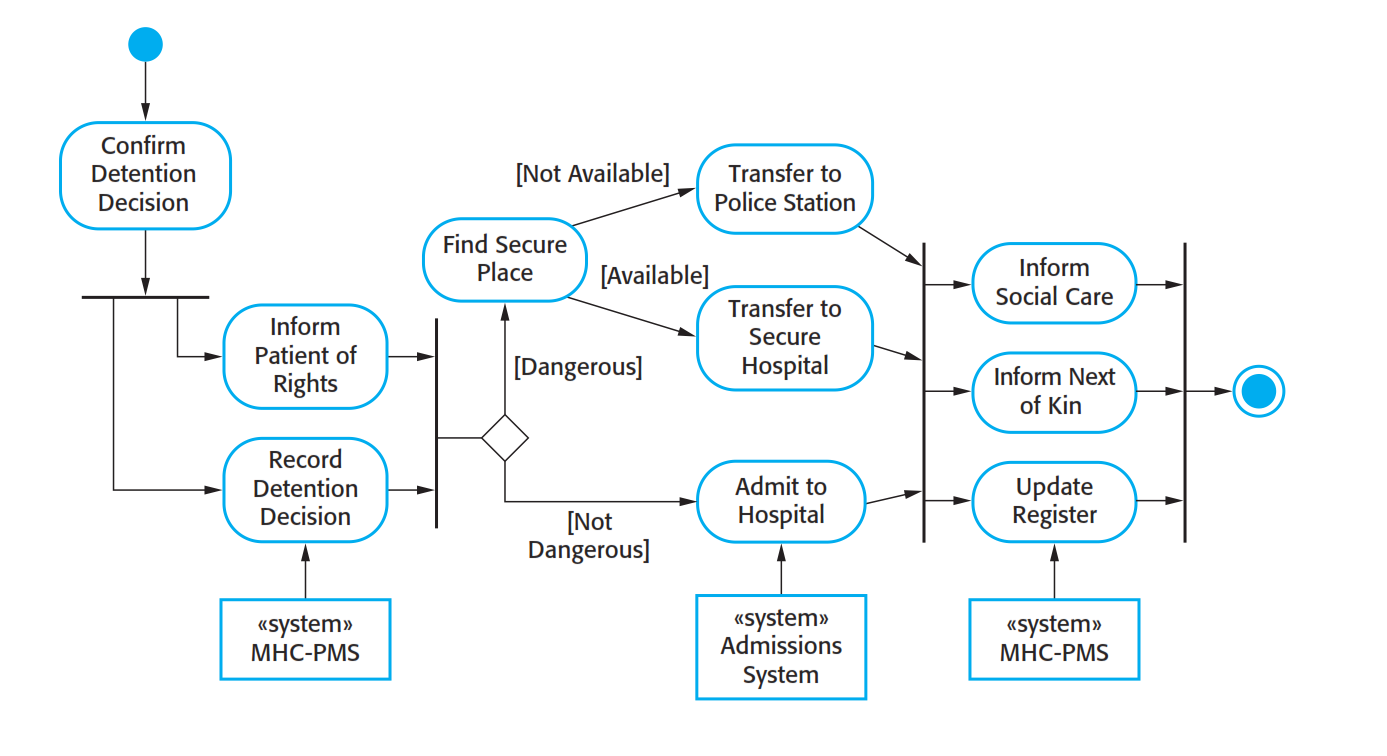
系统建模：（UML）

1. 系统建模就是建立系统抽象模型的过程，每一个模型表示一个系统不同的角度或方面。
2. 在需求过程中使用的模型是为了帮助我们导出系统的需求，在设计过程中使用的模型是为了向现实系统的工程师描述系统，并且实现系统之后还要用模型来论证系统的结构和操作。
3. 在需求工程中要用到现有系统的模型。他们给我们阐述现有系统的功能，并作为讨论系统功能扩展和不足的基础。这些都会导出新系统的需求。
4. 在需求工程中使用新系统的模型有助于解释其他系统信息持有者提出的需求。
5. 系统模型很重要的一个方面就是在于它摆脱了所有的细节内容。系统模型是系统的一个抽象而不是他的一个代替表现。
6. 这些模型可以从不同角度去表述系统从外部来看，它是对系统上下文或系统环境建模。
7. 从交互来看，他是对系统与环境之间或是系统各组成部分之间的交互建模。
8. 从结构来看，他是对系统的体系结构和系统处理的数据的结构建模。
9. 从行为来看，他是对系统动态行为和他对事件的响应方式建模。
10. Krutchen提出的4+1角度
11. UML（Unified Modeling Language）的大多数使用者认为5种类型的图就可以表现出一个系统的本质:
12. 活动图，他表示一个过程或数据处理中所涉及的活动。
13. 用例图，他表示一个系统和他所处的环境之间的交互。
14. 时序图，他表示参与者和系统之间以及系统各部分之间的交互。
15. 类图，他表示系统中的对象类以及这些类之间的联系。
16. 状态图，他表示系统是如何响应内部和外部事件的。
17. 一个模型的具体程度和严格程度取决于我们想怎样来使用系统模型。图形建模通常有3中用途：
18. 为讨论现有系统或提出新功能的系统提供便利。（促进开发系统工程师们的讨论）
19. 论证现有系统。（不必完整，但要正确，可能只是一个模块）
20. 作为详细的系统描述，该描述可用产生系统实现（必须是完整且正确，他们是产生系统源码的基础）
21. 上下文模型



