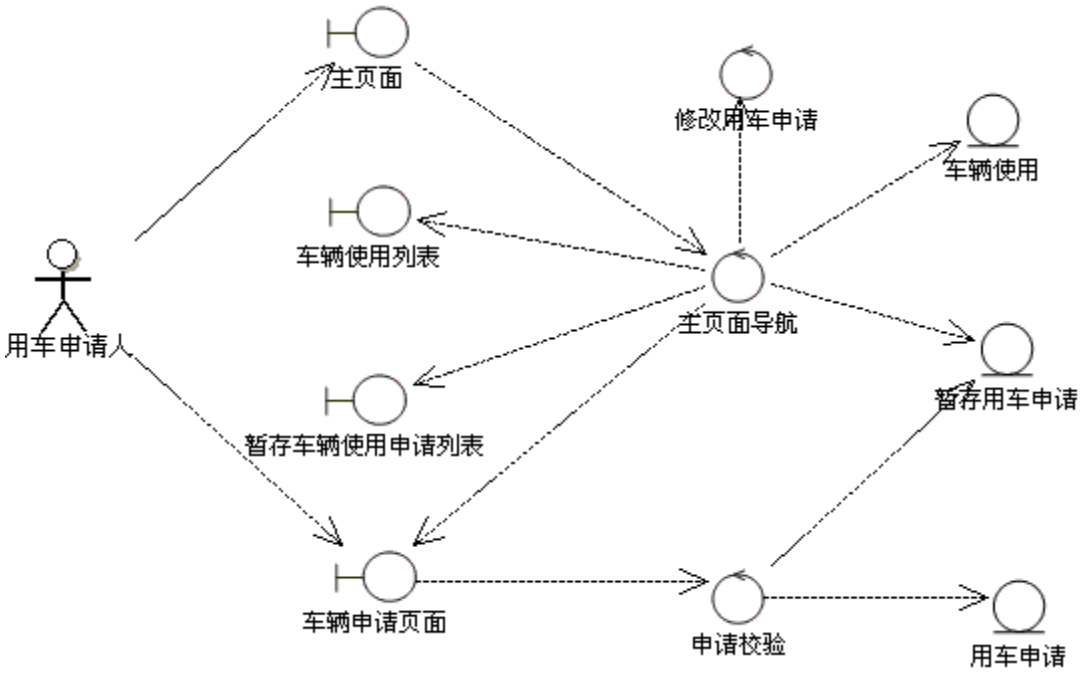


图 8.18 车辆申请健壮性示例

图 8.19 车辆管理主页面

图 8.20 车辆使用情况列表页面

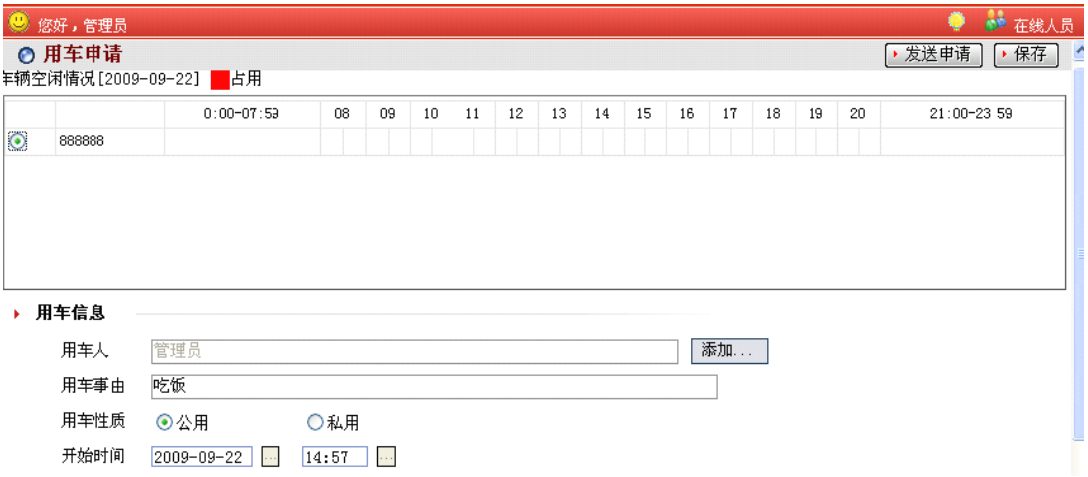


图 8.21 用车申请部分页面

图 8.22 暂存用车申请页面

8.1.6 其他子系统主要界面

1. 考勤子系统：

图 8.23 考勤管理子系统主页面

图 8.24 请假登记页面

图 8.25 考勤时间设置页面

图 8.26 考勤查询页面

图 8.27 请假人员查询页面

2. 办公用品管理

图 8.28 办公用品管理子系统主页面

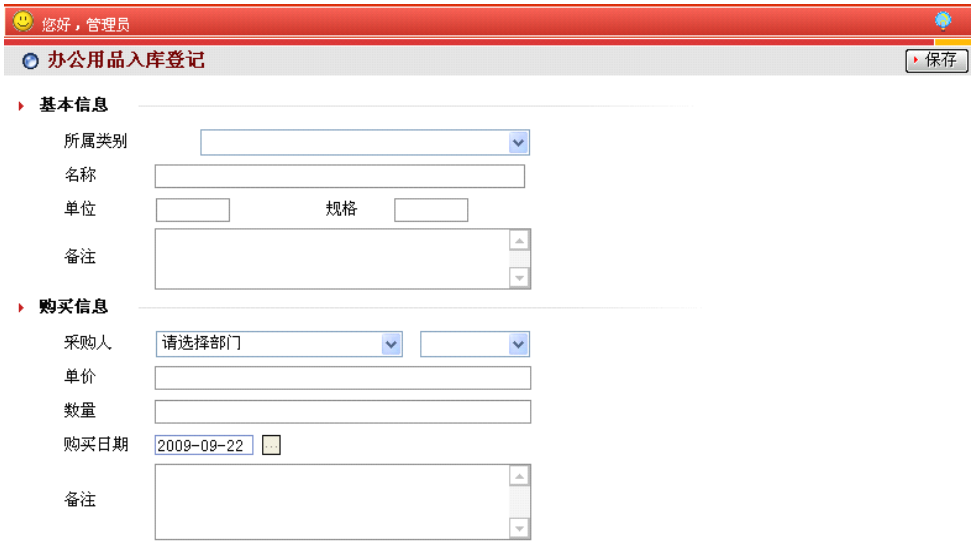


图 8.29 办公用品入库登记页面

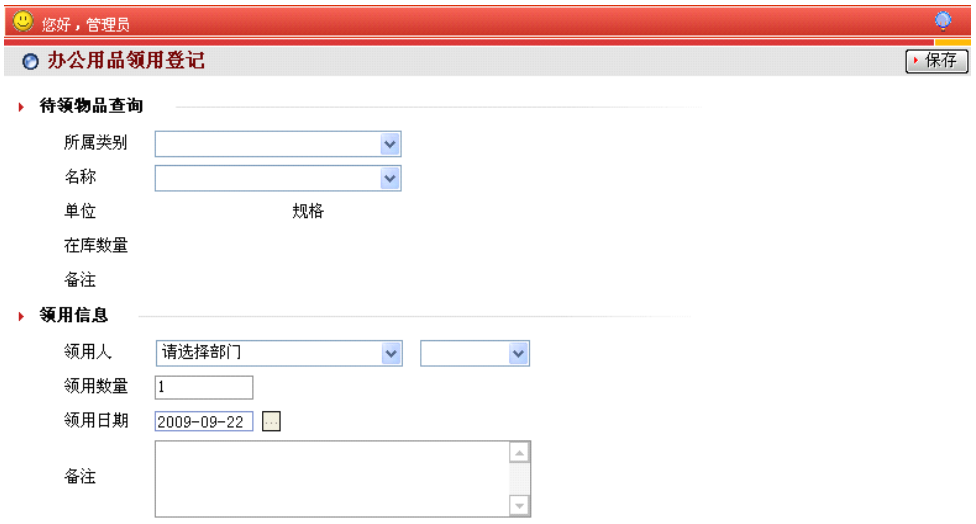


图 8.30 办公用品领用登记页面

图 8.31 办公用品在库查询页面

图 8.32 办公用品领用查询

3. 图书管理子系统

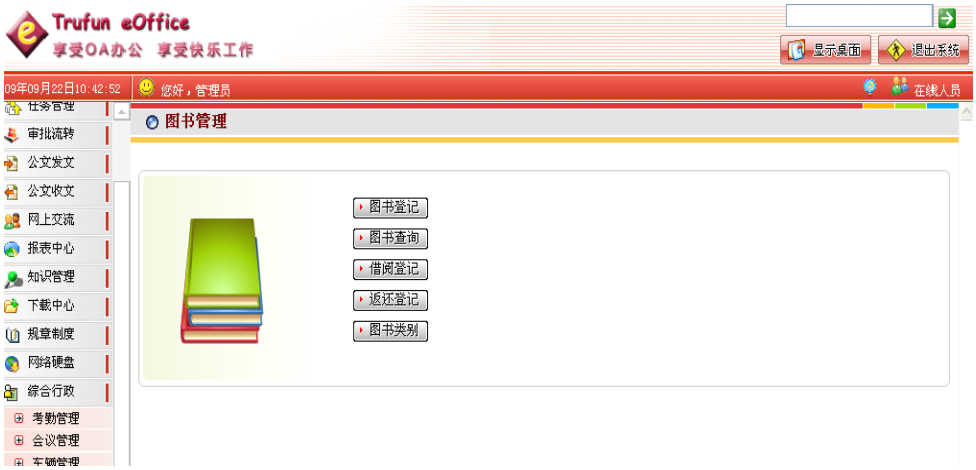


图 8.33 图书管理子系统主页面

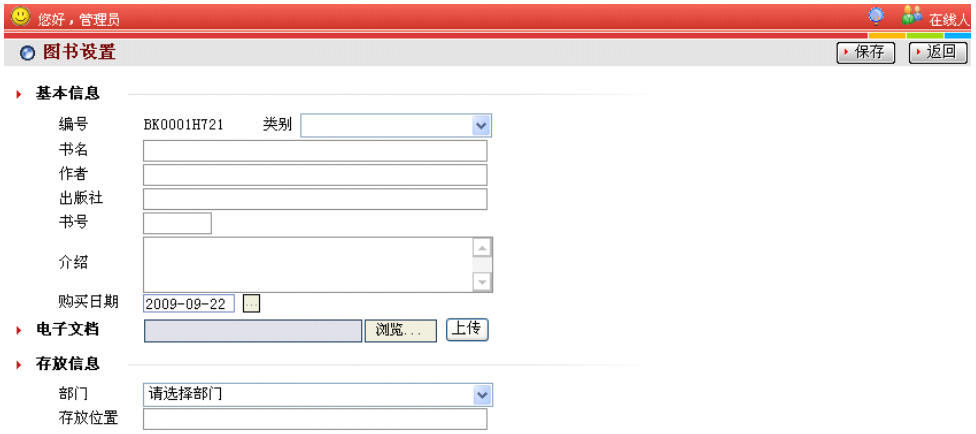


图 8.34 图书登记页面

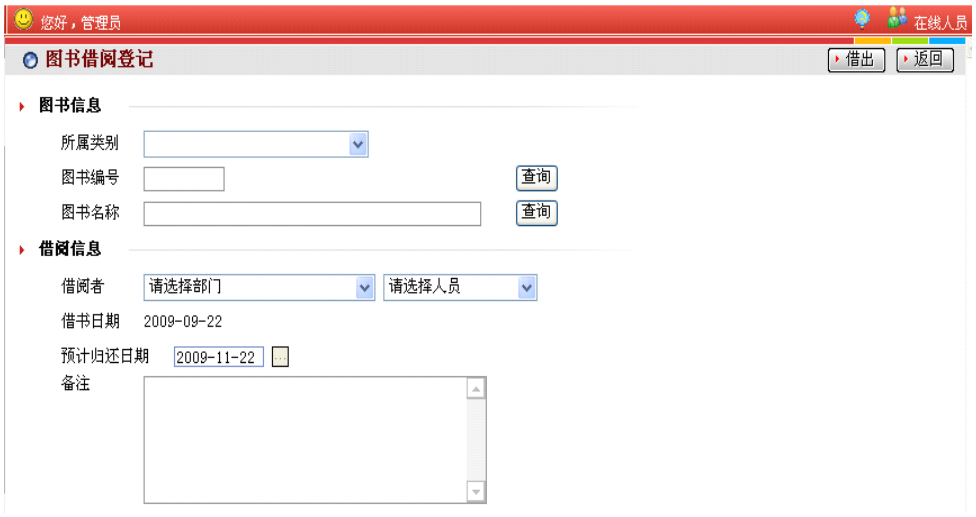


图 8.35 图书借阅登记页面

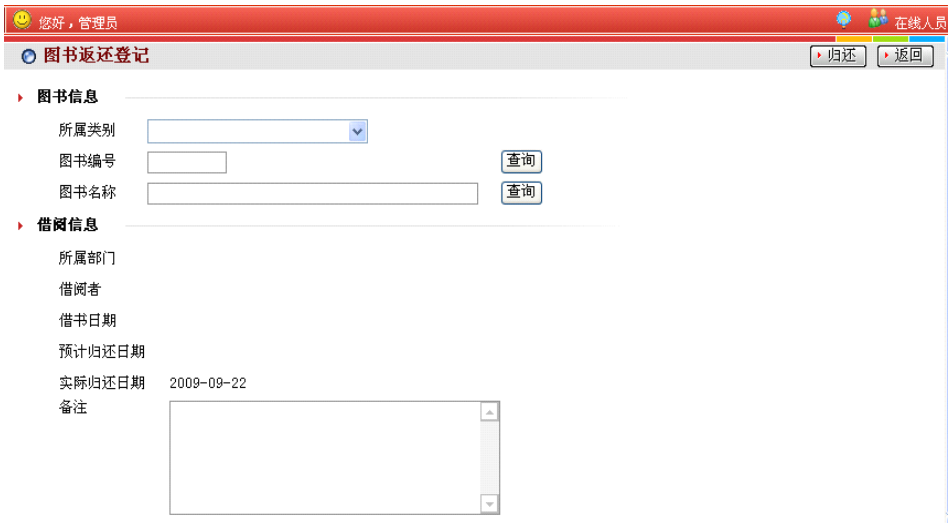


图 8.36 图书归还登记页面

图 8.37 图书查询页面

4. 固定资产管理

图 8.38 固定资产管理子系统主页面

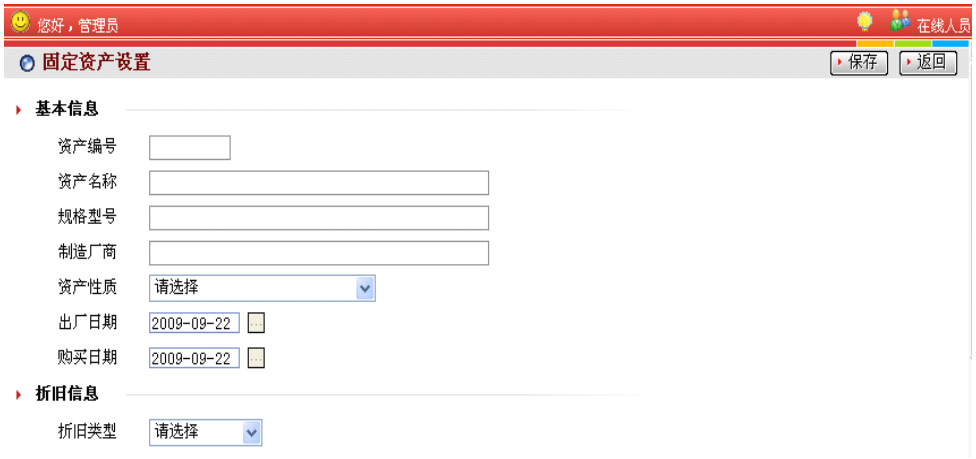


图 8.39 固定资产设置页面

图 8.40 固定资产登记页面

图 8.41 资产性质管理页面

图 8.42 折旧类别管理页面

第二节 顺序图

8.2.1 顺序图的建模元素。

1. 顺序图：显示一组对象为了实现某种功能，而彼此发送和接收的一串消息，这组对象可能是类、接口、构件和节点的具体的或原型化的实例。它是一种以时间为序的表示方法。消息的接收方是通过接口（操作）来接收消息的，消息中的数据是具体的，而不是象类中数据是定义的规则，通过顺序图，可以发现新的操作。顺序图强调消息的事件时间顺序，首先把参与交互的对象或角色放在图上方沿水平轴方向排列，对象的排列顺序从左到右依次为边界对象、控制对象、实体对象。对象的发送和接收消息沿垂直轴方向按时间顺序从上到下放置。

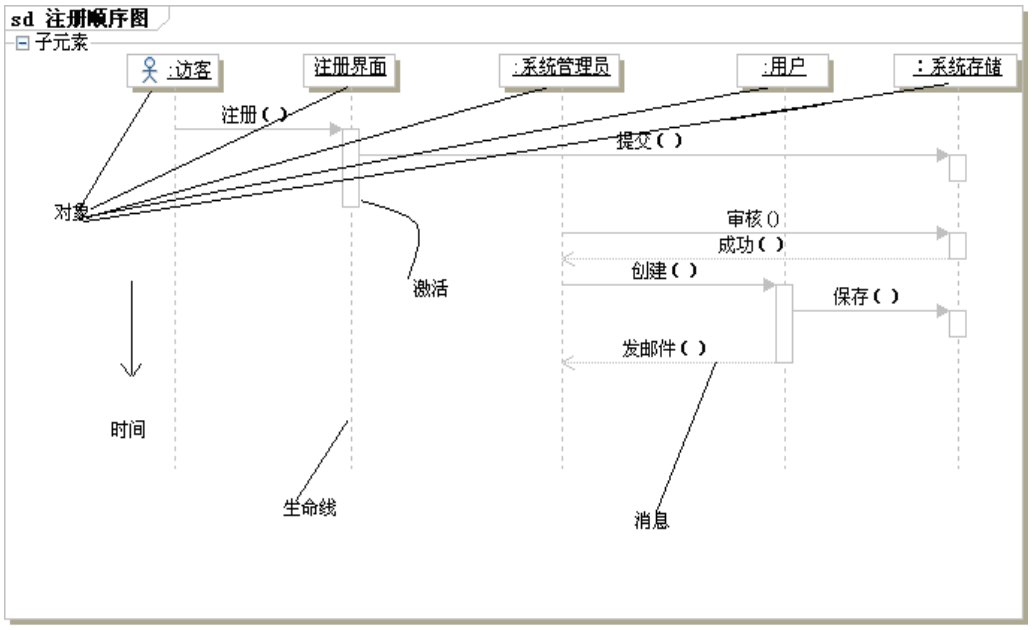


图 8.43 顺序图各元素的布局关系示例

2. 生命线：在面想对象中，行为的执行是对象，而不是类。因此顺序图是对象层次的，而不是类层次，生命线代表交互中的单一参与者，具有类的语义。生命线的图标和类的图标相同，只是多了条虚线尾巴，虚线从交互开始的地方开始，到交互结束的地方结束，其持续时间就是交互时间，但这个时间是相对的，。在分析某些特定交互时，有时可以把生命线看成是类的实例，生命线具有以下属性：

(1) 名称：用于交互的生命线名称，在分析时，常用对象名称（内部参考）代替，放在最前面。

(2) 类型：生命线所代表的类，用“：”号和前边隔开。

(3) 选择器：满足该条件的对象，放在中括号类。

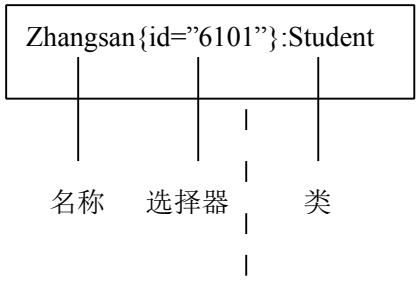


图 8.44 生命线的表示法

3 消息：代表交互中两条生命线之间特定种类的通信，它是对象发出的服务请求，一般消息用一个带实心箭头的实线表示，箭头是消息的接收方，箭尾表示消息的发送方。消息本身由消息名称、参数、返回值，其完整格式为：“返回值：=消息名称（参数：参数类型）：返回值类型"。在生命线上接收的调用消息，生命线代表的类必须有相应的操作。