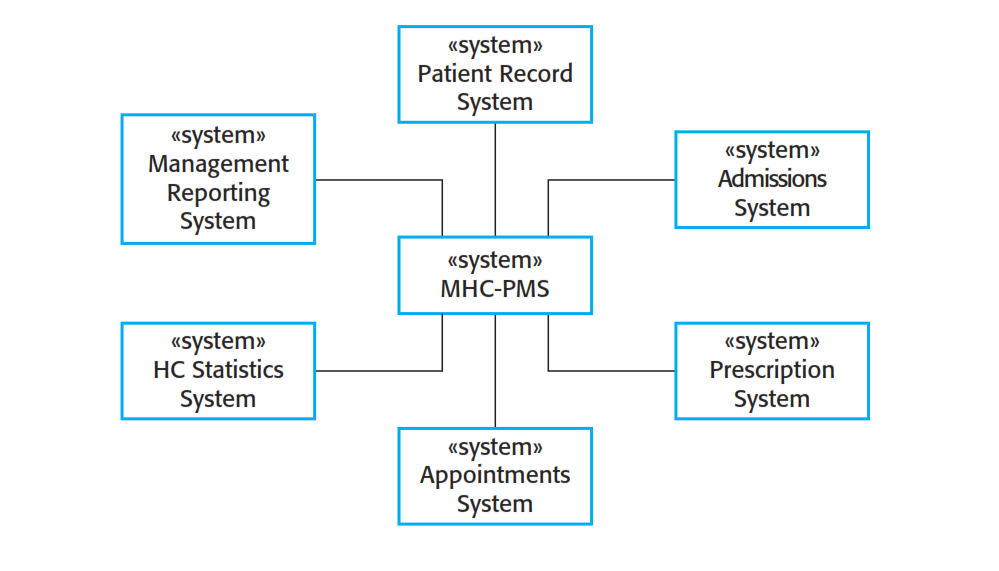
过程模型(UML活动图)

在系统描述的早期阶段，应该首先界定系统的边界，与系统信息持有者一起明确系统应具有什么样的功能以及系统环境提供了什么。由于社会和机构的原因，系统便捷的确定充满了非技术因素，可能需要使系统分析避开与某个难以相处的管理者的沟通。

系统边界一旦确定，接下来的部分分析活动就是定义系统上下文和系统与环境之间的依赖关系，一般来说，第一步是建立一个简单的体系结构模型

系统上下文模型通常表示某一环境包括几个其他的自动系统。然而他们却没有描述其他子系统之间的以及待描述系统与他们之间的关联关系的类型。外部系统可能产生数据提供该系统使用，同时也使用该系统生成的数据。

因此简单的上下文模型要同其他模型结合起来一起使用，比如业务过程模型



系统上下文

UML活动图用来表示组成一个系统过程的活动并控制活动见的控制流，箭头表示各活动间的工作流，实心条表示各活动见的协调关系，当来自多个活动的流导向一个实心条时，那么只有这个活动均完成时过程才能进行，当实心条的流导向多个活动时，这些活动应同时执行。

1. 交互模型：

所有系统都会涉及一些交互，有些是用户交互，与用户输入输出有关，有些是正在开发的系统和其他系统之间的交互。为用户交互建模主要是因为他有助于我们识别用户需求；为系统与系统交互建模重点放在可能产生的交流问题上，为系统各部分之间的交互建模有助于我们分析所提出的系统结构能否实现系统所需的功能及其可靠性。

1. 用例建模，主要用来为系统与外部参与者（用户和其他系统）之间的交互建模。
2. 时序图，用来为系统各部分之间的交互建模，尽管也包括一些外部因素。

用例模型和时序图在不同细节程度上表示交互，因此可能一起使用，高水平用例所表示的交互细节可能在时序图中得到刻画。

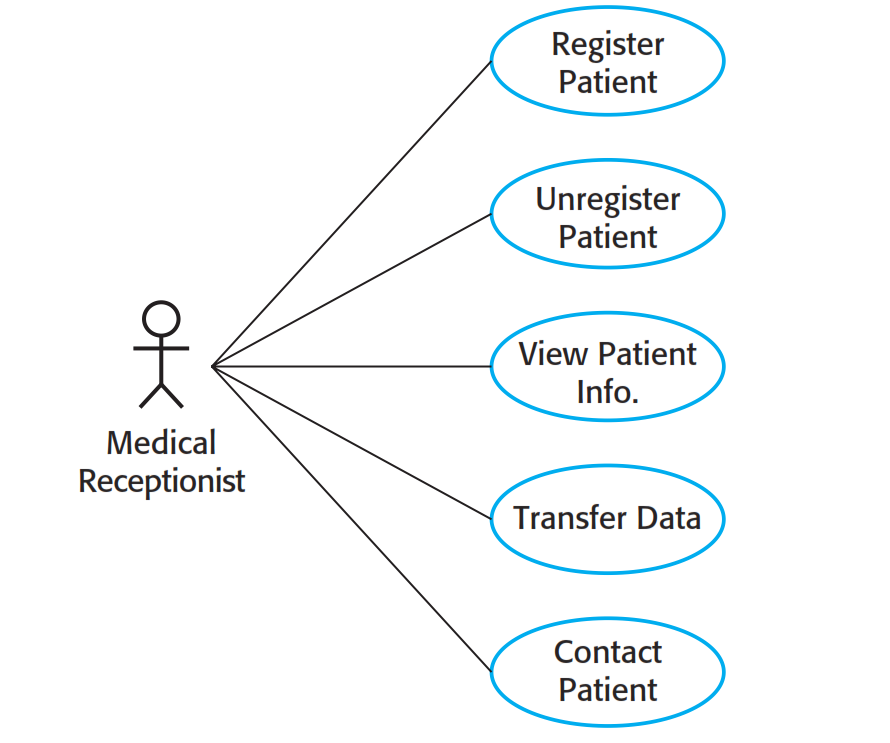
1. 用例建模被广泛用于支持需求导出。一个用例可以被当作一个简单的脚本（情景），用来描述用户对系统功能的期望。每一个用例表示一个具体的任务，涉及与系统的交互。用例用一个椭圆表示，参与者用一个小人表示。