（1） 同步消息是指发送者等待接收者结束所执行的操作，用实心箭头表示

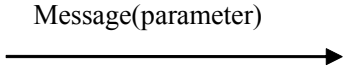


图 8.45 同步消息的表示法

（2） 异步消息是指发送者不等待接收者的返回，异步消息用非实心箭头表示。

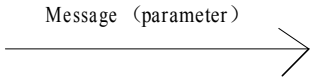


图 8.46 异步消息表示法

（3） 消息返回是指更早的消息接收者返回控制焦点给那个消息的发送者，用虚线的箭头表示。

图 8.47 返回消息表示法

（4） 如果消息的发送者在交互范围之外，又想显示消息的接收，可以用发送消息表示，它的图符是一个箭头尾部带一个实心圆。

图 8.48 发送消息表示法

（5） 如果表示消息由于出错等原因没有到达目的地，可以用丢失消息表示，它的图符是一个箭头头部带一个实心圆。

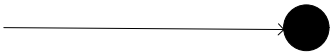


图 8.49 丢失消息表示法

（6） 如果消息的发送方和接收方是同一个对象，这个消息叫反身消息。

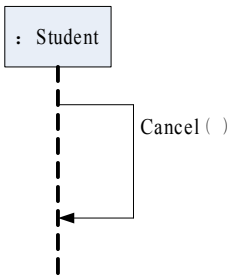


图 8.50 反身消息的表示法

（7） 如果消息附加着时间约束，又被称为定时消息。

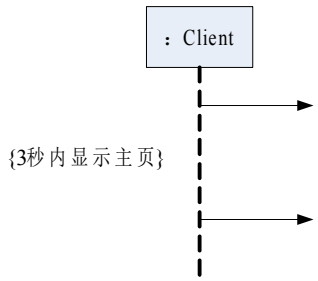


图 8.51 定时消息的表示法

（8） 定向消息是指把消息发个唯一的接收者，而广播消息指把消息发给所有接收者。

3. 对象创建和终止：对象创建和销毁是一种特殊的消息，创建是指由发送者创建并由接收者说明的元素。调用对象的构造函数并实例化，消息上加注<<create>>构造 。

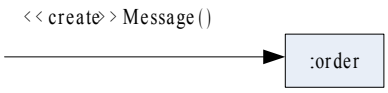


图 8.52 对象创建消息表示法

4. 销毁是指发送者销毁接收者，调用对象的析构函数，消息上加注<<destroy>>构造型，并在生命线上加注一个大“X”。

图 8.53 对象销毁消息

5. 对象激活：当一条消息被传递给一个对象时，就触发了对象的某个行为，这时就说对象被激活了。在生命线上放置长而薄的矩形来表示。矩形表示对象的控制期，表示对象正在执行某个动作。顶部表示行为开始，底部表示行为结束或交回控制权。

图 8.54 激活的表示法

6. 状态常量和约束：对象在收到消息后，状态要发生变化，以反映事件引起状态的变迁，状态常量用放在生命线上的圆角矩形表示，约束放在生命线附近，表示对象实例存在的条件，一般把约束放在大括号内。

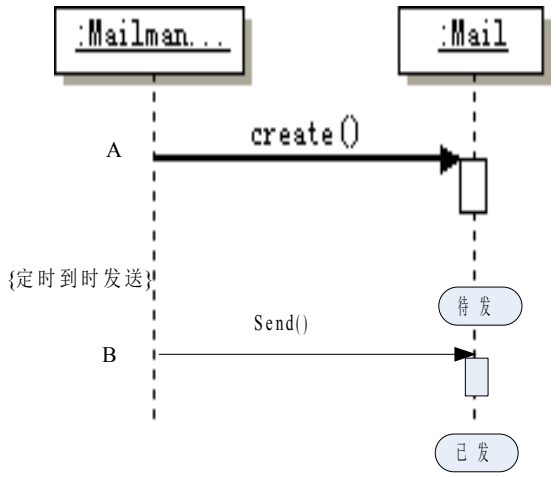


图 8.55 状态常量和约束的表示法

7. 组合区和操作符：为了增强重用，顺序图被划分为区域，该区域被称为组合区或组合片段，组合区有若干运算单元(称为操作数)，监护条件（表示某运算单元是否被执行），并有一个操作符，确定运算单元如何被执行，操作符放在组合区左上角。

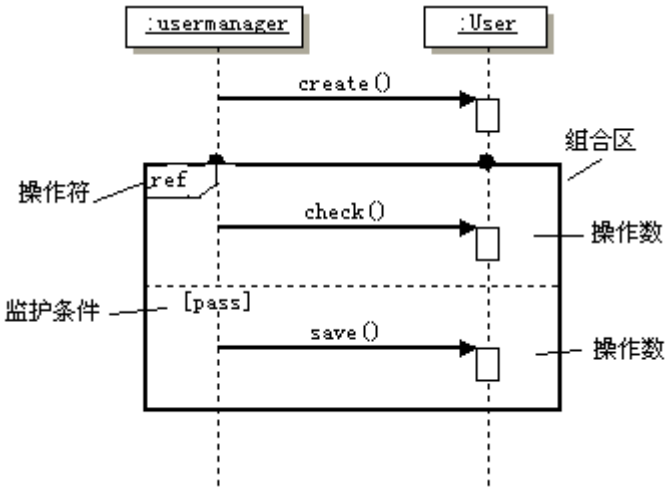


图 8.56 组合区的表示法

8. 控制操作符：表示顺序图上的一个矩形区域，其左上角的小五边形，内部的文字表示控制操作符的类型，操作符作用于穿过它的生命线。
9. 常用的操作符有

（1）opt：可选操作符，监护条件为真执行操作符的主体，监护条件是用方括号括起来的布尔表达式，象语言中 if 语句一样

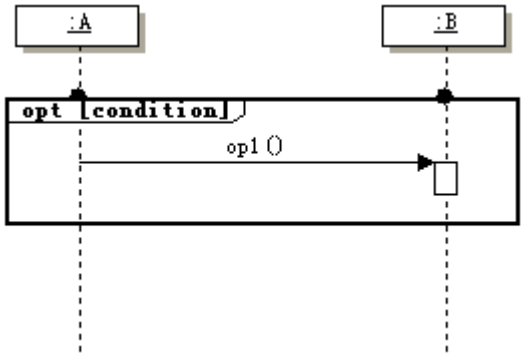


图 8.57 可选操作符的表示法

（2）alt：操作符的主体被分割为几个分区，每一个分区有一个监护条件，如果监护条件为真，就执行这个分区，但只能执行一个分区，所有的条件都为假时，执行一个特殊的分区，就象语言中的 selec…case 语句一样。

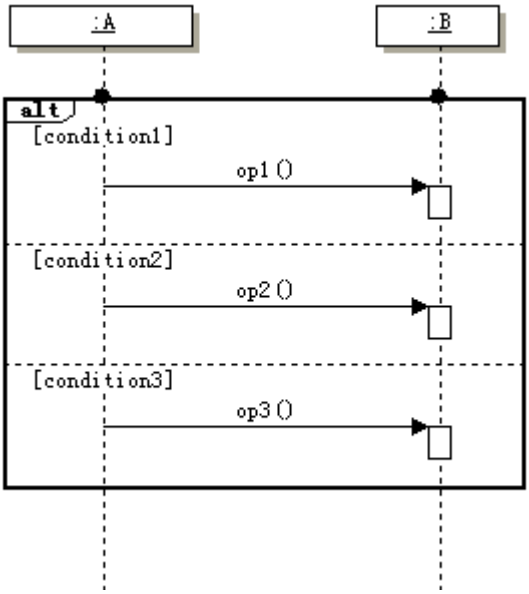


图 8.58 alt 操作符的表示法

（3）loop：循环执行，控制符主体内有一个监护条件，只要在每次循环前监护条件成立，循环主体就被重复执行，就象语言中的循环语句一样。

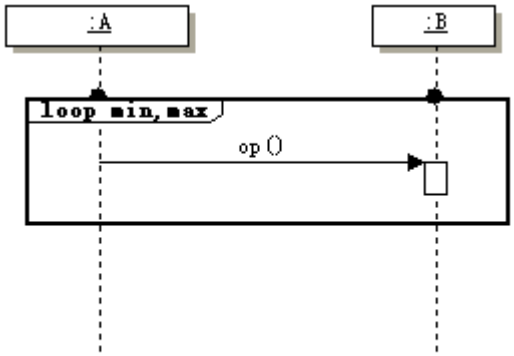


图 8.59 循环操作符的表示法

（4）ref：引用，引用其他交互，如子活动或子行为，其表示法如图 8.20

（5）break：监护为真执行主体，而不是循环的其他部分，而是循环中断以后要执行的内容。

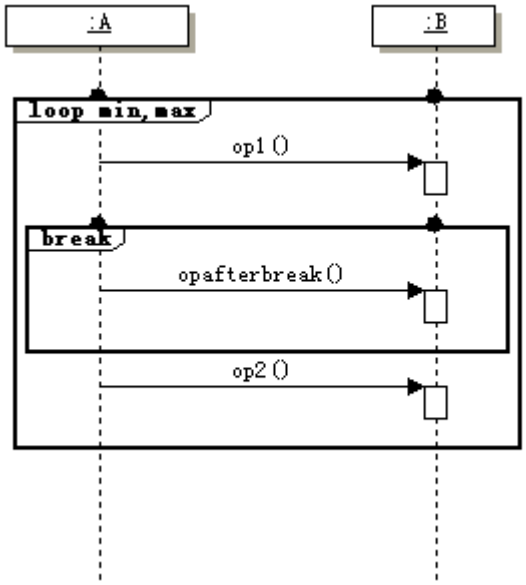


图 8.60 break 操作符的表示法

（6）par：并行执行，操作符的主体被分割为几个分区，每一个分区表示一个并行计算，不同的分区有不同的生命线，进入操作控制符时并发的执行所有分区，各分区内部是顺序执行的。

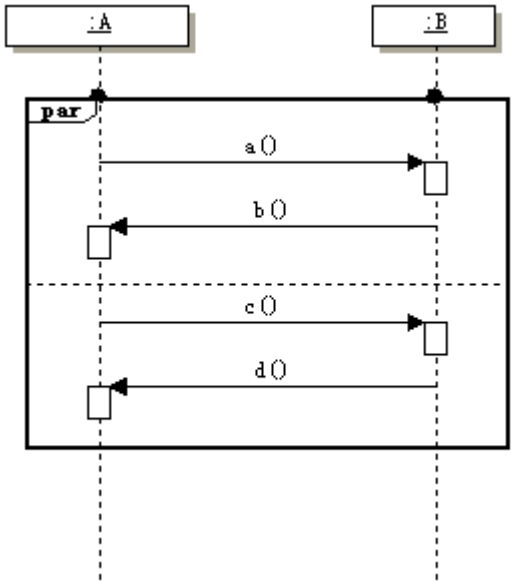


图 8.61 par 操作符的表示法

8. 迭代和条件的语义和表示法和通信图相同，详见通信图的描述。

8.2.2 顺序图的建模分析步骤

(1) 完成用例图的分析。
(2) 对每个用例，识别出参与基本事件流的对象(包括接口、子系统、角色等)。
(3) 识别出这些对象是主动对象还是被动对象。
(4) 识别出这些对象发出的消息是同步消息还是异步消息。
(5) 从主动对象开始向接收对象发消息。

(6) 接收对象再调用自己的服务为主动对象返回结果。
(7) 如果接收对象需要再调用其他对象的服务，需要向其他对象再发消息。
(8) 如此反复，最后返回给主动对象有意义的结果。
(9) 用 UML 建模工具绘出顺序图。
(10) 给顺序图补充必要的说明文档。

8.2.3 车辆管理顺序图的建模分析过程

从前面的理论学习我们知道，顺序图是动态分析模型，是对低级用例的进一步分析，低级用例是未来软件系统的外在行为表现，而顺序图是要反映支持这种外在行为的内部机制，说的更直观一点，就是站在程序内部观察各个对象是如何相互通信来实现这个低级用例的。这个过程和写详细设计相类似，需要逐个用例来写（类似于逐个模块来写详细设计）。

建模开始的第一步是寻找要建模的低级用例，在车辆管理系统，有很多低级用例，原则上，每个低级用例都要进行顺序图建模，这样才能使最后的代码基本可控，由于篇幅所限，我们选择车辆管理中的用车申请作为例子，说明顺序图建模的整个分析过程。

用车申请的操作者，也就是低级用例的角色，和前面进行业务用例分析的角色一致，它是提出用车申请这个行为的使动者，从程序内部来看，是它向软件系统发出了一个主动事件。接着我们要确定哪些对象通过消息交互，来响应了这个事件。健壮性建模就是用来帮助我们确定这些对象的，我们知道在软件系统中的类根据其功能不同被划分为三个类（对象），即边界类、控制类和实体类。用车申请肯定有输入界面，否则申请明细无法输入到系统，至于是 windows 界面还是 page 界面，现在不用考虑(留到实现阶段考虑)，因此用车申请界面类是存在的。其次是寻找控制对象，如果业务很简单，不考虑未来的扩展性，可以考虑不设计控制对象，尽管有很多人这样做，但我们不提倡这样做。我们设计一个类“用车申请处理”类，来完成对界面输入的信息的接收、校验以及完成协调其他对象之间的关系。最后是找到参与此用例的实体对象，怎样找实体对象呢？首先我们从数据存储上寻找，这是因为有一部分实体对象是要永久保存的，提交用车申请后，车辆管理员要在网上进行用车审批（而不是把用车申请打印出来在纸上审批），所以存在一个用车申请实体和用车申请存储实体，在提交用车申请时，还要查看车辆的使用情况，所以车辆的使用也是一个永久保存的实体，当然很多低级用例还存在着其他实体，如权限实体、日志实体等，由于该系统没有这方面需求，所以不作考虑，另外还有一些如数据库连接实体这样的公共对象，一般不画在业务用例的顺序图中，而是画在如系统启动这样的低级用例中。前面所找的这些对象除了用车申请的操作者(一般在系统内都有代理对象)对象外都是被动对象，对象间的消息存在着先后逻辑顺序，所以是同步消息。

用车申请的操作者首先发出查询车辆使用的消息，消息中可以附加查询条件等参数，该消息首先被用车申请界面对象接收，该界面对象首先查看自己的方法，能否响应这个请求（如果查询方法写在界面类中，就可以响应请求），如果能响应请求，就直接在界面给出响应结果，如果不能响应，界面对象就在包内或系统查找那个对象能够响应这个消息，界面对象就把该请求发给这个对象，请注意在这里，界面对象是整个消息控制中心。这种做法虽然很传统，但开发简单。（后面我们将采用另外一种做法：即把界面对象只作为输入的窗口，响应使动者请求的这个任务交给专门的控制对象，这个控制对象是这个用例的整个消息控制中心）。“车辆使用”存储对象中封装了和消息同名的方法，界面对象把查询车辆的使用信息的这个消息发到这个对象。车辆使用存储对象接到查询车辆使用这个消息，同样它首先查看自己能否直接响应这个消息，如果不能，还需要其他对象配合，“车辆使用”

存储对象就暂不发响应消息，而向其他对象发出请求，待所有条件都具备后，“车辆使用”存储对象即向界面发出返回消息，界面对象就显示查询结果。

用车申请人看到界面对象返回的有空车的消息后，向用车申请界面对象发出起草用车申请这个消息，界面对象不处理这个请求，把这个请求发给专门用来处理用车申请的“用车申请处理“控制对象，用车申请控制对象根据界面对象传递的参数，调用用车申请对象的构造方法，生成一个用车申请实体，用车申请这个实体本身也不能完成对用车申请信息的响应，通过向用车申请存储对象发送保存消息，保存成功后（只画基本消息），发出返回消息，用车申请实体收到这个消息后把这个消息发给用车申请处理控制对象，申请处理控制对象再在界面对象上显示返回信息，这样，用车申请人就可以看到此信息。最后，用车申请处理控制对象向用车申请对象发出销毁消息，即可销毁刚产生的用车申请对象。整个低级用例即告完成。

8.2.4 顺序图的建模案例

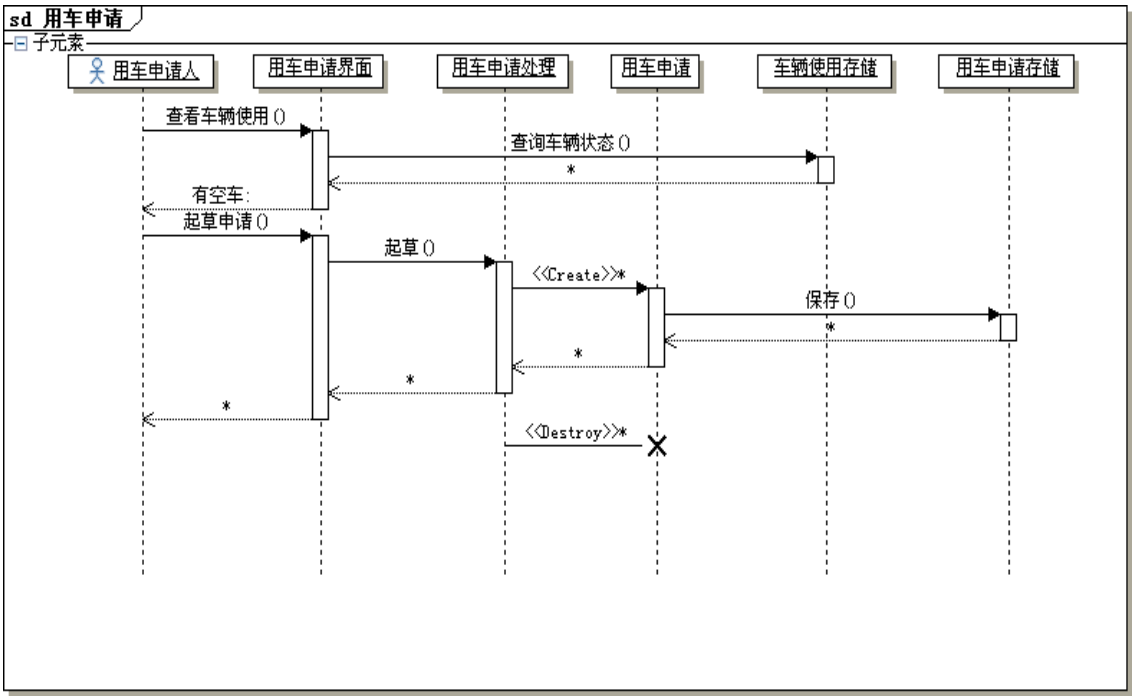


图 8.62 车辆管理顺序图建模

8.2.5 操作步骤

对于操作步骤的描述，为了避免案例单一，从下面开始我们将用车辆管理系统进行描述。

1. 在第四章第四节项目和包建模的基础上进行顺序图建模，如果要直接建顺序图模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“车辆

管理”前面的“+”符号。

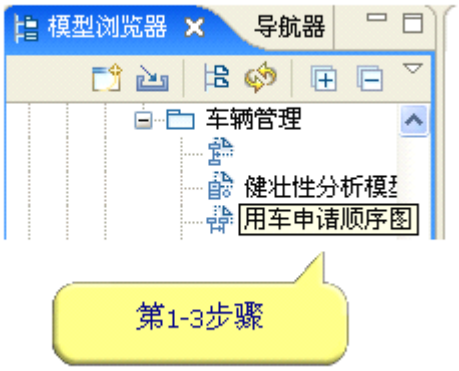


图 8.63 顺序图建模操作步骤 1-3

4. 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在"新建框图“菜单下选择”序列图“。则在该包下增加了一个节点，一个是以”sequencediagram“开头的节点，代表这个顺序图的全局属性，并可在“常规”选项卡中修改该节点名称为“车辆管理顺序图"。并通过调整绘图区四周的锚点，调整绘图区大小。

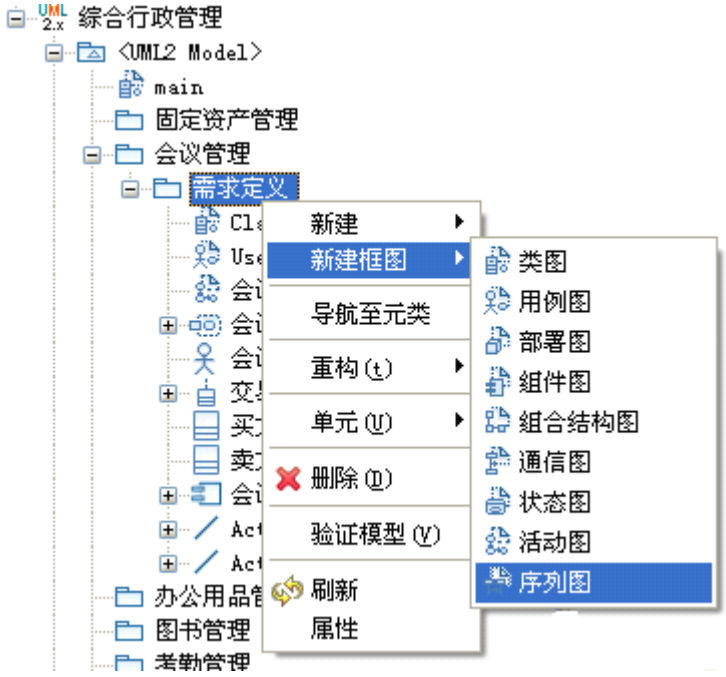
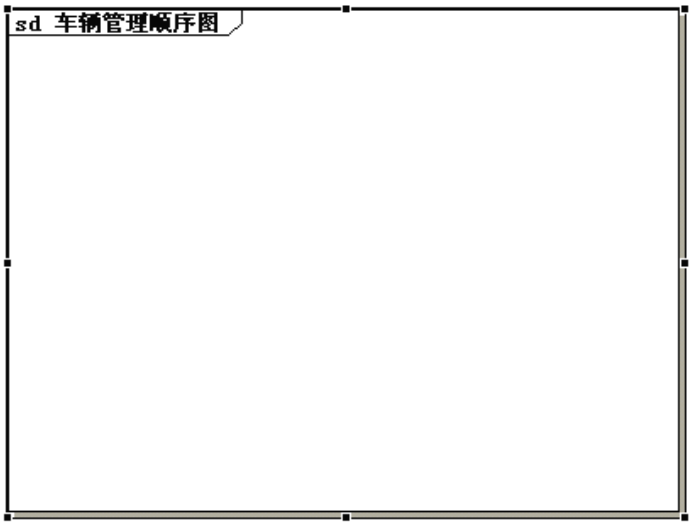


图 8.64 序列图操作导航



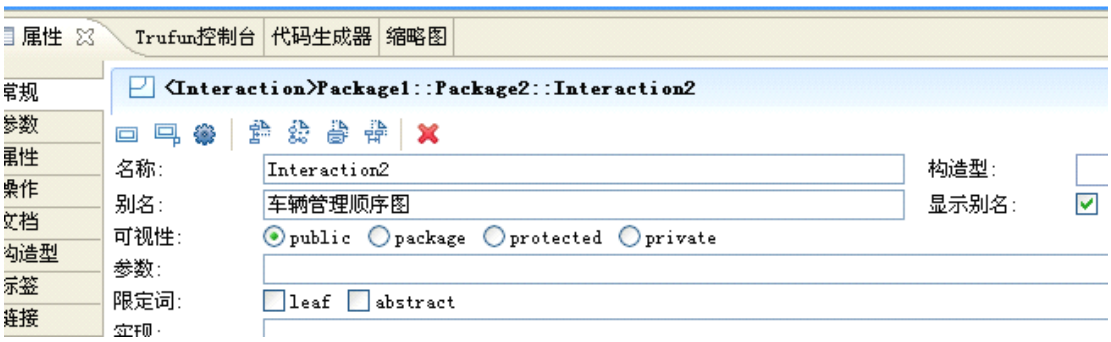


图 8.65 顺序图的属性设置

5. 首先从绘图工具面板（或从导航树选中以前建立的对象）选中生命线元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“用车申请人”。在名称栏目输入“applicate”，以完成对象（生命线）的属性设置。同理可完成对象（生命线）：用车申请界面对象、用车申请处理对象、用车申请对象、车辆存储对象的布局。

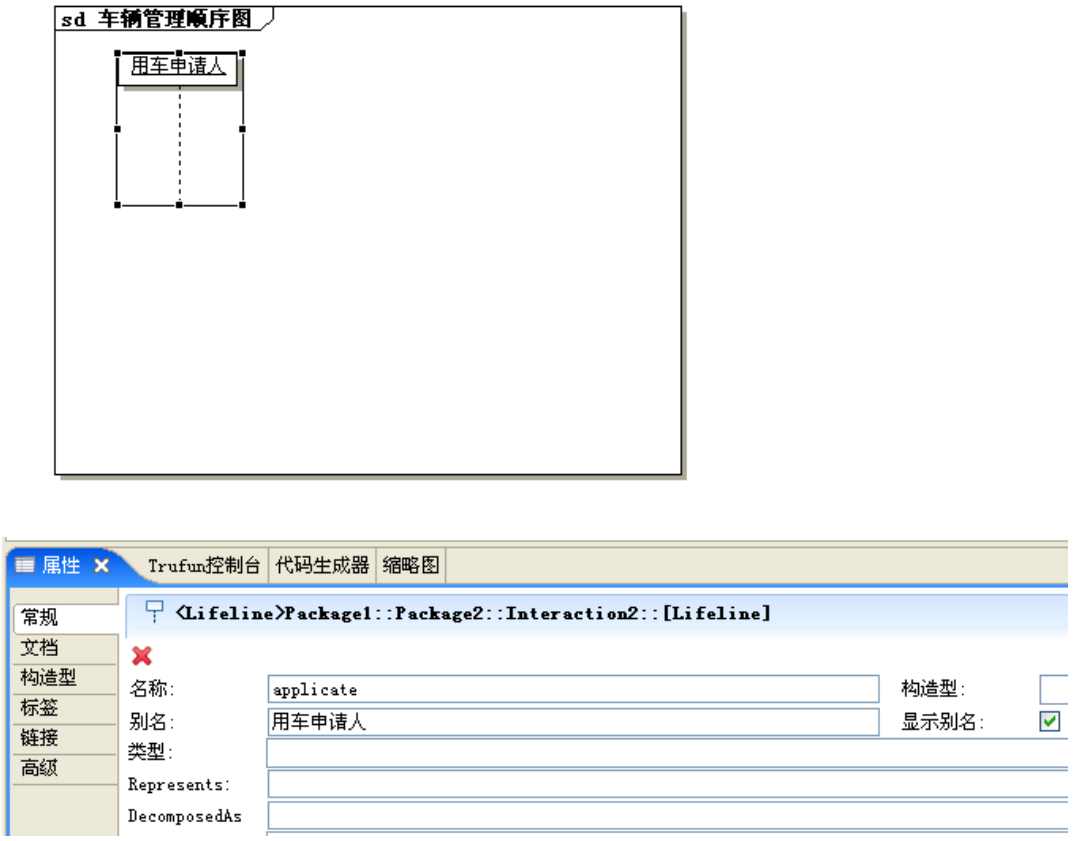


图 8.66 生命线属性设置

6. 从绘图面板选中引用组合片段元素，拖到绘图区，经过位置调整、命名，和生命线连结以后，画出内部交互，反映用车申请人查看车辆使用情况。

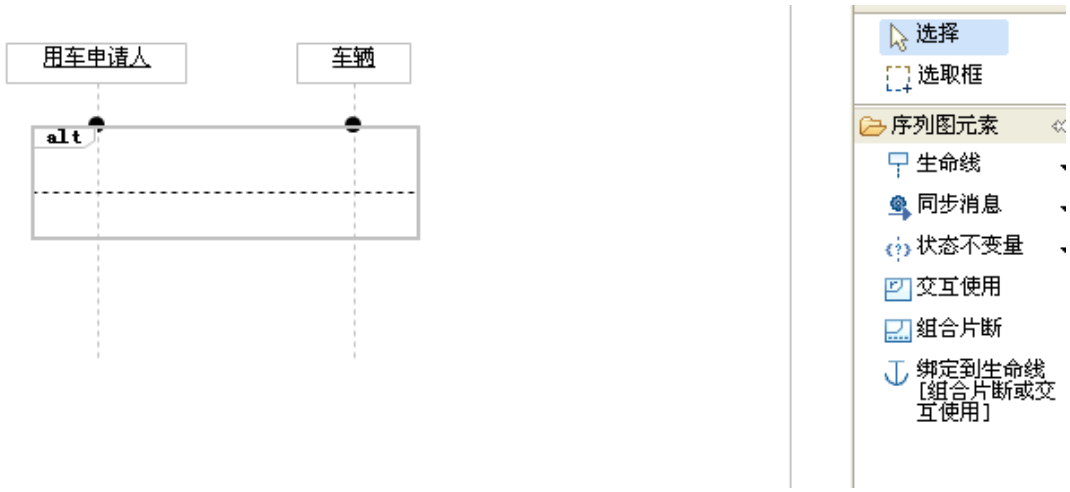


图 8.67 组合区的使用操作

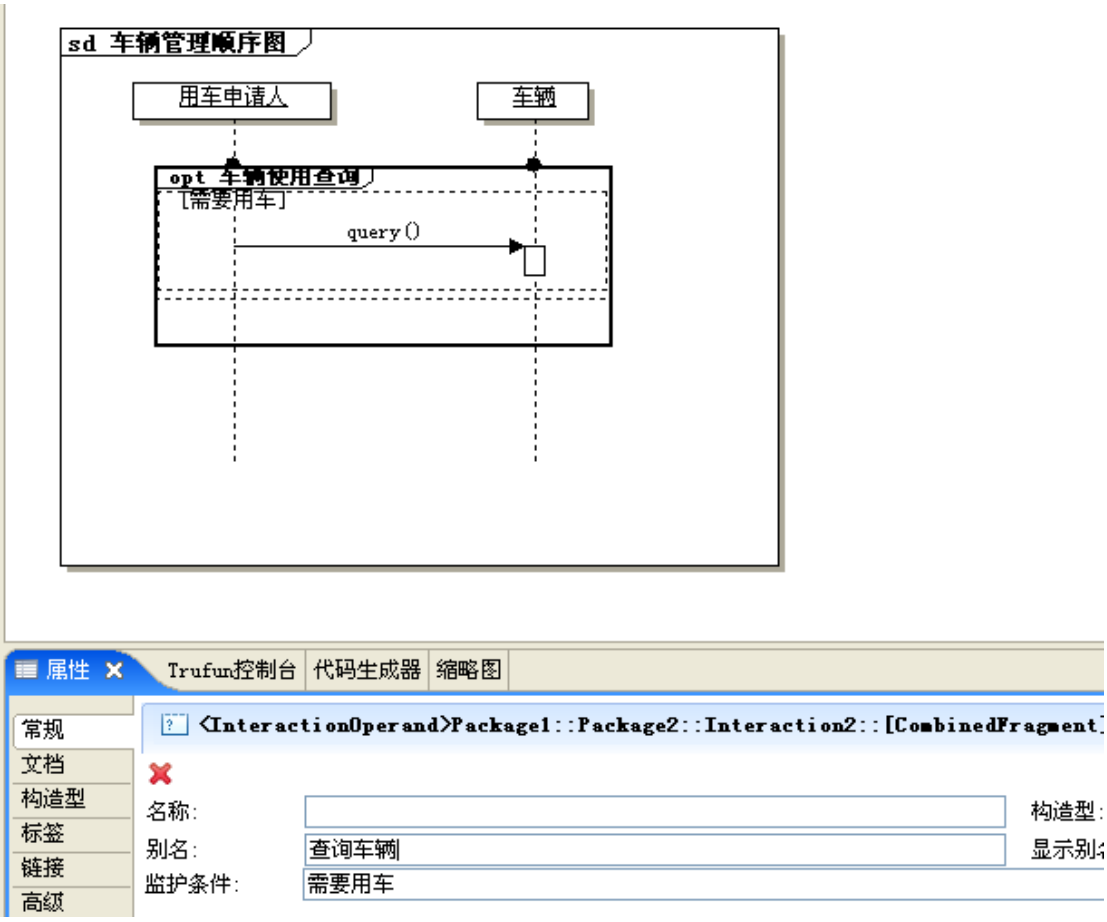


图 8.68 组合区的属性设置

7. 从绘图面板选中消息元素，拖到绘图区，连接“用车申请人”主动对象和“用车申请界面”对象，在该消息选中的情况下，在下面的“常规”选项卡的别名栏输入“起草申请（）”，在名称栏输入“draft()”，如果要显示返回消息，则选中下面的复选框，该反映“用车申请人”对象向”用车申请界面“对象发出了起草用车申请的服务请求；”用车申请界面“对象收到服务请求后，把这个消息发给”用车申请处理“对象，同样方法，可以完成该消息的布局。
8. 跟踪消息的处理过程，从上到下，完成所有消息的建模。

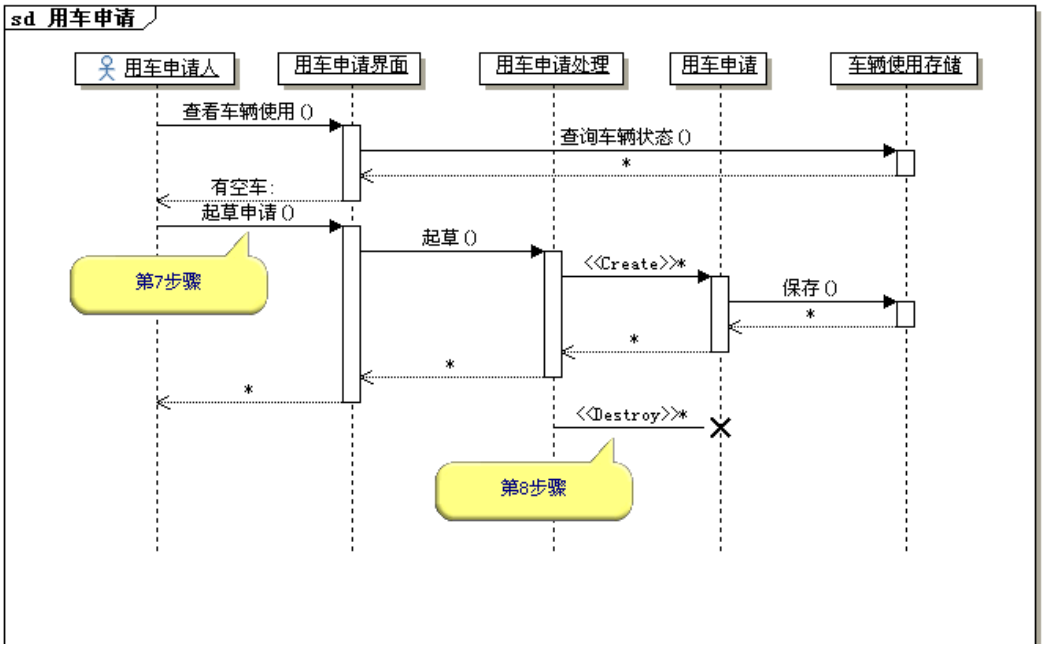


图 6.69 顺序图建模步骤 7-8

8.2.6 会议管理中会议申请顺序图

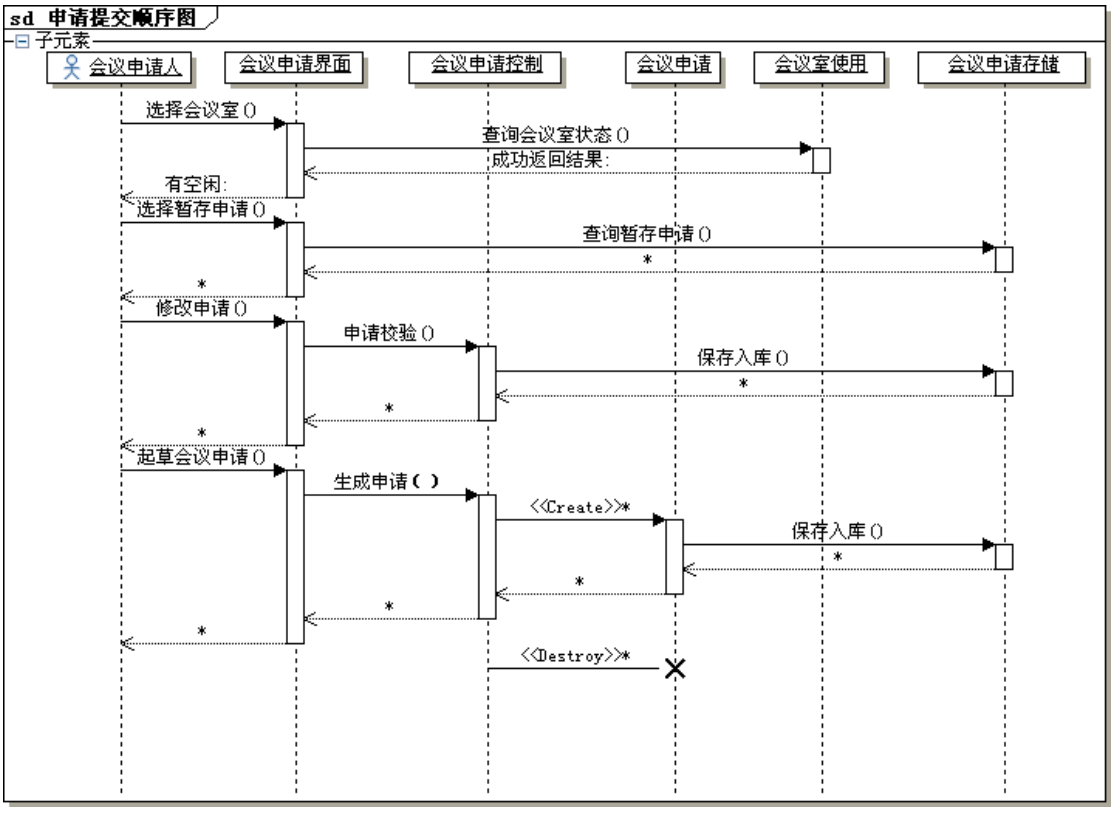


图 8.70 会议管理顺序图建模

第三节 通信图

8.3.1 通信图的建模元素

1. 通信图：在 UML2.0 中，通信图实际上是以前版本的协作图，使用通信图重点是把消息和对象之间的链直观的布局展示出来，它从空间角度反映对象之间的组织关系。通信图侧重对象之间的交互、对象的结构，有助于验证类间的关联。它同样可以表示消息的类型，如同步消息、异步消息、返回消息、丢失消息、发现消息以及对象的创建消息，但其表示方法和顺序图截然不同。

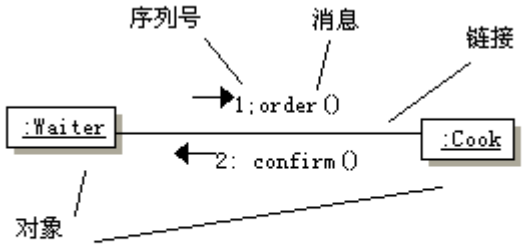


图 8.71 通信图的表示示例

2. 对象或对象组：和顺序图中的对象含义和表示一样，是消息的发起点和目的地。
3. 链接：对象之间的通信路径，是类间关联的实例。反映对象之间的交互，用直线表示。如果两个对象之间有一个链接，可以不用名称（链接可以使用关联名称或角色名称）。基于关联的链是永久的，基于依赖为完成某一操作而建立的链是暂时的。
4. 顺序号：又叫序列号，通信图的消息有一个顺序号作为引导，该顺序号是一个表达式，表示消息的执行顺序，它的内部结构使用“.”点号隔开，序号最后是一个”:”冒号,通过点号的层次关系，可以反映消息的执行顺序和并行。
5. 迭代表达式：当消息需要重复执行时，在顺序号中附加迭代的信息，表示方法是在分号之前添加一个“*{迭代子句}“，迭代子句说明迭代次数，一般用自然语言或伪代码表示。如果迭代是针对对象集，在链接上必须附加多重性和角色名称，并在消息前面附件迭代的标志”*“星号。

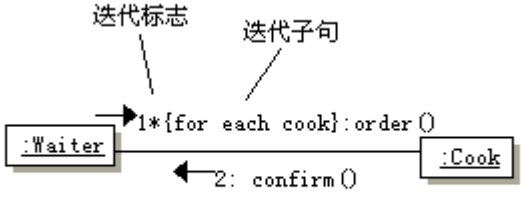


图 8.72 迭代元素的表示法

6. 并行表达式，如果迭代中的消息是并行执行，在顺序号中还要附加并行符号，表示方法是在分号之前添加一个“*||{迭代子句}“。

图 8.73 并行元素表示法示例

7. 条件表达式：它是用来表示某个消息是否被执行，只有当该监护条件为真时，消息才被执行。在顺序号中附加监护条件信息，表示方法是在分号之前添加一个“【条件子句】“。但一般只添加一些简单条件。

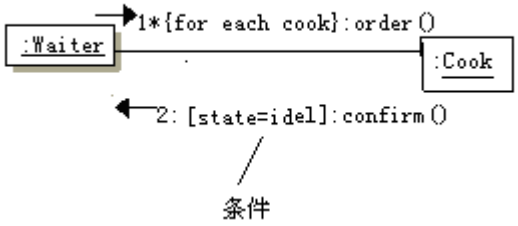


图 8.74 条件元素的表示法

8. 主动对象：是指具有控制区的对象，而被动对象只会对要求做出应对。
9. 多重对象：如果多个对象执行相同的操作，叫做多重对象。

8.3.2 通信图的建模分析步骤

1. 完成用例图的分析
2. 对每个用例，识别出参与基本事件流的对象(包括接口、子系统、角色等)
3. 识别出对象间的连接关系
4. 识别出这些对象发出的消息顺序
5. 从主动对象开始向接收对象发消息
6. 接收对象再调用自己的服务为主动对象返回结果
7. 如果接收对象需要再调用其他对象的服务，需要向其他对象再发消息
8. 如此反复，最后返回给主动对象有意义的结果
9. 用 UML 建模工具绘出通信图