不带箭头表示信息双向传递，带箭头表示流向

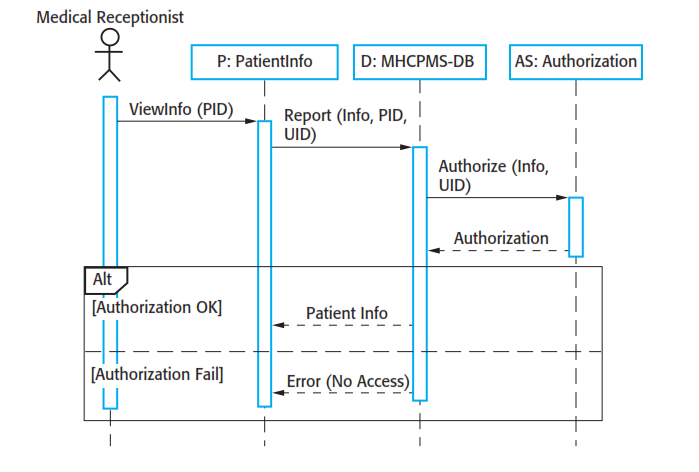


用例图所给出的只是交互非常简单的概要，所以我们必须提供更多的细节才能理解交互的内容，这些细节或者是一个简单的文本描述，或者是一个表格形式的结构化描述

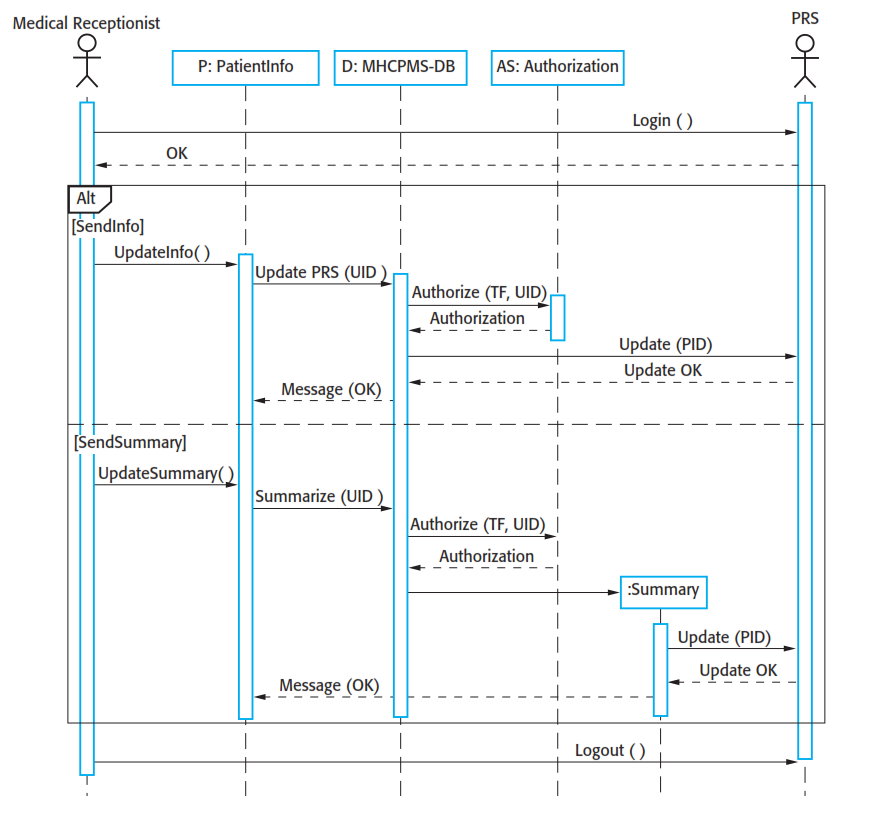
复合用例可以表示许多不同的用例。



1. 时序图最初是用来为参与者与系统对象之间的交互和对象之间的交互建模的。语法非常丰富。时序图表示在特定用例中的交互发生顺序



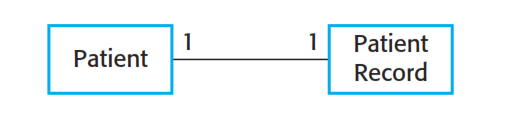
涉及的对象和参与者列在图表顶端，向下垂直划一条虚线。对象之间的交互用带注释的箭头表示。从上到下为交互的顺序。箭头上的注释表示对对象的调用、他们的参数、返回值。Alt的框表示的条件下使用。



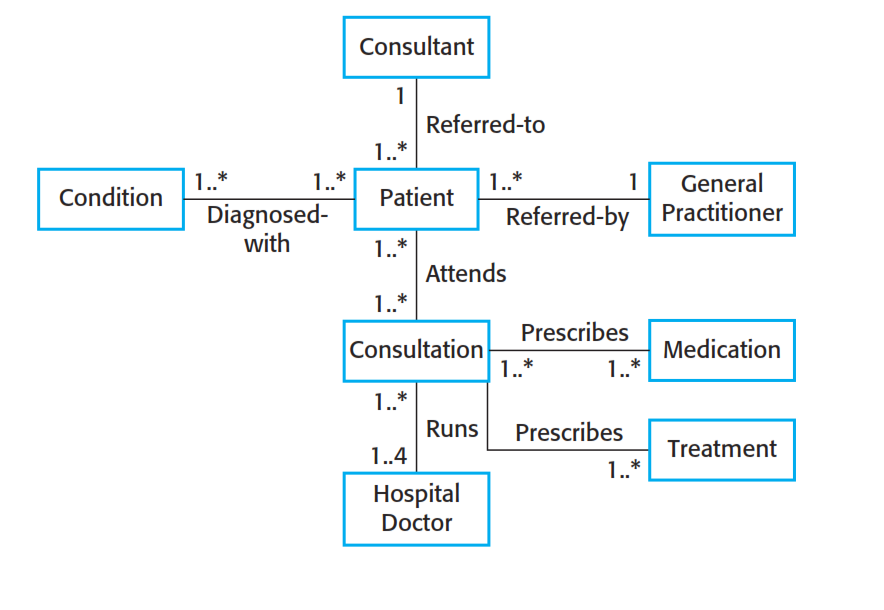
创建了一个Summary类的对象来存储即将上传给病人信息记录系统的摘要数据。

1. 结构模型软件的结构模型表示的是系统的构成，表示为组件构成系统以及组件之间的关系。有的结构模型是静态模型，表示系统设计的结构；有的是动态模型（是交互线程的集合），表示系统执行时的结构。
2. 类图
3. 泛化
4. 聚合
5. 类图：当开发面向对象的系统模型时，我们用类图来表示系统中的类和这些类之间的关联。所谓的关联就是类和类之间的链接，表示类与类之间具有某种关系。

在软件工程过程的前期阶段开发模型时，我们用对象来表示现实世界的事物。在系统实现阶段，我们需要定义额外的实现对象，他们用来提供所需要的系统功能。这里强调的是对现实世界中的对象的建模。