10. 给通信图补充必要的说明文档

8.3.3 车辆管理通信图的建模分析过程

通信图的建模分析过程和顺序图相同，不再赘述。

8.3.4 通信图的建模案例

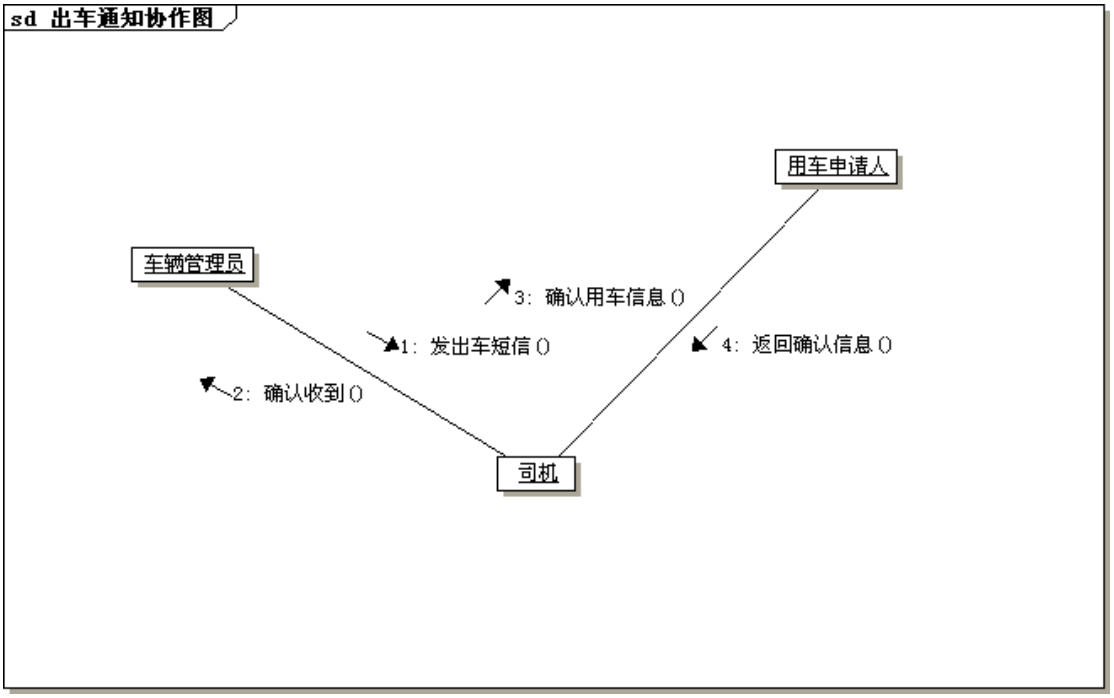


图 8.75 车辆管理通信图建模

8.3.5 操作步骤

1. 在第四章第四节项目和包建模的基础上进行顺序图建模，如果要直接建顺序图模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“车辆管理”前面的“+”符号。
4. 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在"新建框图“菜单下选择”通信图“。则在该包下增加了一个节点，一个是以”communicationdiagram“开头的节点，代表这个通信图的全局属性，并可在“常规"选项卡中修改该节点名称为“车辆管理通信图"。通过调整绘图区四周的锚点，调整绘图区大小。

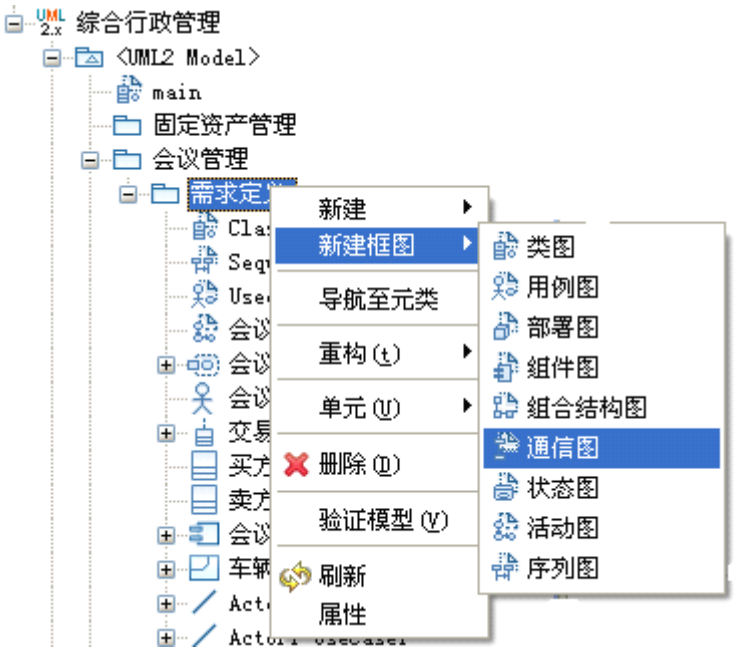


图 8.76 通信图操作导航菜单

5. 首先从绘图工具面板（或从导航树选中以前建立的对象）选中生命线元素，将其拖入绘图区，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“用车申请人"。在名称栏目输入“applicate"，以完成对象（生命线）的属性设置。同理可完成对象（生命线）：车辆管理员对象、司机对象的布局。
6. 从绘图工具面板选中消息元素，拖到绘图区，连接“车辆管理员”主动对象和“司机”对象，连线上面在该消息选中的情况下，在下面的“常规”选项卡的签名名栏输入“发短信通知（）"，在名称栏输入“sendmsg()”，如果要显示返回消息，再选中绘图工具面板的消息元素，只要反向拖动连接，即可出现返回消息的图标和序号，同样办法输入返回消息的名称。该反映“车辆管理员”对象向”司机“对象发出了出车短信的服务请求；”司机“对象收到服务请求后，把确认消息发给”车辆管理员“对象，同样方法，可以完成该消息的布局。
7. 跟踪消息的处理过程，完成所有消息的通信图建模。

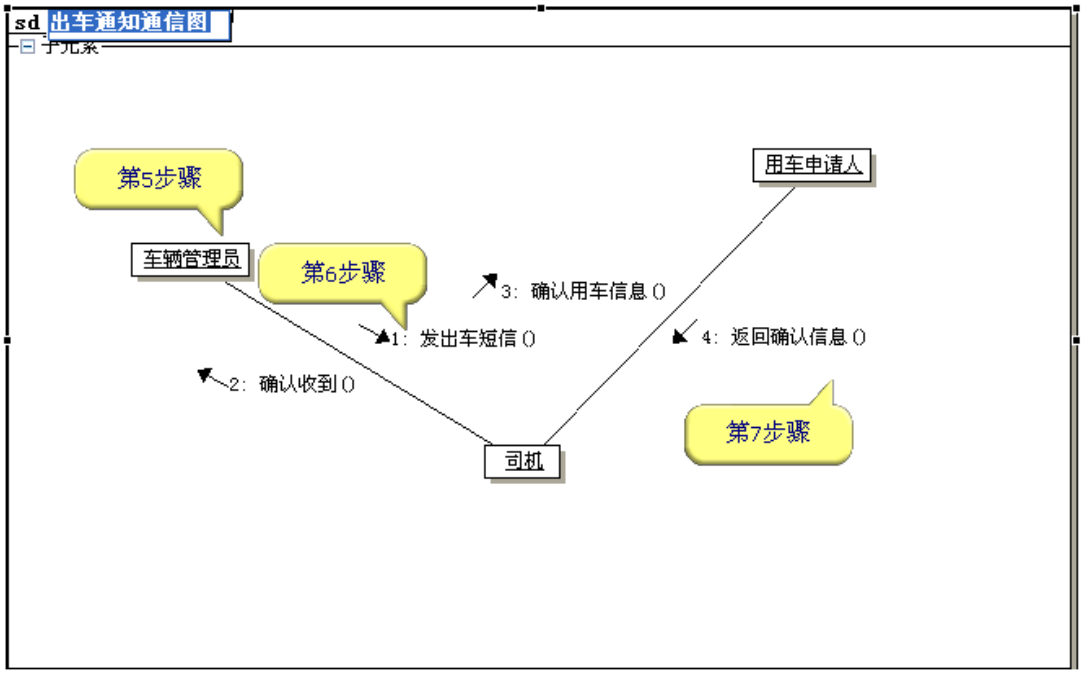


图 8.77 通信图的操作界面

8.3.6 会议通知通信图

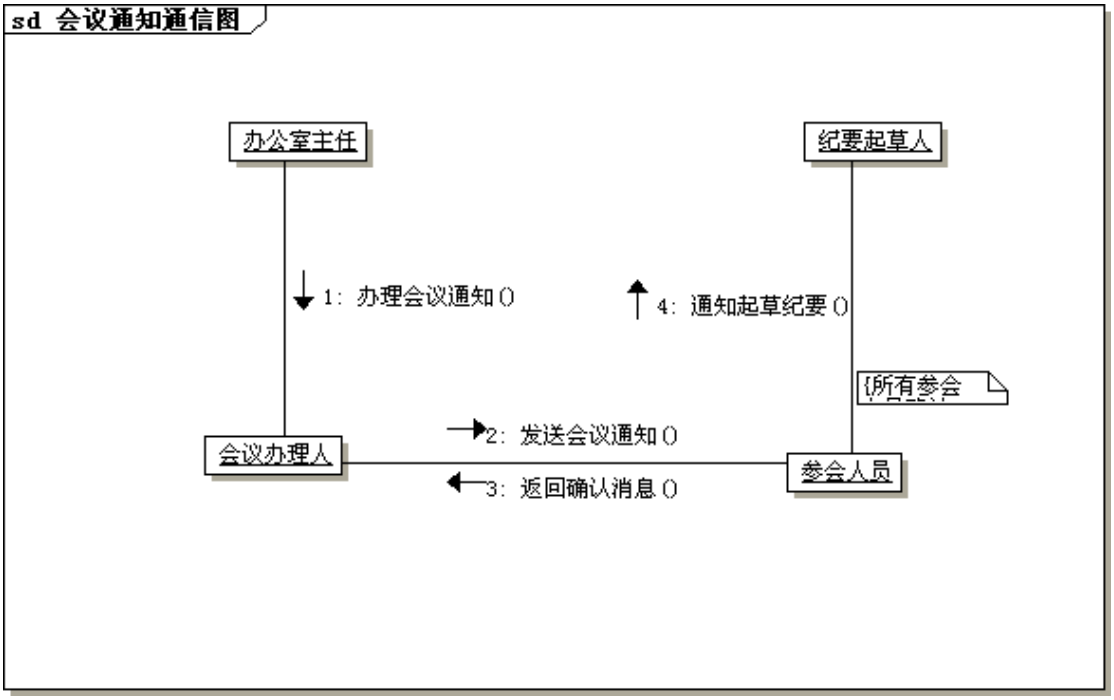


图 8.78 会议管理通信图建模

第四节 顺序图和通信图的区别

顺序图和通信图是由于描述对象交互的关注点不同而引起的，如果关心对象间的关系（上下文关系），就用通信图，如果关心对象之间的顺序、时间选项，就用顺序图。它们之间既有联系，又有区别。

相同点：

1. 它们都表现出了对象之间的交互信息。
2. 两个图对象的绘制方式相同。

不同点：

1. 顺序图反映了对象之间交互的时间关系，而通信图反映了对象之间交互的空间关系。
2. 顺序图用于展示特定的业务场景，而通信图用来展示详细的业务过程。
3. 顺序图的对象在图形的顶部一字排开，而通信图对象的摆放位置在二维空间只要选择合适的位置即可。
4. 通信图不能表现组合片段。

第五节 小结

本章主要对对象间交互，从不同角度进行了讨论，进一步深化了对用例图、类图的认识，消息是对象交互的灵魂，本章从消息的来源、分类、表示等方面，都作了较多的介绍；理清对象、对象组、类、对象组之间的关系，对于知识的承上启下，准确地把握建模元素的语义，具有十分重要的意义；组合片段、迭代子句、条件子句是本章出现的全新建模元素，能否很好地运用这些元素，反映了一个分析人员分析的详细程度。在实际工作中，并不是每个系统操作都要进行交互图建模，通常只选择一些有代表性的重要业务进行分析。

第六节 习题

1. 常见的消息分类有哪些？它们是从那个角度进行分类的？
2. 试编写会议管理中的会议申请顺序图的建模的操作步骤。
3. 试找出图书管理案例中重点系统操作，并画出顺序图。
4. 试找出办公用品管理中的重点系统操作，并画出通信图。
5. 有一种模型叫交互概览图，它是把活动图和顺序图结合在一起画，所不同的是活动图的活动节点是顺序图的组合片段（无内部交互），试把会议管理中的会议申请顺序图转化为交互概览图。
6. 试对图书管理系统的图书借阅进行健壮性分析。

第八章 对象行为

上一章我们主要讨论了一组对象间的交互行为，如果是一个对象，我们怎样来描述该对象呢？回答是状态图，状态图是反映在对象的生命周期内，对象的动态行为。人为什么变老，机器为什么会报废，都是因为状态发生了变化。在这里，每一个对象都是一个孤立实体，能够接受事件（外部信号或能够影响对象的任何东西）并对事件作出回应。与事件相关的另外一个元素是状态，状态是给定类的对象的属性集合，这个属性集合对所发生的事件具有相同性质的反应，即状态相同、事件相同则反应相同。状态机描述了类的行为，而状态代表了类的对象执行了一步。

第一节 状态图

9.1.1 状态图的建模元素

1. 状态机：在类层次反映状态与状态转化的图，它是一个类的对象的所有可能的生命历程的模型。主要用来捕捉外部事件引起的变化，它将一个对象与其外部世界隔离开来独立考察其行为。不宜用来描述系统的整体运作（如有此要求，可用顺序图）。状态机用来描述界面和控制类业务比较合适。
2. 状态图：描述交互对对象内部的影响，交互图中的消息在这里变成外部事件对对象发出的命令，对象对这些命令的响应导致对象的状态发生变化。因此，从这个意义上说，状态图是顺序图的进一步细化，并且是对核心对象（选择核心对象的依据是看是否在多个交互图中有多个消息指向该对象）的细化。
3. 状态：每一个对象都至少有一个状态，状态是在此之前执行所有活动的结果，通常用一组属性值来表示，并且可能与其他对象相关。有时类用一个属性来表示状态。状态用圆角矩形来表示。起始状态和结束状态是两种特殊的状态，起始状态用一个填充的实心圆来表示，状态图必须有一个起始状态，终止状态用一个圆圈内加一个实心圆，终止状态可以访问，但不能改变。扩展的状态图标分俩栏，上面是名称分栏，下面为内部转换分栏。

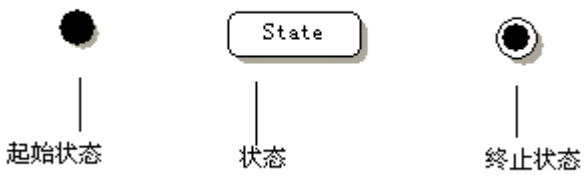


图 9.1 起始状态、终止状态和简单状态的表示法

图 9.1 扩展状态图标的表示示例

4. 事件：事件是指发生某件事，可以导致产生某一动作，是激活状态转换的触发器，一次只能触发一个事件，引起一次状态转换。根据所开发的系统不同，事件的触发分为以下四种情形：

（1） 改变事件：是指事件的触发依赖于某个条件表达式，条件表达式为真时，事件触发。

（2） 信号事件：两个对象之间通过信号进行通信，信号的接收是信号接收对象的一个事件。这类事件有时被称为消息。

（3） 调用事件：指一个对象对操作调用的接收，与普通方法不同，调用事件触发的状态转换可以继续它的执行过程，与调用者处于并行状态。

（4）时间事件：代表时间的流逝，它由一些特定对象的信号引起。

事件用一个箭头上的标签来表示。箭尾表示收到消息时所处的状态（非事件来源），箭头表示收到消息后迁移的状态。对象对事件的应对取决于事件和状态的共同作用。事件的执行顺序为：

（1）如果当前状态下某个活动正在执行，妥善地终止该活动。

（2）执行出口动作。

（3）执行和引起转换事件相关的动作。

（4）执行新状态的入口动作。

（5）开始新状态下的活动。

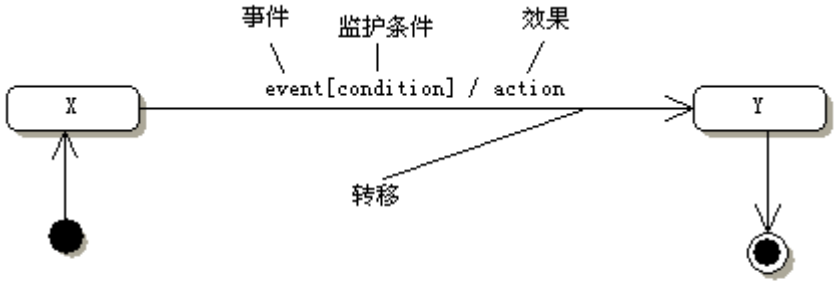


图 9.2 状态图的转移、事件、监护条件及效果的表示法

5. 转换：从一个状态出发的转换定义了处于此状态的对象对事件发生所作出的反应。内部转换只有源状态，没有目标状态，也就是说事件，对象对此作出了响应，但没有发生状态改变。转换包括引起转换的事件触发器，监护条件、效果和转换目标。转换分为以下几种类型：

（1） 入口转换：进入某一状态时执行的入口活动（由进入状态的外部转换引起）。

（2） 出口转换：离开某一状态时执行的出口活动（由离开状态的外部转换引起）。

（3） 外部转换：对事件作出响应引起状态的变化。

（4） 内部转换：对事件作出响应引起一个特定效果。

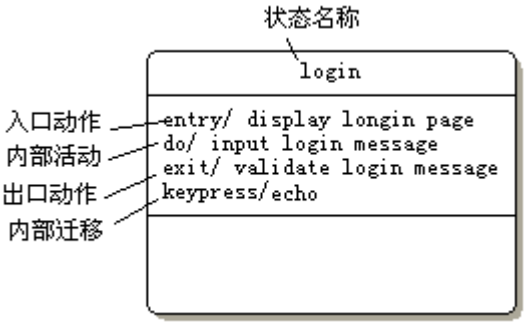


图 9.3 状态所包括的动作和活动

6. 监护条件：指转换的条件，当监护条件为真，转换引发。它只能在事件发生时被计算一次。监护条件也可以由某个任务完成来触发。
7. 效果：有时又被称为动作，可以是一个动作，也可以几个动作，动作和动作之间用逗号隔开，由动作构成的表达式和事件之间用“/”隔开，转换被触发后，它对应的效果会被执行，且不可被中断。
8. 入口动作和出口动作：当效果被执行完后，转换的目标状态被激活，触发入口动作或出口动作。如果一个动作在所有使对象迁移到某状态的事件中都用到，就为入口动作，入口动作的关键词为“entry”，后面加一个斜杠，出口动作是对象退出状态时的动作，出口动作的关键词为“exit”，后面加一个斜杠。如果动作是另一个对象发生的，则动作前要加“对象名、”。
9. 活动：在一个状态内的处理过程，可以分离成多个任务，活动可以中断而不影响状态。活动用关键词“do”，后加“/”或冒号。
10 内嵌状态图：又被称为子状态，状态图可以被嵌套在其他复合状态内部，组成状态的转换适合于它所有的内嵌状态图，常用来表示例外和异常，所有状态图的表示法都可以使用。
11 组成状态：可以分解为区域的状态，每个区域包括一个或多个子状态。组成状态又叫超状态。如果几个子状态之间互斥，被称为互斥子状态。为了区分不同的活动同时发生的情况，超状态被分解为区域，每个区域都是一个状态，子状态之间用虚线隔开。这被称为并发子状态。可以用画活动图的元素表示并发子状态的演化。

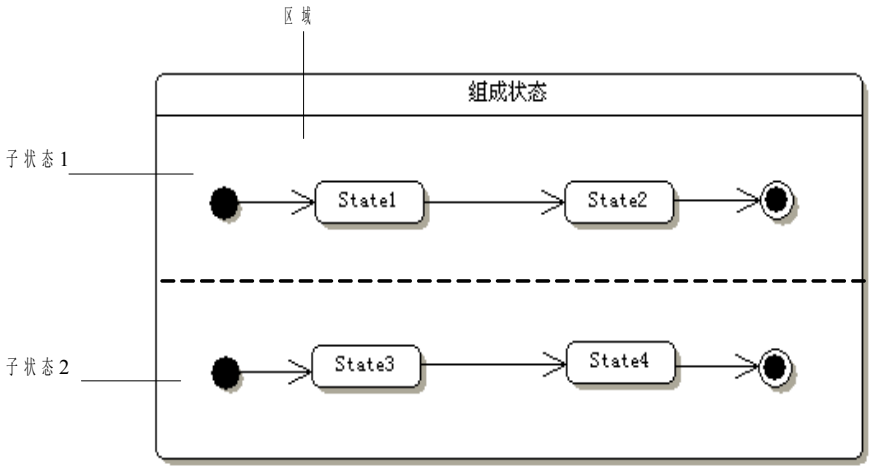


图 9.4 子状态、组成状态区域的表示示例

12. 历史状态：表示返回到以前的某个状态，用一个圆圈，内部加一个 H 来表示，如果是详细子状态，在 H 的由上角附加一个星号。

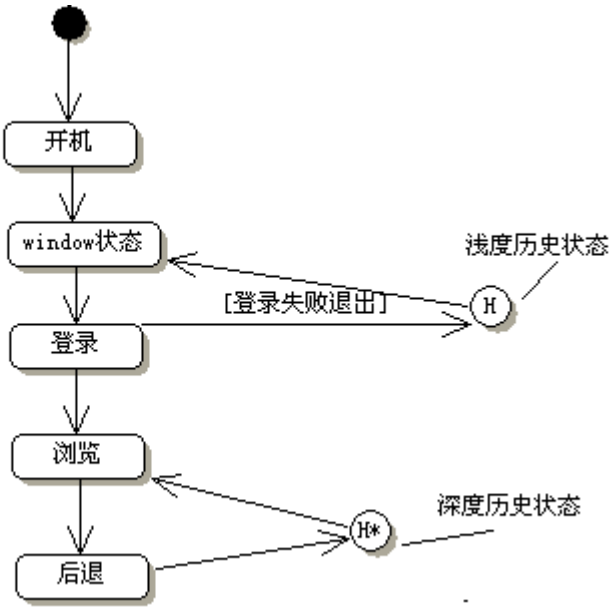


图 9.5 表示登录子状态退回到刚刚离开的 windows 子状态需要经过浅度历史伪状态；浏览了许多页面后，需要查看任意历史浏览页面，需要从当前页面，通过深度历史伪状态，退回到以前某一浏览页面。

13. 交叉伪状态:用一个实心圆表示转移的汇合点或者分支点，可以有一个或多个输入转移和一个或多个输出转移。

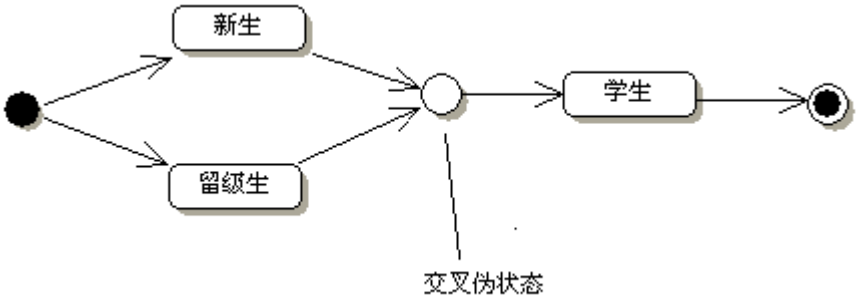


图 9.6 交叉伪状态的表示法

14. 选择伪状态：如果没有汇合的简单分支，就选择使用选择伪状态，分支可以是有条件的，并且分支是条件互斥的，只能触发一个输出转移。

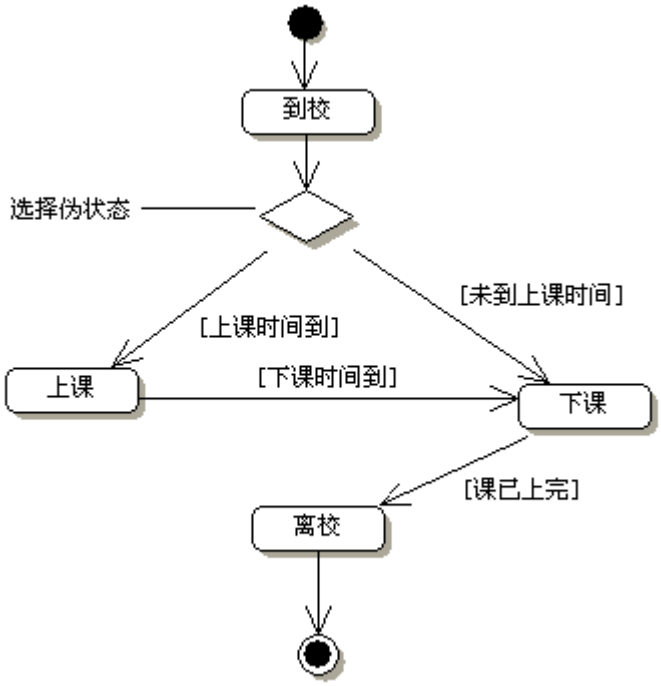


图 9.7 选择伪状态表示示例

9.1.2 状态图的建模分析步骤

（1）首先要确定进行系统控制的对象，可以从前面分析的顺序图中寻找。
（2）确定对象的起始状态和结束状态
（3）在对象的整个生命周期寻找有意义的控制状态
（4）寻找状态之间的转换
（5）补充引起转换的事件
（6）UML 建模工具画状态图
（7）补充必要的文档

9.1.3 车辆管理状态图的建模分析过程

状态图建模是针对系统中的对象，车辆管理系统的对象很多，比如车辆对象、车辆使用对象、车辆申请对象….。是不是对每个对象都画状态图呢？回答显然是否定的，一是时间所限，二是没有必要。我们只要把该软件系统需要控制（控制是计算机的能力）的对象的状态图画出来就可以了，这样我们就可以使用计算机达到对业务进行控制的目的。车辆管理的主要目的是对车辆的生命周期进行控制，达到高效有序的使用车辆，所以，我们可以对车辆进行状态图建模，如果我们要监视公司的车辆管理人员的办事效率，也可对用车申请对象的状态进行建模，如果要对司机进行在岗状态监控，还可以对司机对象进行状态图建模等等。由于篇幅所限，我们只对车辆对象进行了建模分析。

对于车辆对象来讲，整个生命周期有无数种状态，是不是把每种状态都画出来呢？回答当然是否定的，对于常见的有限状态，比如车辆自从购进后，有空闲、等待加油、正在加油、加油完成、等待修理、正在修理、修理完成等等也不是都要建模，我们只对影响对象责任的状态进行建模。所以只对于车辆对象的新购、空闲、预定、正在使用、维修、报废几种状态进行建模。

对于车辆对象的某个状态来讲，如果不特别强调某些行为是对象进入该状态时发生的行为或者是离开这个状态时发生的行为，一般都认为是对象处在这个状态时发出的行为，比如会议这个对象的有一个正在开会状态，进入这个状态时会发出一个“现在会议开始”这个消息，离开这个状态时会发出“现在会议结束“这个消息，处在这个状态时会发出”请安静“这个消息。显然这几个消息之间如果不用出入口动作来表示，将无法表示出消息这种关系，对于子状态，同样也是根据需要来创建，状态图建模切忌复杂，如果太复杂，反而达不到交流和引导开发的目的。在车辆管理中，我们对新购这个状态画出了出入口动作。其他状态无需标示出一些特殊的行为和子状态。

接下来的工作是建模，首先是要画出起始状态，这很容易，接下来的状态与所要处理的业务有关，如果你所关心的业务延伸到车辆的采购甚至签合同，哪儿就是车辆管理的第一个状态，我们的车辆管理主要关心的是车辆被购进单位以后的使用情况，所以新购是第一个状态，车辆在系统登记后，车辆对象的状态就变为空闲状态，在这个状态，标志该车辆可以被申请使用，一旦外部向这个对象发出使用申请、并且被批准后，该车辆的状态就变成预定状态；在空闲状态，如果是超期使用，也可变为报废状态。预定之后一旦为此而出车，哪怕在院子转了一圈，车辆的状态也变为正在使用状态，使用完毕后变为空闲状态，如果在使用过程中损坏，车辆的状态则变为维修状态，如果维修不好车辆的状态变为报废状态，否则变为空闲状态。这些状态之间的转换，完全是根据业务规则。这样我们就完成了对象状态图的建模。

9.1.4 状态图的建模案例

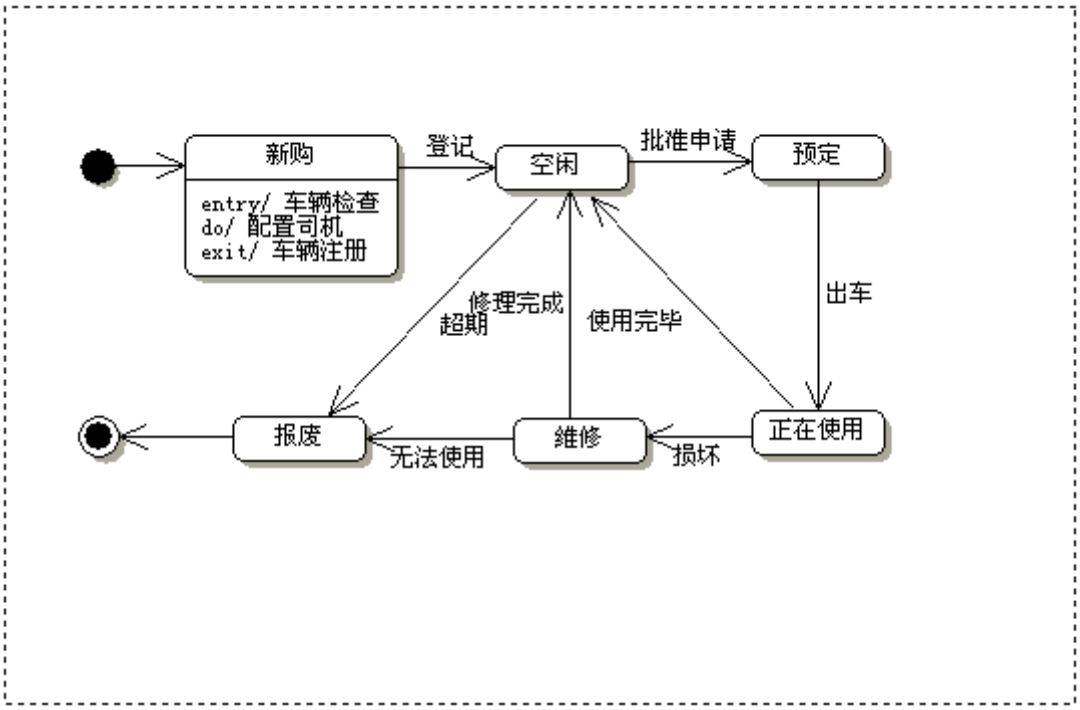


图 9.8 状态图建模案例

9.1.5 操作步骤

1. 在第四章第四节项目和包建模的基础上进行状态图建模，如果要直接建状态图模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 点击“<UML2 Model>”节点前面的“+”符号，在展开的导航树中再右击“车辆管理”前面的“+”符号。

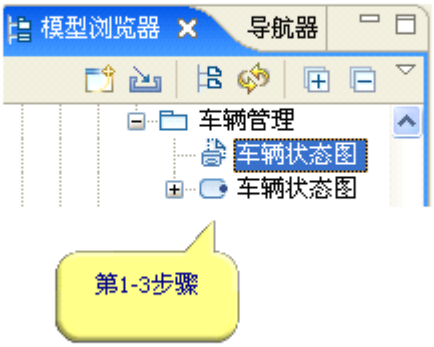


图 9.9 状态图建模操作步骤 1-3

4. 右击展开导航树上的“动态分析“节点，在弹出菜单中选择”新建框图“，再在"新建框图“菜单下选择”状态图“。则在该包下增加了一个节点，一个是以”statemachinadiagram“开头的节点，代表这个状态图的全局属性，并可在“常规"选项卡中修改该节点名称为“车辆管理系统车辆状态图"。并通过调整绘图区四周的锚点，调整绘图区大小。

图 9.10 建立状态图操作导航

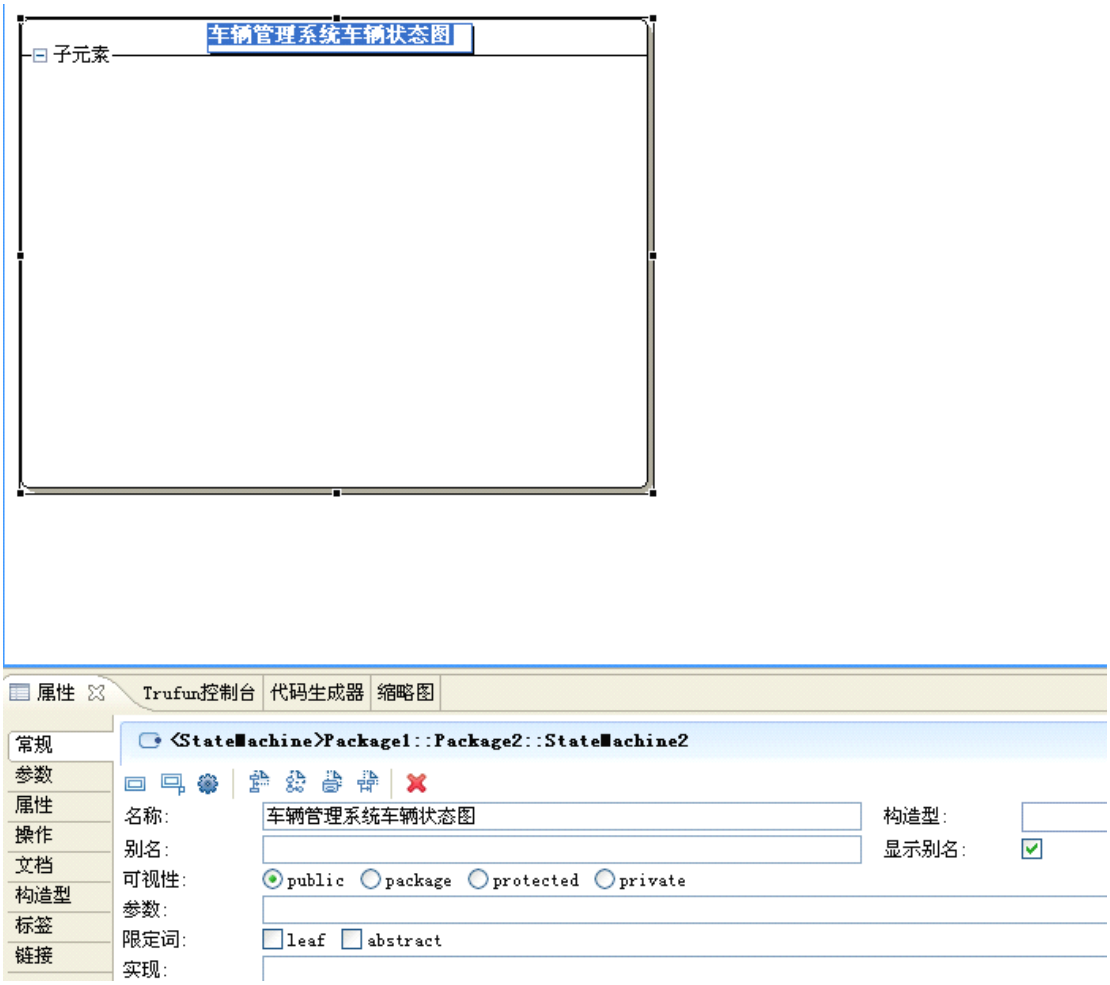
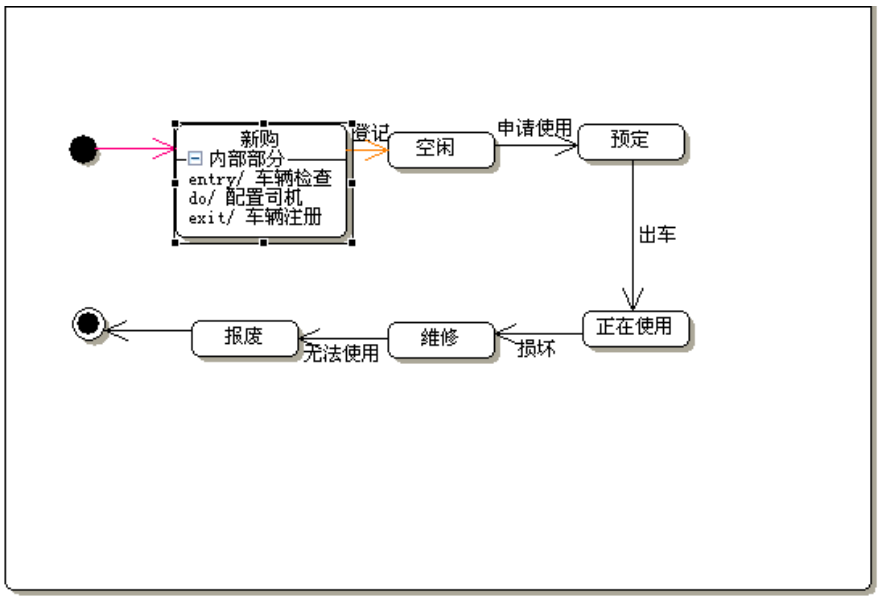


图 9.11 状态图的属性设置

5. 首先从绘图工具面板选中起始状态元素，将其拖入绘图区，再从绘图工具面板选中状态元素，在适当调整大小和位置后，在界面下面的“常规”选项卡中修改别名，输入“新购”。在名称栏目输入“new”，在入口动作栏目中输入“车辆检查“，在活动栏目中输入“配置司机“，在出口动作中输入”车辆注册“，以完成车辆对象的新购状态布局。同理可完成其他状态的布局，如：空闲、报废、预定、正在使用、维修。



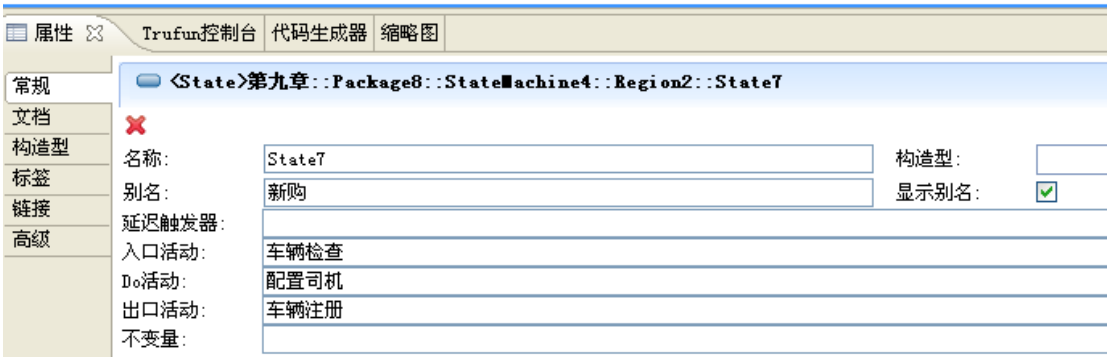


图 9.12 状态元素的属性设置

6. 首先从绘图工具面板选中转换元素，将其拖入绘图区联接新购和空闲状态，在界面下面的“常规”选项卡中触发器栏目，输入“调度”。在监护条件栏目输入“符合出车条件”，在效果栏目中输入“改变状态“。同理可完成其他状态之间的转换布局。
7. 如果该对象经分析有终止状态，首先从绘图工具面板选中终止状态元素，将其拖入绘图区，与相关状态联接。