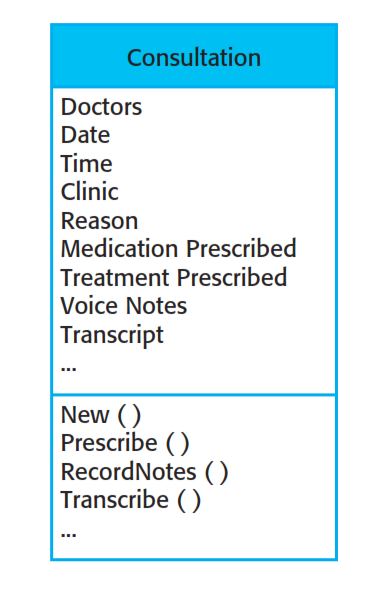


类的另一个特征就是能够表现出有多少个对象参与这种关联当中。\*表示联系中所涉及的对象数目不确定。



在这个细节层次上的类图看起来像是语义数据模型。语义数据模型在数据库设计时使用，他们表示数据实体，与他们相关的属性，以及实体之间的关系。

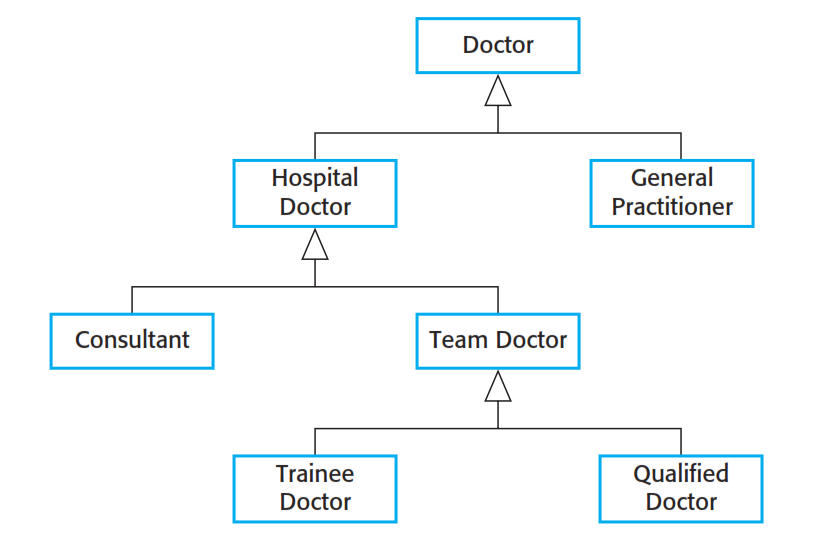
我们可以把语义数据模型中的实体看做是简化了的对象类，把属性看做是对象类属性，把关系看做是对象类间的命名的关联。



属性和操作。

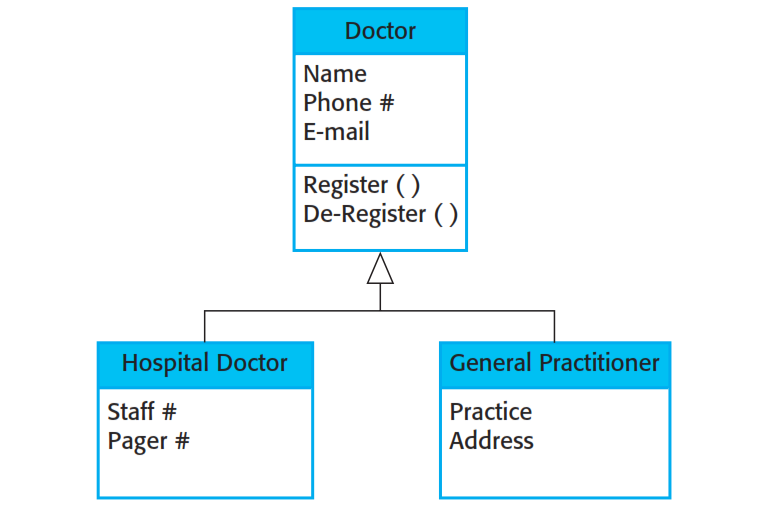
1. 泛化：我们用这项技术来管理复杂性。我们可以针对类的所有成员给出一般性的描述。

在系统建模中，检查系统中的类。看看它们是否还有继续泛化的空间，这通常是有用的。这意味着共同的信息仅保持在一个位置。这是一个很好的设计习惯，因为当发生变更时，我们就可以不必观察系统中的所有类看它们是否受到变更的影响。（JAVA，泛化是使用内嵌在语言中的类继承机制实现的）