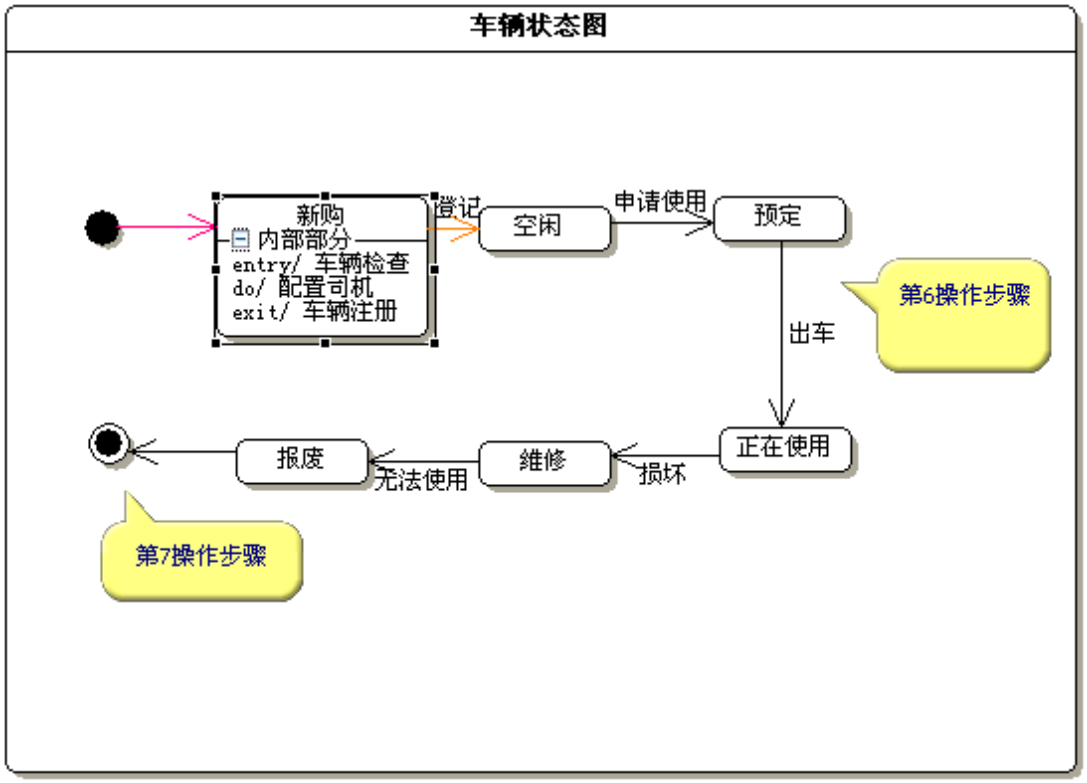


图 9.13 操作步骤的第 6—7 步骤示意图

9.1.6 会议管理系统主要分析类状态图

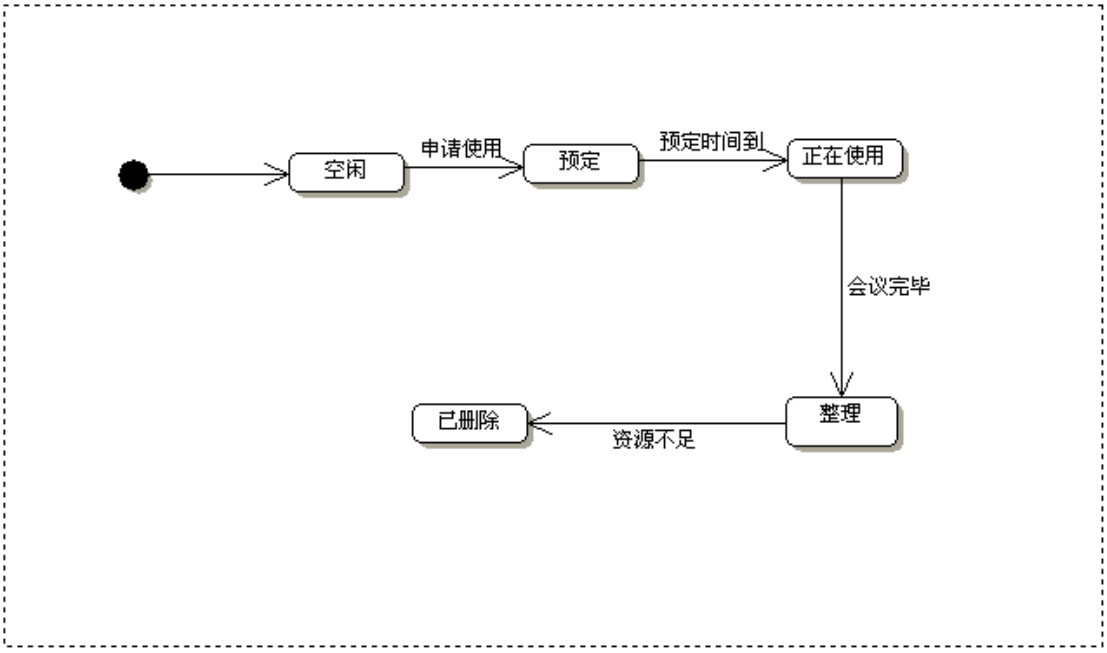


图 9.13 状态图建模案例

第二节 小结

本章是动态分析建模的组成部分，是顺序图建模的延续，我们重点讨论了对象的状态、事件、转换等模型元素。状态是用对象的一个或一组属性值来表示，外部事件，引起状态发生改变，也就是发生了转换，动作是事件触发的行为，它可能引起状态改变，也可能不引起状态改变。很多事件引起的动作都使对象转换到相同状态，则这些动作叫入口动作。相反则为出口动作，分解的状态叫子状态，是状态内的状态，子状态又分为互斥和并发两种，可以使用同步条，表示子状态的分叉和合并。对于事件发生后，各建模元素之间的变化顺序，也是本章要关注的一个重点，它对理解各个建模元素的语义有很重要的意义。

第三节 习题

1. 进行状态图建模的对象和状态来自哪里？
2. 状态图中都有那几类动作，它们和活动有什么区别？
3. 试编写会议管理中会议室状态画法操作步骤
4. 试对图书管理中图书对象画出状态图。
5. 试举例说明复合状态的语义。

第九章 系统实现

前面我们主要在分析领域从用例、类、交互几个大的方面完成建模的逻辑分析，从现在开始，我们将从实现角度出发，讨论软件的物理实现。物理实现主要从以下三个方面来描述，首先是用组件图描述软件，在用部署图反映硬件的物理结构，最后将两个图结合，反映软件在硬件上的分布。这种分布是战术型的，具体执行与语言平台有关。

实现模型是分析模型的一个实现视图，一个分析模型，由于其在实现阶段所选择的语言及支持系统不同，可以形成不同的实现模型。类描述了软件的逻辑组织意图，而构件描述了物理软件模块之间的关系，在这里，软件成了一组可以相互替换的模块单元，许多模块单元的集合可以构成更大的模块单元，使基于构件的开发思想成为可能。

部署图是展示系统实现环境的静态视图，对一个系统的管理可以从多个方面描述，每一个描述构成了一个部署视图，对整个系统的完整描述必须由多个部署视图来完成。

接口：

当我们谈起系统实现时，又有一个不可回避的概念就是接口，接口就是通过将功能性的说明和它的实现分离，说明部分就是接口，实现部分就是类、组件。实现部分必须遵循说明部分所定义的契约。接口说明的特征包括：操作、属性、关联、约束、构造型、标记值、协议。接口按其功能分为提供接口和需求接口。可替换性原则同样适用于接口和继承，这两种关系都可产生多态，那么，在那种情况下使用继承，那种情况下使用接口呢？一般来讲，如果要使元素之间的独立性强，系统具有更好的灵活性，并且使解决方案简洁、语意清晰，使用接口是最好的选择。端口是组合了一组语意上相关的提供接口和需求接口，表示类和环境之间的交互。

子系统：

与实现有关的另外一个概念是子系统，子系统用来把大系统分解为可管理的功能逻辑构件，它是一个非常具体的分组机制，每个子系统都有行为，子系统的行为通过规范或子系统接口来表示。在 UML2.0 中，子系统是组件的一种类型，往往比组件更大的组合。子系统本身不会被实例化（抽象的），但其内容组件可以被实例化，子系统用来分离设计和实现，或者代表更大颗粒的组件，也可以用子系统解决系统遗留的问题。识别子系统时，需要注意一下几点：

（1）子系统往往是可以在一定程度上独立开发、发布和部署的主要功能部分，在功能、逻辑和物理上是内聚的。

（2） 子系统的边界与技术、政治、法律有关

（3）要描述子系统的语境，以反映子系统和周围的角色之间的交互协作。就象描述系统一样。

（4）子系统在较低的抽象层次可以被看做系统。

（5）从实现角度看，与子系统相关联的是制品。

（6）在进行系统和子系统建模时，只描述他们之间的基本聚合关系，而将它们之间的连接细节放到较低层次的图中。

第一节 组件图

10.1.1 组件图的建模元素

1. 组件：又叫构件，代表从类到应用、子系统和系统的任何事物，组件是对物理实现类型的定义，是一个抽象的物理软件需求。每个组件定义了一个或多个和其他构件通信的接口，组件暴露了所包含类的一个或多个接口，而不是全部接口（其余用于内部通信）。一方面组件依靠自身的类指定其行为。另一方面，组件依靠多个实现它的制品。组件是一种容器，没有自己的特征，一般用组件表示一个业务过程。组件用一个矩形和位于其左边上的两个小矩形组成一个小图标，放在一个矩形框里，组件的名称方在矩形里。

图 10.1 组件的表示法

2. 制品：制品定义了真实世界事物的规格说明，如源文件、链接库等。根据制品的作用不同，组件又被分为执行组件、部署组件、产品组件。制品用原型<<artifacts>>表示，制品是设计逻辑结构到实现的映射，制品和组件的关系就象对象和类的关系。

图 10.2 制品的表示法

3. 组件原型：对组件在体系中扮演的角色提供可视性的暗示，常用的组件原型有：

（1） executable：在过程机上运行的软件
（2） library：运行时可执行文件应用的资源。
（3） table：可执行的数据库组件。
（4） file：数据和代码总和
（5） document：文档页

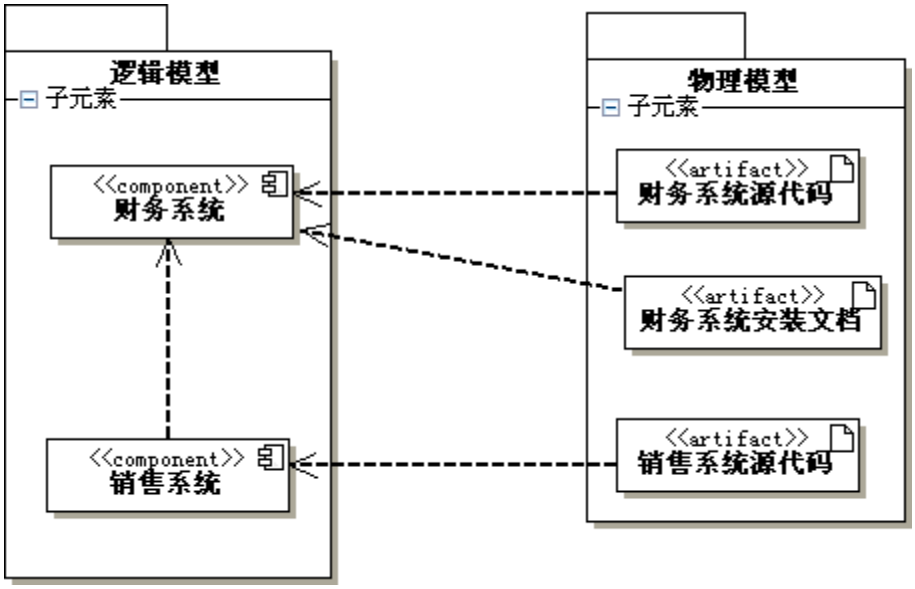


图 10.3 组件和制品之间关系示意图

4. 组件接口：表示组件接口有两种方式，一种是用一个带原型<<interface>>的类，另一个表示法是用一条直线，一头连接于组件，另一头连接于一个小圆圈，表示提供接口，如果用半圆表示需求接口。

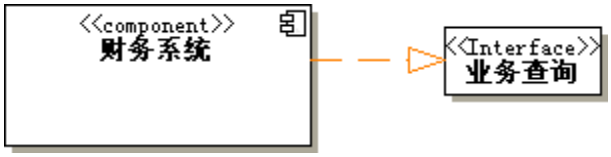
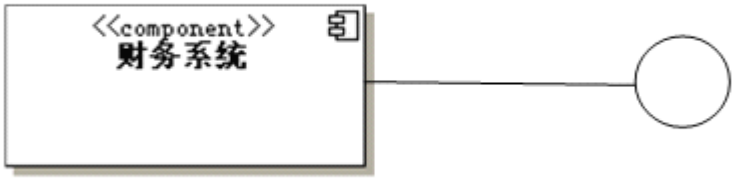


图 10.4 使用带原型的类表示接口



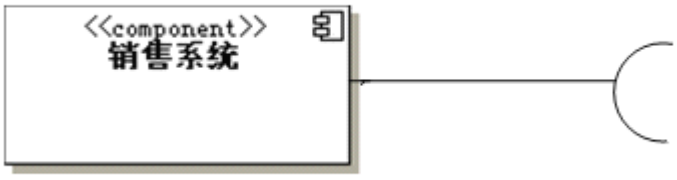


图 10.5 上半部分表示提供接口表示法，下半部分表示需求接口表示法

5. 组件的依赖：用一个虚线箭头表示，箭尾位于依赖组件，箭头位于被依赖组件。

图 10.6 组件之间的依赖关系表示法

6. 端口：实现组件接口与内部实现之间的转换，端口用在组件矩形边框上的一个小方块表示。它和实现之间用一个带虚线箭头的连接器表示。端口具有状态、名称、可视性、多重性等特性。

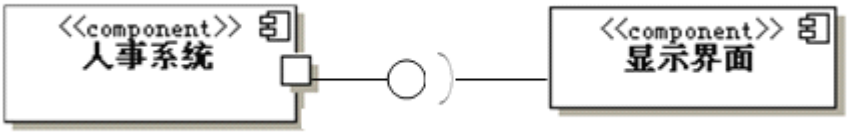


图 10.7 端口的表示法

7. 子系统：

子系统表示有两种方式，一种是用一个矩形，再加上构造型《subsystem》，另一种是用分栏式，分为左右两栏，左边是规范定义部分，右边是实现类部分，一般情况下，使用前一种表示方式就可以了。

10.1.2 组件图建模分析过程

从前面的分析我们已经清楚的知道，从现在开始我们所分析的事物都是类及类以上的级别，但还属于系统内部（前面讲的用例是系统层次，静态分析是类层次，动态分析是对象层次的，现在要分析的相组件当子系统层次的），是前面静态分析的类的进一步抽象。那是不是说组件的组成就是类呢？回答显然是错误的，这是由于组件是实现阶段的建模元素，而类是分析阶段的建模元素。组件实际上是由软件的一个个具体物理实现来构成，这些实现不外乎代码和相关的文档，它们统称为制品。由于子系统往往对应一个业务过程，所以组件图也是对一个过程的描述，记得前面我们曾学习过一个分析模型 ：活动图，它也是用来描述业务过程的，它们的区别是一个站在业务过程里面进行描述，一个站在业务过程外面进行描述。封装是组件的一大特征，可替换性是使用这个元素所带来的设计优势。

在明白了组件以上概念以后，让我们对综合行政管理系统进行分析，首先要作的工作是对业务过程的识别，需要从以下几个方面进行考虑，首先要站在系统角度识别组件，在综合行政管理系统，在需求采集阶段，我们就把整个系统分为 6 个主要业务过程，所以，可以确定系统打包的组件就这 6 个，即考勤管理组件、办公用品组件、固定资产组件、图书管理组件、车辆管理组件以及会议管理组件。其次要从业务过程进行识别，在这里你可能要问，每个组件应该包括几个业务过程呢？这个没有一个确定的数目，一般要求满足单一性原则，以便把接口封装为端口。从第四章的需求来看，在这 6 个组件中，每个组件事实上只有一业务个过程，其他的工作都是围绕这个业务过程，并为业务过程提供基础信息，如考勤子系统的业务过程是考勤登记、车辆管理子系统的业务过程是用车申请、会议管理子系统的业务过程是会议申请、固定资产子

系统的业务过程是折旧、图书管理子系统的业务过程是图书借阅、办公用品子系统的业务过程是办公用品领用。第三是从独立性角度进行识别，由于组件和其他组件只能是依赖关系，如果业务之间之间关系密切，就很难分成不同的业务组件。最后要从可替换性进行考虑，设想如果把该组件换掉，比如用新的考勤组件代替旧的考勤组件，其他程序不用改，只要改一改配置文件就可以了。当然在软件开发过程中，由于新需求的增加，使原组件中的业务过程比较多，这时可考虑在组件内部再封装组件的办法，这在设计中是允许的。

上面讨论的组件封装主要从业务角度进行，事实上，对于界面操作，也可以封装为组件，以供客户端下载使用，对嵌入式系统，界面组件往往是必须的。综合行政管理系统没有这样的特殊需求。所以也就没有设计这样的组件。有些系统，出于安全和性能等方面的考虑，专门设计了数据库访问组件，综合行政管理系统没有这样的特殊需求。所以也没有设计这样的组件。

在找到组件以后，就需要找组件和组件之间的关系，如果该软件系统未来要和其他系统有接口，需要用组件图表示这种关系，这时新开发的软件系统、其他软件系统都用组件建模元素来表示，它们之间用依赖关系连接起来，综合行政管理系统没有这样的特殊需求。所以也没有进行这方面的建模。

如果要想反映组件的组成情况，可以把组成组件的制品都画出来，再用依赖关系把组件和制品连接起来。综合行政管理系统没有这样的特殊需求。所以也没有进行这方面的建模。

如果要想反映组件内包含组件的情况，可以在组件中再画出所包含的组件，要画出内部组件之间的接口关系，并且要画出内部组件和端口之间的关系。综合行政管理系统没有这样的特殊需求。所以也没有进行这方面的建模。

10.1.3 组件图建模案例

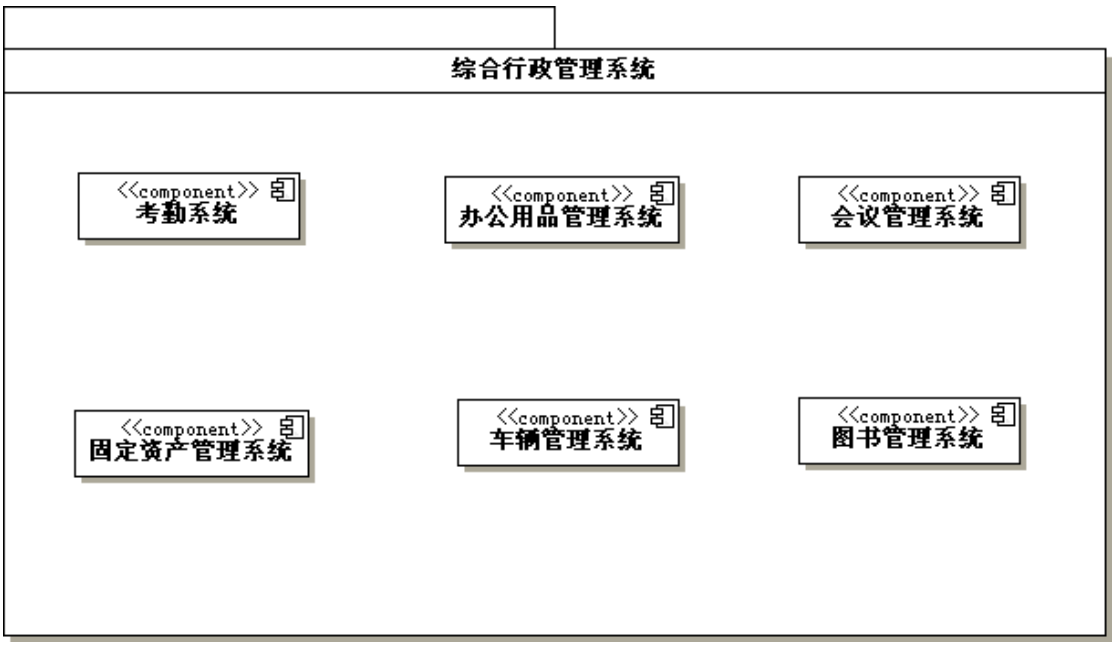


图 10.9 组件图案例

10.1.4 操作步骤

1. 在第四章第四节项目和包建模的基础上进行组件图建模，如果要直接建组件图模型，项目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 右击展开导航树上的“mode1“节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”组件图“。则在该包下增加了一个节点，一个是以”componentdiagram“开头的节点，代表这个组件图的全局属性，并可在“常规”选项卡中修改该节点名称为“系统组件图”。

图 10.10 组件图的建模操作导航

4. 首先从绘图工具面板选中包元素，将其拖入绘图区，并通过调整绘图区四周的锚点，调整绘图区大小。在界面下面的“常规”选项卡中修改别名，输入“综合行政管理系统”。在名称栏目输入“component”。

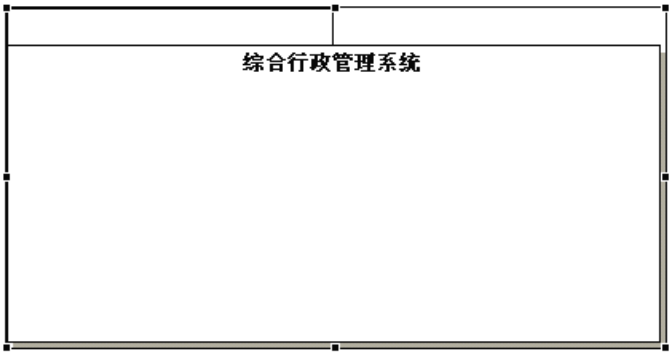


图 10.11 组件图的属性设置

5. 再从绘图工具面板选中组件元素，将其拖入绘图区，并通过调整绘图区四周的锚点，调整绘图区大小。在界面下面的“常规”选项卡中修改别名，输入“车辆管理”。在名称栏目输入“carmanage”，在从绘图工具面板选中制品元素，将其拖入绘图区中车辆管理组件上，并通过调整绘图区四周的锚点，调整大小。在界面下面的“常规”选项卡中修改别名，输入“车辆管理”。在名称栏目输入“applicateJAR”，这样就完成了车辆管理组件的布局。同理可完成界面组件、会议管理组件、办公用品管理组件、图书管理组件、考勤管理组件、固定资产管理组件的布局。
6. 再从绘图工具面板选中依赖元素，将其拖入绘图区连接界面组件和车辆申请组件，同理可以建立其他组件之间的连接。