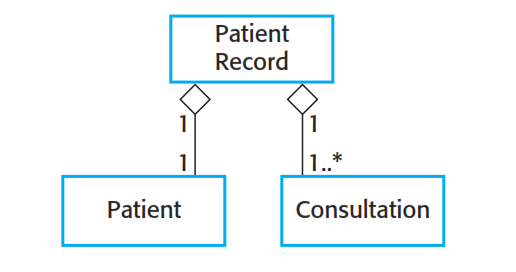


泛化用指向上面更一般的类的箭头表示。

在泛化关系中，更高层次上的类中的属性和操作也是较低层次上的类的属性和操作。



1. 聚合：现实世界中的对象通常是由不同部分组成。它意味着一个对象（相对与全体）可以由其他对象（相当于部分）组成。我们用与类相邻的菱形表示全体。



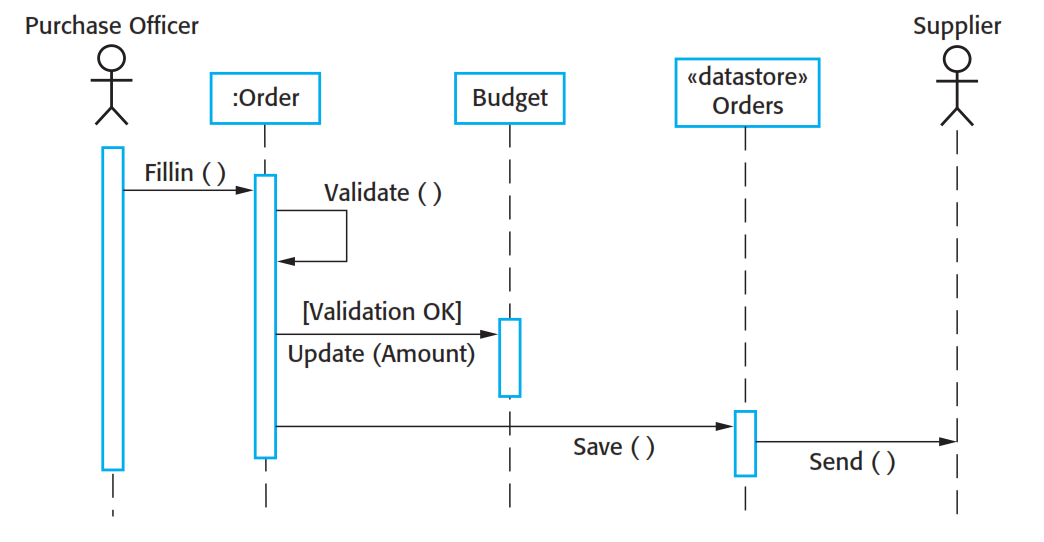
1. 行为模型，行为模型是描述系统运行时的动态行为的模型，表示当系统响应来自所处环境的刺激时所发生的或有可能发生的事情。这样的刺激有两种：
2. 数据 一些数据到达必须由系统处理。（数据流图DFD是个系统模型，它是从功能角度描述系统，每一个变换代表一个单独的函数或过程。）
3. 事件 某些触发系统处理的事件发生。
4. 许多业务系统是主要有数据驱动的数据处理系统，他们由输入数据所控制，很少处理外部事件他们的处理包含一系列在那些数据上的动作以及输出的生成。

相对而言，实时系统主要处理事件而很少涉及数据处理。

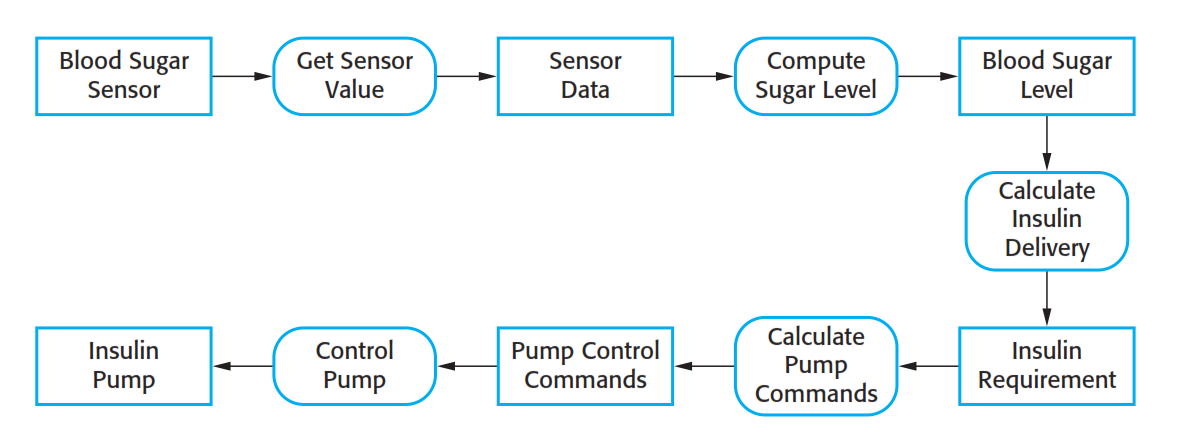
1. 数据驱动建模，数据驱动模型描述一个动作序列，该动作序列涉及输入数据的处理和相关输出的产生。（一个完整的动作序列）

数据流图使得结构化分析更清晰，他跟踪并记录了与一个特别过程相关的数据是如何通过这个系统的。

UML一开始没有数据流图，因为DFD关注与系统的功能而不是识别系统中的对象。



UML中，时序图可以表示系统处理序列的方法。这里，顺序模型强调的是系统中的对象，而数据流通关注的是功能。

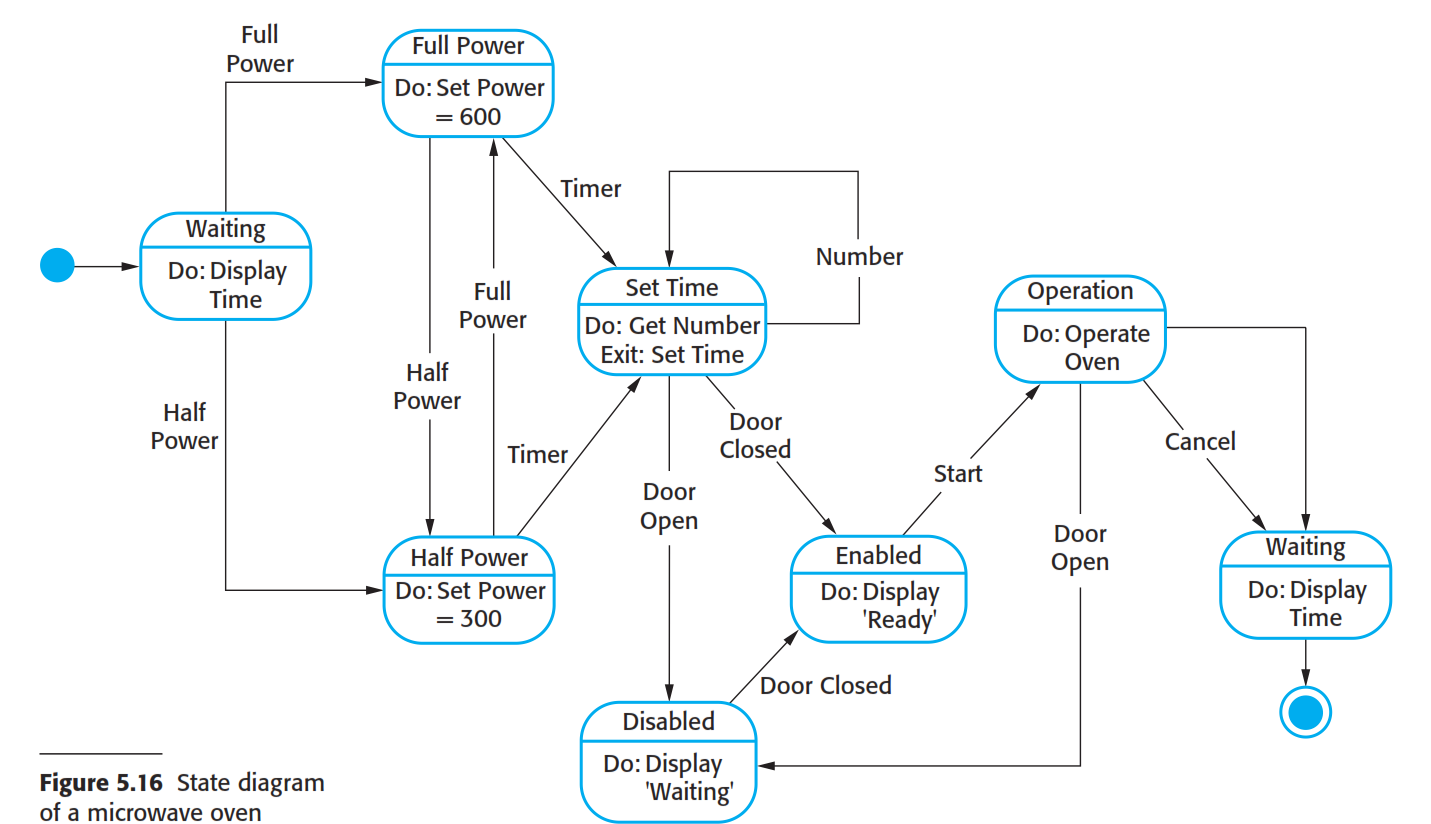


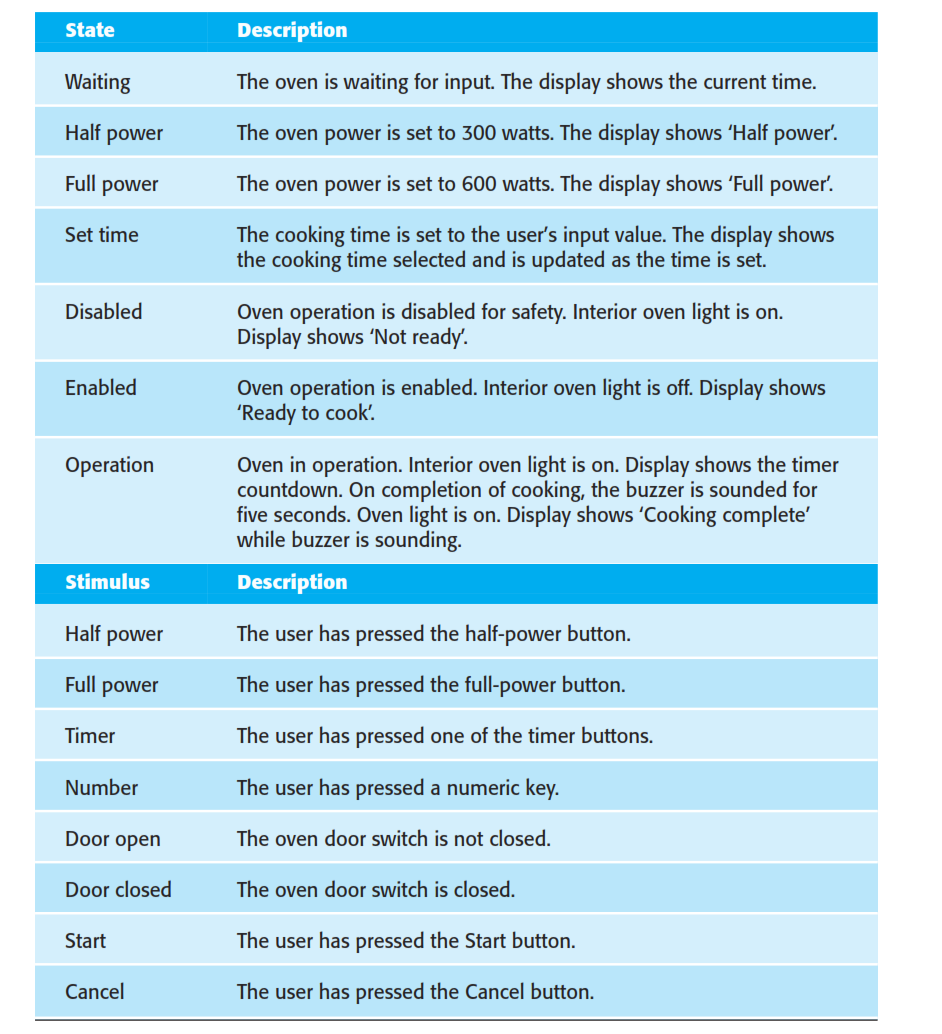
1. 事件驱动模型，事件驱动模型表示系统对内外部事件的相应方式。下面的讨论是以一个假设为基础的，即系统状态是有限的，并且事件（激励）可能引起从一种状态向另一种状态的转变。这种系统观点特别适合于实时系统。