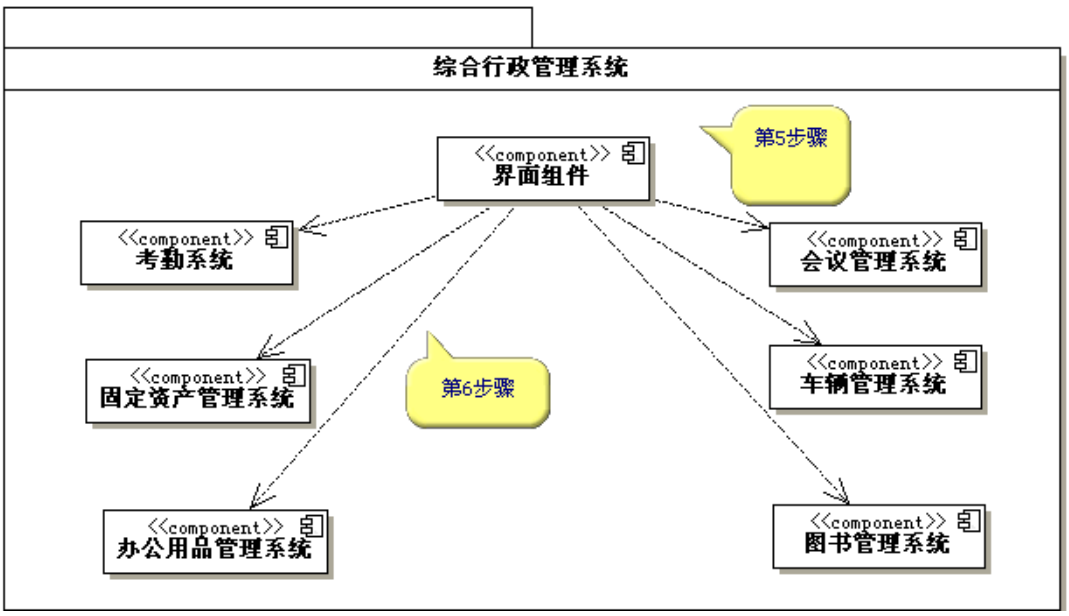


图 10.12 组件图建模操作步骤 5-6

第二节 部署图

10.2.1 部署图的建模元素

1. 节点：部署图是由节点和节点之间的关联组成，节点代表处理资源的类型或执行工作的任何事物，其上的制品可以被部署和执行。节点具有属性和操作（有点象类），也可参与关联，节点的属性和操作一般用注释来表示。节点的图标是一个平面画的立方体。特别要注意类型级节点和实例级节点在名称的区别。

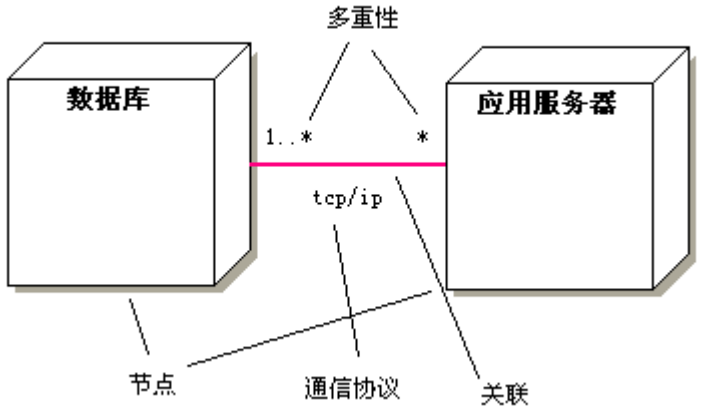


图 10.13 节点及节点之间的关系表示示例

2. 节点关联：表示节点之间通信类型，它是构成网络的基础，它关心节点是如何连接的，一般使用协议原型表示 如：<<tcp>>.，关联上可以添加多重性。
3. 节点原型：为了将节点的行为和结构归类，常用的节点原型有：
   （1） application server：远程执行应用的后台计算机。
   （2） client workstation：执行应用的用户接口计算机。
   （3） Mobile device：移动设备
   （4） Embedded device：嵌入式设备。
4. 设备子类：表示物理的计算单元或人的组织单位，其父类为节点。
5. 执行环境子类：定义一组对软件提供的支持服务，其父类为节点。
6. 部署规范：把制品放在它保存和使用它的位置就是部署，部署的目标是节点，部署规范中定义了如何向节点分配的模式和参数，部署规范为制品。

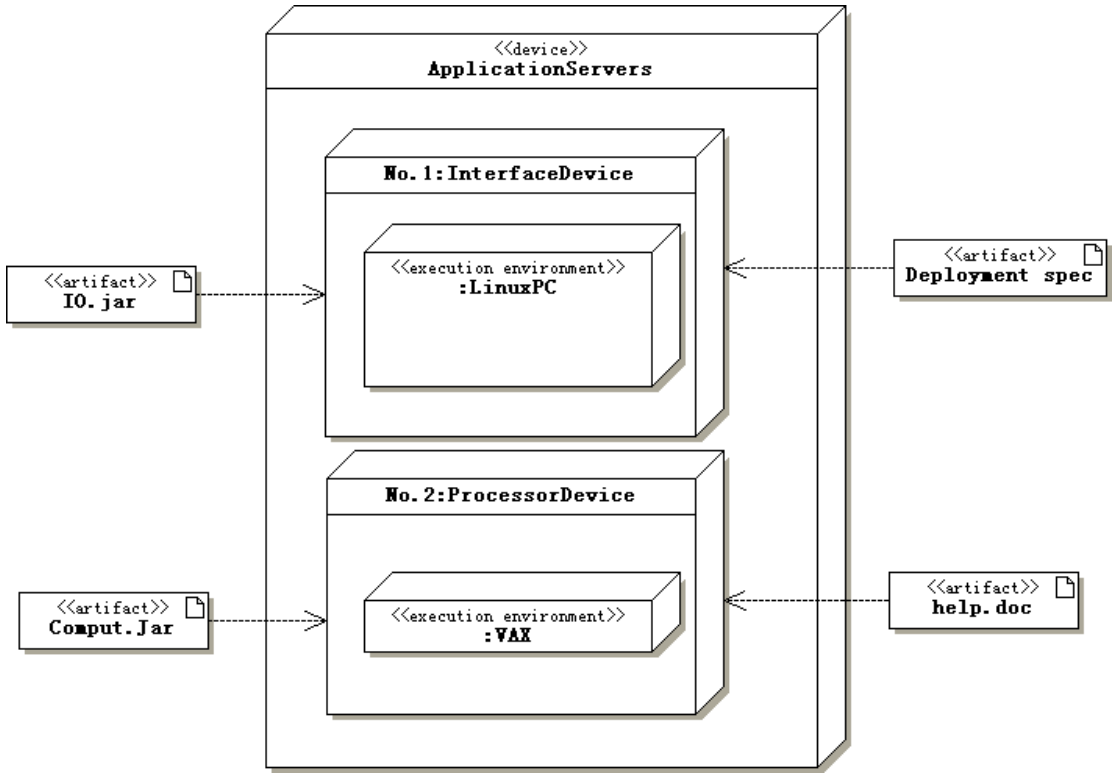


图 10 .14 制品与节点类、节点对象的关系示例

7. 节点泛化：节点具有类的性质，因此，和类一样，节点同样包括有子节点，它的操作和属性可以被继承。

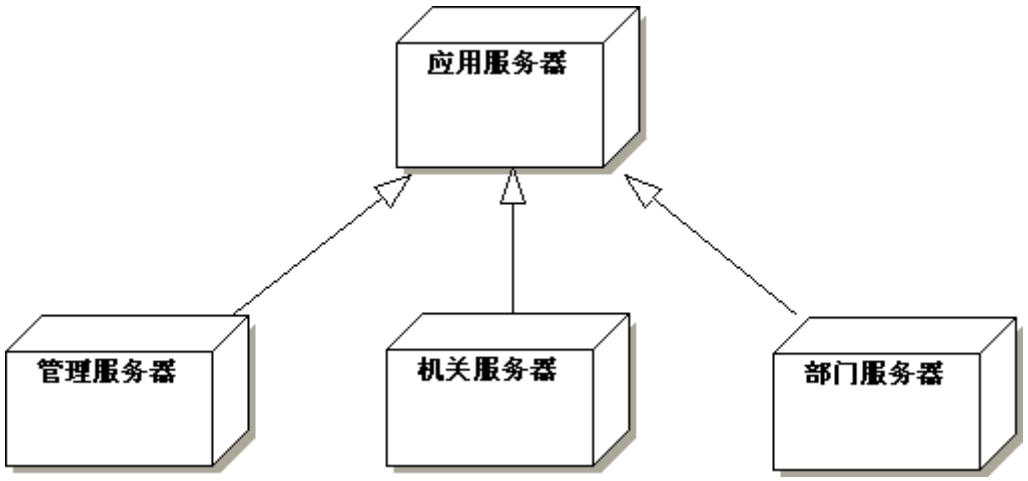


图 10.15 节点泛化示例

8.节点的属性：用一组特征值来描述实体，如服务器 cpu 的速度，内存容量、端口编号、所属设备名称，安装位置等等。

9.节点操作：就是在该节点内的软件所能提供的服务，注意，这里指的是应用服务而不是底层操作。

10.节点的其他特性：

（1）可以接收事件。

（2）有自己的行为规范。
（3）可以拥有端口。
（4）拥有内部结构
（5）可以包含节点。

11. 制品的定义
（1）制品具有属性和操作
（2）制品可以包含其他制品，如页面制品可以包含 applet 小程序制品。
（3）制品和使用制品之间是一种“呈现”依赖关系。

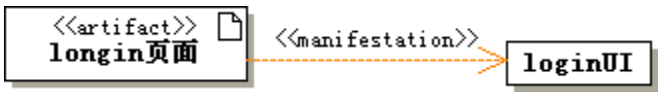


图 10.16 制品和制品使用之间的关系

（4）制品分为《source》和《executable》两种分类原型，当然也有<<JSP>>、<<Servlet>>、<<EJB>>、<<HTML>>标准原型。

10.2.2 部署图建模分析过程

部署图是以软件运行时的硬件节点为中心，从静态方面反映软件与硬件的关系，包括硬件之间连接关系，软件在硬件上的部署关系以及系统与环境的关系。与此相对应有一个网络拓扑结果图，该图不是属于 UML 的范畴，但也用来表示系统的物理网络结构，只是比部署图更抽象，根据侧重点不同，画图建模的内容也不同，一种是侧重于硬件之间的连接，画出各个硬件节点和它们之间的连接关系，其次是在逻辑层次上建模，既表现出各个硬件节点和它们之间的连接关系，又表现出逻辑组件在硬件上的部署关系，最后一种表现方式是在部署层次的建模，既表现出各个硬件节点和它们之间的连接关系，又表现出部署在各个节点上的制品及部署文件。不管哪种方式建模，建模的第一步是寻找物理建模节点。

综合行政管理系统采用三层架构，其硬件节点包括客户端、应用服务器、数据库服务器，当然还有网络设备，在部署图中往往不画网络设备（网络设备一般在网络拓扑结构图上画出），这个也是被大家默认接受的，这是由于部署图与网络拓扑结构图不同，它要反映软件在硬件上的部署关系，而网络设备如路由器、交换机、防火墙一般是不部署应用软件（特殊情况除外）。客户端一般也画两个或多个，以表示适用于多种客户端。接着要确定几个节点之间的通信连接关系，包括多重性和连接协议。，一个应用服务器可以接多个 a 类型和 b 类型的客户端，一个客户端可以连接一个应用服务器。它们之间的协议是《tcp/ip》，同理可以识别出应用服务器和数据库之间的关系是 1：1，它们之间的通信协议是《jdbc》。

在确定了硬件节点和它们之间的关系之后，就需要确定软件在硬件上的部署情况，何种方式部署，完全取决于软件系统的架构选择，由于本系统选择了三层架构，所以，考勤管理系统制品、会议管理系统制品、车辆管理系统制品、固定资产管理系统制品、图书管理系统制品、办公用品管理系统制品都部署在应用服务器上，当然，如果开发了部署文件，也可用制品表示出来。

10.2.3 部署图建模案例

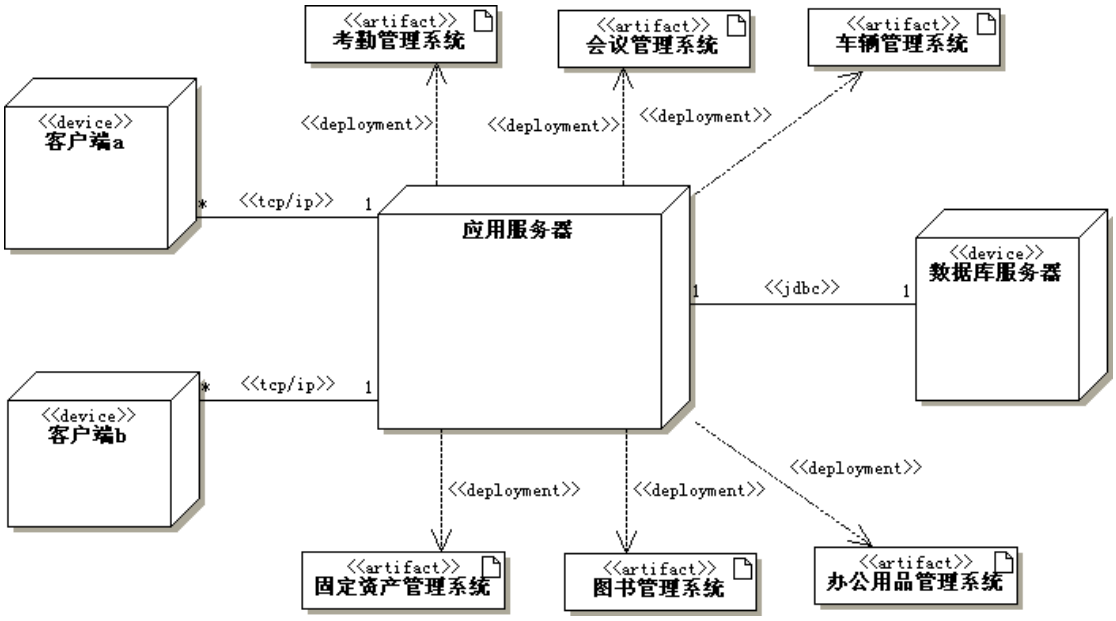
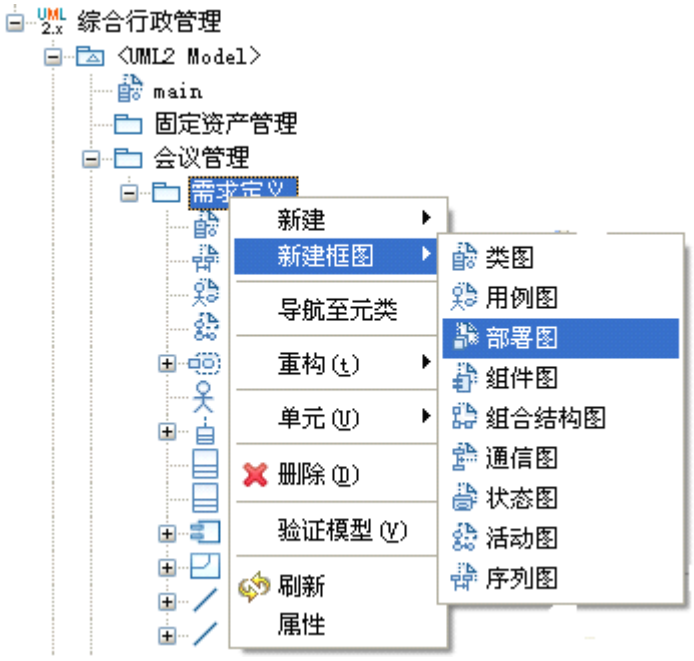


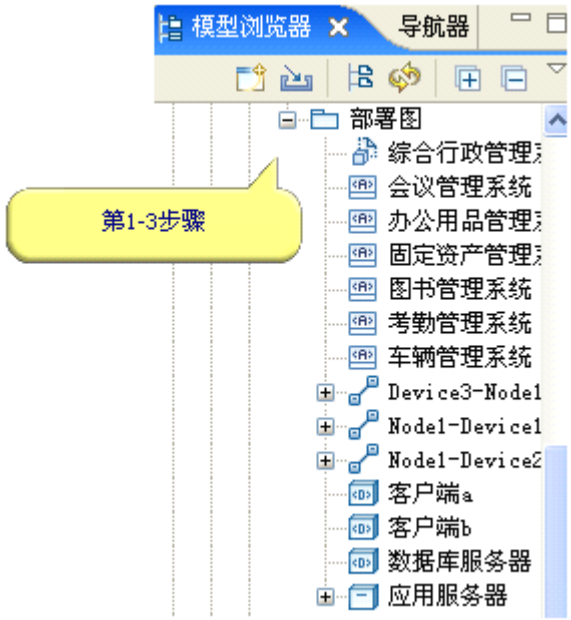
图 10.17 部署图建模案例

10.2.4 操作步骤

1. 在第四章第四节项目和包建模的基础上进行部署图建模，如果要直接建部署图模型，项 目和包的建模操作步骤和第四章第四节相同。
2. 在主界面左侧的模型浏览器导航栏，选择项目“综合行政管理“，并点击项目名称前面的”+“符号，系统显示缺省的该项目文件目录树。
3. 右击展开导航树上的“<UML2 Model> “节点，在弹出菜单中选择”新建框图“，再在”新建框图“菜单下选择”部署图“。则在该包下增加了一个节点，一个是以”deploymentdiagram“开头的节点，代表这个部署图的全局属性，并可在“常规”选项卡中修改该节点名称为“系统部署图”。



（a）



(b)

图 10.18 部署图建模操作导航界面

图 10.6 部署图的属性设置

4. 从绘图工具面板选中节点元素，将其拖入绘图区，并通过调整四周的锚点，改变大小。在界面下面的“常规”选项卡中修改别名，输入“应用服务器”。在名称栏目输入

“application server”，再从绘图控制面板选中设备元素，将其拖入绘图区，并通过调整四周的锚点改变大小。在界面下面的“常规”选项卡中修改别名，输入“数据库服务器”。在名称栏目输入“database server”。同理可以完成客户端节点元素的布局。

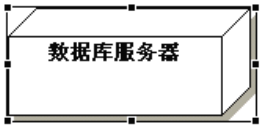


图 10.19 节点的属性设置

5. 从绘图工具面板选中制品元素，将其拖入绘图区，并通过调整四周的锚点改变大小。在界面下面的“常规”选项卡中修改别名，输入“车辆管理”。在名称栏目输入“carmanageJAR”，这样就完成了车辆管理制品的布局。同理可完成会议管理制品、办公用品管理制品、图书管理制品、考勤管理制品、固定资产管理制品的布局。
6. 再从绘图工具面板选中部署元素，将其拖入绘图区连接服务器节点和车辆管理制品，同理可以建立其他制品和节点之间的连接。
7. 从绘图工具面板选中关联元素，将其拖入绘图区连接服务器节点和数据库节点在界面下面的“常规”选项卡中修改别名，输入“通信”。在名称栏目输入“<<JDBC>>”，点击角色 A（B）后面的链接，在显示的新页面的多重性选项中改变角色 A（B）的多重性。同样操作可以完成所有节点关联的布局，

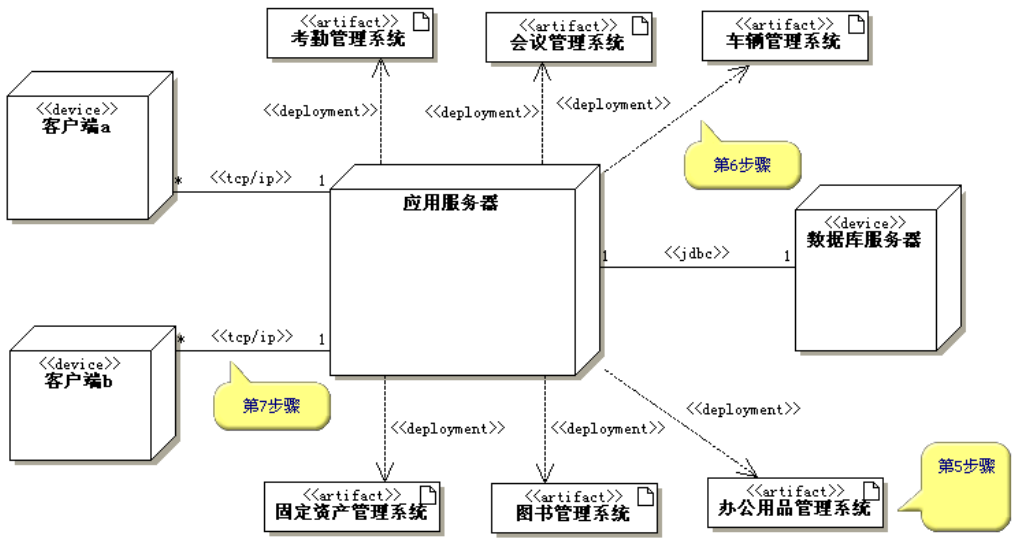


图 10.20 部署图建模步骤 5-7

第三节 小结

进行组件图和部署图建模，是整个建模的最后部分，在这里有三个重要的建模元素，即：制品、组件、节点，制品是一个真实的物理实现，与平台有很大的关系，而组件是对制品的一种软件逻辑管理，是类图的一个映射，而节点是一个物理的环境，制品只有部署到节点上，才能发挥其作用。节点之间用关联链接，节点和制品之间用依赖链接，组件之间用依赖链接，节点内部包含组件，组件内部包含制品。

第四节 习题

1. 简述组件图和部署图的建模语义差别。
2. 简述组件和制品的区别。
3. 试分析操作、接口、端口、需求接口、提供接口之间的关系。

第十章TUP（Trufun 统一过程）简介

第一节 UML 建模与软件开发过程模型

通过前面的讨论，想必大家对 UML 建模已经有了一个概括的了解，现在谈到软件开发过程，大家可能也不会陌生，学过软件工程的人都能随口说上几个软件过程模型，现在要把这两种不同的模型拿到一起来讨论，一方面是软件开发的实际需要，另一方面也是 UML 建模工具要和其他面向对象开发模型结合的一种必然要求。

但是，OMG 为了防止 UML 建模和某种开发过程模型结合过紧，导致其适应性降低，使统一性大打折扣，从而影响 UML 建模工具的普及和推广，只制定了语义规则和表示符号，对于一个实际问题怎样进行建模，并未制定象数据库设计范式那样的规范和原则，对于一个项目，应该先建什么模型，后建什么模型，也没有做什么限制。也就是说，没有规定 UML 建模的工作过程和方法，UML 建模可以适应任何开发过程模型。

软件开发过程模型的理论定义比较简单，而把这一过程模型在实践中应用成功，却有许多制约因素，首先是软件的范围，一个大型分布式软件系统和一个单机版的个人软件系统在开发管理上肯定不同；其次软件的开发目的，一个为了提高浏览量而开发的网站和一个为密集计算而开发的的一个处理系统在开发过程管理上肯定不同。最后一点是团队，不同的团队在磨合度、个人能力、团队协作等方面各不相同，开发相同的项目使用相同的开发过程模型，开发结果完全不同的实例多得数不胜数。另外，软件复用是面向对象的一大特点，它不但与所选择的开发过程模型有关系，而且与企业文化和企业的做事方式有关。

上面这一些都说明，选择或设计一个好的，能够反映软件开发过程在什么时候做什么、如何作的过程模型并不是件容易的事。UML 建模工具和统一过程（RUP）结合，是很多人

熟知的理论，这很大程度上得益于 UML 三位主要创始人的功劳，因为它们曾共同出过一本关于 UML 与统一过程的书，另一方面是 UML 建模工具和统一过程的发源地都是 rational 公司，也使人们误认为使用 UML 建模工具就得使用统一过程，事实上，UML 自 1.0 版本以后，就归 OMG 所有，而 RUP 不是 OMG 发布的，只有 OMG 发布的信息，才能作为我们的行业标准。

一切先进的思想，往往是融合了先前其他人的先进思想，在介绍 trufun 的 TUP 建模过程之前，我们有必要回顾一下和 UML 建模结合的几种软件开发过程模型。

统一过程（UP）模型：统一过程模型在和 UML 建模结合时，采用以用例为驱动的方式，用用例连接所有活动，每个活动都建一组模型，如业务领域模型、责任领域模型、实现模型、测试模型，每组模型中又由多个不同的角色共同协作完成，比如具有专门进行用例建模的角色和组件建模的角色等等，采用增量迭代方式建立和完善用例，并对每一次建模进行评估，在项目的计划、监控等方面并非以建模为中心，而是把建模作为统一过程的一个小部分。该模型的主要缺点是周期长、人员要求多、建模工作量大。

迭代模型：它是采用较多的小迭代来实现最终的模型，也就是说，模型图是通过一系列步骤一步一步地建起来，每一次迭代都有新信息添加到模型中来，每一次迭代都要经过评估，都是下一次迭代的输入，迭代会使系统开发的活动（需求、分析、设计和测试）执行多次，并且每次都有新的内容增加进来。这个方法有一个缺点是在迭代的后期，仍然有新的需求增加进来。

增量模型：增量模型开发每次迭代都能产生一个可执行的结果，这个结果是一个可“交付的”系统版本，每一次迭代要经过评估，并且增加了一些新的功能，增量模型主要包括分析、设计、实现、测试四个活动。该方法有一个很大缺点是到了项目迭代后期还要进行设计，会给系统带来很大的风险。

XP 模型：又叫极限编程，它是一个轻量级的、灵巧的软件开发方法；同时它也是一个非常严谨和周密的方法。它的基础和价值观是交流、朴素、反馈和勇气；即，任何一个软件项目都可以从四个方面入手进行改善：加强交流；从简单做起；寻求反馈；勇于实事求是，整个开发是以测试为驱动的，它属于小型方法，对于初级软件开发企业有效，无法站在软件过程的行列谈和 UML 建模结合的问题。

第二节 TUP 的定义

前面我们提到的软件开发过程模型，都能实现从软件需求到最终软件的工程管理，每一种过程模型所围绕的中心不同，统一过程模型是以架构为中心，增量模型是以建立可执行版本为中心，而迭代模型是以需求变化为中心，它们各自独立演化，在开始定义时并未考虑 UML 建模，UML 建模只是作为它的一个管理对象在使用，地位比较低下。随着软件的复杂度、大型化的程度不断提高，交流分析的重要性日益凸显，可以预料，UML 建模在软件开发中将会被提到很高的地位。根据我们长期对 UML 建模技术的研究和跟踪，提出了 TUP 开发过程模型，它有一个很大的特点，就是以模型为中心，以逻辑推理手段，以追求精确为目标，分别采用迭代、增量、构件开发方法，最终实现从软件需求到最终软件的开发过程的组织管理。

软件的最终实现代码，它可以被看成是那一种模型，是用某种语言代码写成的模型；需求原始资料是文档，它也可以看成是用户业务的一种抽象模型，同时它也是最终实现代码模型的某个层次抽象；连接软件的原始需求到最终实现代码之间的 UML 建模，更是一种模型，它是一种图形化的模型，每一种模型都是最终代码模型在某种程度上的抽象，也是需求原始资料模型在某种程度的精确化。TUP 强调的就是以这一个个模型为中心，来组织

软件的开发过程。

从原始软件需求资料模型到最终的实现代码模型，这中间存在多种演化可能性模型，为了使软件的分析设计具有更加坚实的科学基础，避免因分析人员的技术背景、设计风格而造成的演化模型差异过大，把软件分析设计过程设计成如下过程：即通过对原始软件需求模型，施加约束和提供条件，使模型逻辑转化为下一层演化模型，下一层演化模型比上一层演化模型更精确，然后再对下一层模型施加约束和提供条件，通过逻辑推理，再推出更新的演化模型，如此反复，直到形成最终的代码模型。

从前面的分析我们知道，最终实现代码模型以前的所有模型，都是最终实现代码模型的某种层次的抽象，所以软件的开发过程，是一个由粗到细的建模过程，每一个阶段的目标是以追求更加精确为目标。这和面向对象中的泛化特化概念是一致的。

长期的软件开发团队管理和丰富的教学一线工作经验，使我们认识到：软件开发各个阶段的活动，因为其支持和协作环境不同，而重现出不同的特点（在这里，我们强调软件开发过程的环境），由于环境不同，如果在软件的开发过程中自始至终采用同一个过程模型，将无法发挥开发过程与环境的最大程度的配合，也就无法使成果最大化，最优化，在继承前人研究的过程模型的基础上，我们提出了一种混合过程模型。

在草图建模阶段，以迭代方式组织过程活动，最大程度地捕获需求的范围和目标，主要包括 5 个核心活动，它们的依次顺序为：计划活动、交流活动、建模活动、确认活动，修改活动，每个迭代主要由这几个活动组成，每次迭代最后都要经过开发团队评审。需要说明的一点，这里的建模可以是 UML 建模，也可以是原型建模。

在蓝图建模阶段，以增量方式组织过程活动，蓝图阶段的建模主要分为四种活动，系统建模活动、子系统建模活动、重构活动、文档提交活动，重构活动是前两种活动的补充，在草图阶段结束后，并不是开始所有的建模，为了在分析的前期确定系统的架构雏形，首先要建模的系统模型，主要是影响架构的核心分析类作静态建模和对影响架构的主要业务机制进行动态建模。经过评审之后把此模型信息反馈给用户，形成一个有价值的模型，在用户对系统模型认可之后，就可以开展其他子系统的建模，从而形成一个模型增量，如果在建模期间，配置或需求发生了变化，在经过草图建模之后，融合到相应的系统或子系统建模中去。

在精图建模阶段，以复用和构件开发方式组织过程活动，对于大型复杂系统来讲，这种结合能充分发挥 UML 蓝图的易于评估交流和组件的快速开发特性。它主要包括以下几个活动，系统选择、转化细化、新构件的设计开发、软件测试、模型库管理。基于构件的开发活动从标识候选构件开始，通过搜查已有构件库，确认所需要的构件是否已经存在。如果已经存在，则从构件库中提取出来复用；否则采用面向对象方法开发它。之后利用提取出来的构件通过语法和语义检查后将这些构件通过胶合代码组装到一起实现系统。

第三节 TUP 的目标

TUP 软件开发过程主要希望达到以下目标：

1. 简化建模过程，是建模过程的工作量减少，无需建立多套模型。
2. 无需多个角色相互完成一个模型，建模过程只有两个角色，系统建模人员和子系统建模人员，系统建模人员主要完成与系统架构有关的建模和子系统之间的关系建模。子系统建模人员每人负责一个子系统，完成从蓝图到精图的全部建模，避免了由于频繁交流而引起不必要的成本增加。
3. 整个建模过程概念统一，避免对同一个概念为了适应过程模型而作不同修改，比如出

现象“业务用例、分析用例、实现用例、测试用例”等大量的新概念。

4. 与以前的面向对象概念接轨，如 OOA，OOD，OOT 等概念在 TUP 可以找到相应的位置。并且和以前的定义不矛盾。
5. 从软件开发过程来看，覆盖了从需求定义到软件实施以后的升级开发所有活动，从技术上看，覆盖了从对象、类、组件、模式、架构的所有元素。
6. 与楚凡科技的建模工具结合，最大程度地实现需求、分析、设计、代码空间统一和相互转换。
7. TUP 可以根据企业的需要进行定制和裁剪。

第四节 TUP 的结构

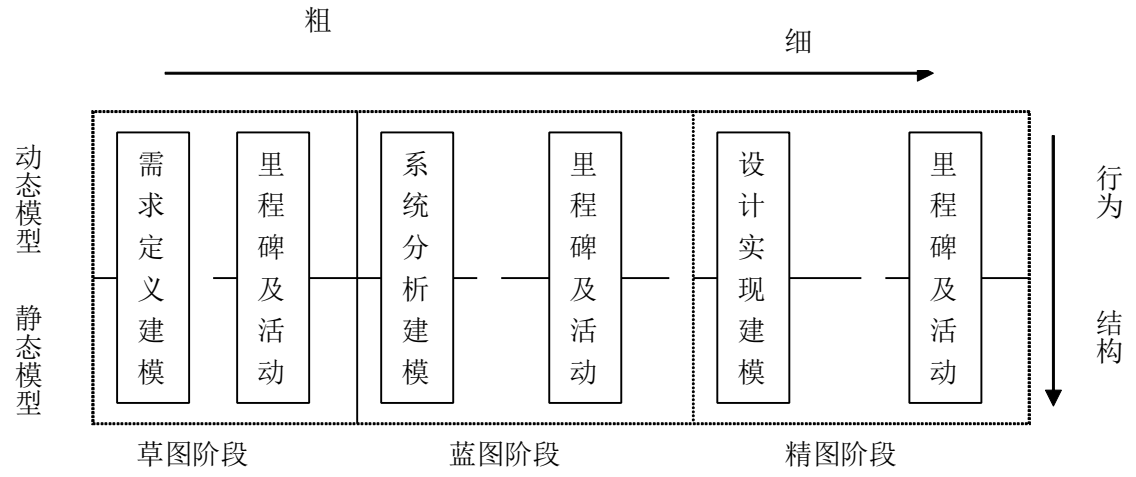


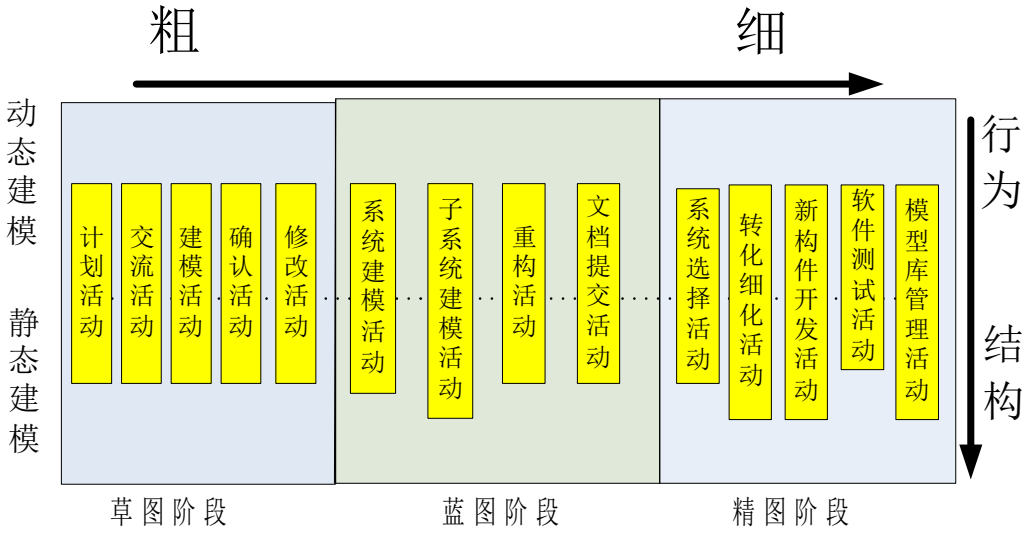
图 18.1 从 OOAD 看 TUP 结构图

图 18.2 TUP 过程实例

图 18.1 表示了 TUP 的结构，很明显，这是一个二维结构图，整个项目的生命周期被划分为四个阶段，即草图阶段，蓝图阶段，精图阶段，上一个阶段的输出是下一个阶段的输入，每个阶段都结束于里程碑。草图阶段的主要任务对需求定义进行建模，采用迭代方式，有一个里程碑和 5 个支持活动。蓝图阶段的的主要任务是对系统和子系统进行建模，有一个里程碑和 4 个支持活动。精图阶段的主要任务是根据所选平台设计和实现蓝图，有一个

里程碑和 5 个支持活动。另外，从草图到精图的过程是一个由粗到精的逻辑推理过程。

从垂直方向看，每个阶段都包括动态建模和静态建模，建模的顺序是从外向内，由表及里，由行为到结构，说的更确切一点，是由用例到支持用例别的类，体现了用例驱动的思想。



18.3. 从软件过程看 TUP 各阶段活动

第五节 TUP 的阶段

TUP 的每个阶段都包括一系列目标、若干重要的焦点活动、若干建模成果以及一个组织活动的方法和一个阶段结束的里程碑。

19.5.1 草图阶段

1. 目标：草图阶段的主要目标是获取和定义用户需求，主要包括以下几个方面：
   - 确定客户高层需求；
   - 确定主要的用户业务需求；
   - 确定软件的范围；
   - 确定软件系统的实现目标；
   - 确定架构雏形。
2. 主要软件过程方法： 草图阶段采用的主要过程方法是迭代法，该阶段分为许多小的迭代，如获取高层需求迭代，获取业务需求迭代等等，因目的不同，可以有多次小迭代，每次迭代都是上一次迭代的深化和细化，迭代的驱动力是发现还有尚未定义的需求。每个小迭代又分为：计划活动、交流活动、建模活动、确认活动，修改活动。
3. 焦点：本阶段的重点是交流活动和建模活动，在建模活动中除了 UML 建模以外，可能还有原型建模。
4. 主要阶段成果：
   - 生成了系统映像；
   - 生成了架构雏形；

- 建立了主要分析类模型；
- 建立了主要流程的活动图模型；
- 建立了系统用例图模型。

5. 里程碑;

确定了软件的范围和目标，并取得客户的认可。

19.5.2 蓝图阶段

1. 目标：蓝图阶段的主要任务是分析用户需求，它于面向对象的概念 OOA 相对应，目标为用户提供一个未来软件系统的业务蓝图，主要包括以下几个方面：
   - 确定系统架构；
   - 确定各子系统结构；
   - 确定支持业务的结构类和类间关系；
   - 确定对象之间的消息；
   - 确定了系统责任边界。
   - 确定支持系统交互的边界类、控制类和持久类。
2. 主要软件过程方法： 蓝图阶段采用的主要过程方法是增量法，主要任务是进行增量建模，增量提交。该阶段分为许多小的增量建模，第 1 个增量是核心建模，它勾画了系统架构和子系统之间的关系，但很多子系统细节尚未确定。可以提交给客户，第二个增量是在客户对第一个增量评估后进行，主要建立各个子系统的分析模型，在这个过程中，如果需求发生变化，则可修改某一增量，然后再发布，此过程不断重复，直到产生了最终的完善业务蓝图,增量模型包括 4 个活动：系统建模活动、子系统建模活动、重构活动、文档提交活动。
3. 焦点：本阶段的重点是系统建模活动和变更建模活动，当然各子系统内部建模也是不能掉以轻心的。
4. 主要阶段成果：
   - 生成了类图；
   - 生成了类关系图；
   - 建立了业务用例图；
   - 建立了顺序图；
   - 建立了通信图。
   - 建立了状态图
5. 里程碑;

确定了软件的系统架构和各子系统结构，具备向代码空间转移的基本条件，并得到客户的认可。

19.5.3 精图阶段

1. 目标：精图阶段的主要任务是设计和实现，它于面向对象的概念 OOD 相对应，目标为用户提供一个可最终发布的软件系统，主要包括以下几个方面：
   - 系统选择，包括系统平台、语言、拓扑结构、体系结构、前后台实现技术、安全、

并发等方面的选择；

- 系统转换，实现业务蓝图向具体语言平台的转换，包括命名、数据类型等；
- 系统细化，由于与语言、平台相关后，需要增加相关的类、消息，需要补充修改业务蓝图，并细化方法契约；
- 查找、选择、测试可复用的组件；
- 设计、开发、测试新的组件。
- 合成、联合编译、系统测试新的系统。
- 组件库更新
- 系统可持续升级优化。

2. 主要软件过程方法： 精图阶段采用的主要过程方法是复用和基于组件的开发方法，复用是该阶段的主要旋律，在前期的设计过程中，首先是要复用系统体系、并发、安全等方面的成熟技术。其次要复用与平台有关的新添类，最后要复用其他项目在由蓝图转换为实现代码上的成功经验；在后期的实现过程中，可以复用其他项目的源码、系统的公共代码、通用组件、领域公共组件等；在最后持续升级优化过程，主要是复用业务模式、通过定制修改复用原来的类、复用框架、模式等。当新的系统需要的组件无法找到时，开发组件成为必然，新组件的开发包括以下步骤：

   （1）确定一个基类或子系统。
   （2）创建一个新的组件单元。
   （3）在新组件中添加接口。
   （4）测试该组件。
   （5）注册该组件。
   （6）为该组件建立帮助文件。

   新开发的组件，一方面要集成到新系统并进行系统测试，另一方面如果是合格的组件，还需要保存到组件库中去。该阶段主要包括 5 个活动：系统选择、转化细化、新构件的设计开发、软件测试、模型库管理。
3. 焦点：本阶段的重点是系统复用查找活动和新组件开发活动，当然系统测试对于发布一个成熟稳定的版本也是十分重要的。
4. 主要阶段成果：
   - 生成了可发布的系统软件；
   - 生成了新的专用组件；
   - 建立了组件图；
   - 建立了部署图；
   - 丰富了模型库；
   - 生成了系统测试报告。
5. 里程碑;

   生成软件的最终代码，具备向客户现场环境部署的条件，并具有持续升级优化的潜能。

第六节 小结

TUP 是一种以模型为中心的全新软件开发过程模型，它的核心思想认为软件的开发过程是构建从需求到最终实现代码的一个个中间演化模型，该模型分为三个阶段：草图、蓝图、精图，它的过程方法是一种组合方法，不同的阶段采用不同的过程方法，草图阶段采用迭代法，蓝图阶段采用增量法，精图阶段采用组件开发法。整个过程共有 14 个活动，若干目标和成果，3 个里程碑。

第七节 习题

1. 是比较 TUP 和 RUP 的差别和各自的优缺点。
2. 试从三个阶段的成果中，找出那些属于动态模型，那些属于静态模型。
3. 其他软件过程模型都是采用以一种方法为主，局部兼顾其他方法的模式，而 TUP 采用三种方法分阶段使用的组合方式，您认为有什么好处？

第十一章 关于楚凡科技

楚凡科技，即西安楚凡(Trufun)科技有限公司。创建于 2004 年 3 月。

楚凡科技是中国专业的软件 CASE 工具厂商，致力于开发中国人自己的优秀 CASE 工具。

欢迎访问楚凡科技官方网站： http://www.trufun.net

如有咨询 UML 工具、数据库建模、MDA 工具及相关 UML 培训事宜，请致电：029-88346234。

版权所有