UML通过使用状态图支持基于事件的模型。状态图表示系统状态和引起状态改变的事件。状态图不表示系统中的数据流，但可能包括在每一状态中所执行运算的附加信息。

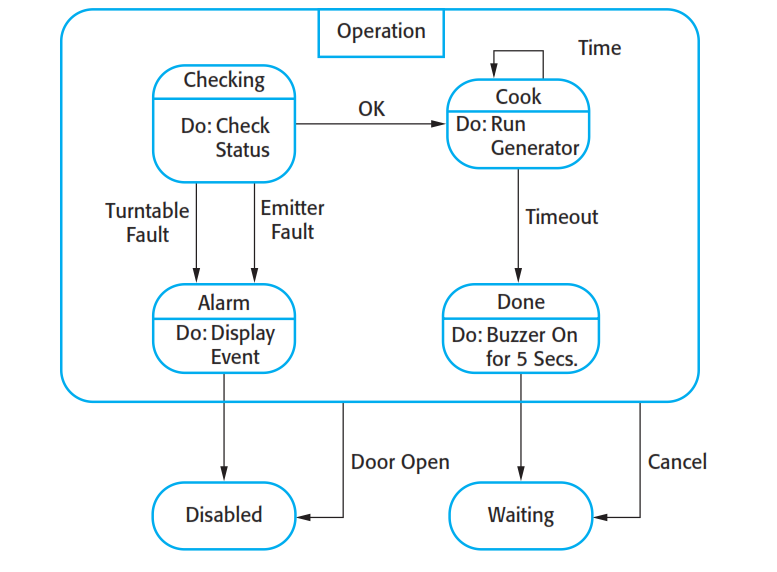
在UML中，圆边矩形表示系统状态，其中可能包括此状态中所执行动作的简单描述（由“DO”引出）。带标签的箭头代表促使系统从一种状态变为另一种状态的激励因素。

我们必须提供更多有关激励因素和系统状态方面的细节。





以状态为基础的建模遇到的问题是状态的数量可能急剧增加。对于大的系统模型，我们需要在模型中隐藏一些细节。其中一种方法就是使用超态的概念，超态是封装了多个独立的状态得到的一种状态，这种超态看起来像一个高层模型中独立的一个状态，但是之后会在一个独立图中被扩展来表示更多的细节。



1. 模型驱动工程

模型驱动工程（MDE）是软件开发的一种方法，在这种方法中，模型而不是程序成为开发过程中的主要输出。

模型驱动工程（MDE）起源于模型驱动体系结构（MDA）。MDA关心的是软件开发中的设计和实现阶段，而MDE关系软件工程的所有方面。

支持者认为模型为基础的工程允许工程师在更高的抽象水平上考虑系统，而不用关心他们实现的细节。这样减少了出错的可能性，加速了设计和实现过程，并且支持可复用性的、具有独立平台的应用模型的建立。（同一模型在不同平台上实现）。

反对者认为模型是一种很好的便于讨论软件设计的方法。对于不同的系统，模型支持的抽象不一定是正确的，能够实现的抽象。而对于一些大的长生命周期的系统更实用，但是这些系统面对的主要问题不是实现，而是需求工程，信息安全性和可依赖性，与遗留系统的集成和测试。

现在并不清楚模型驱动方法的成本和风险是否会超过他给我们带来的便利。

1. 模型驱动体系结构 是一种以模型为中心进行软件设计和实现的方法。MDA方法认为应该产生3种类型的抽象系统模型
2. 计算独立模型（CIM）
3. 平台独立模型（PIM）
4. 平台特定模型（PSM）
5. 可执行UML

模型驱动工程背后的基本概念是，模型向代码的完全自动化转化是可行的。

UML是一种支持软件设计和记录软件设计的语言，而不是编程语言。为了建立一个UML可执行子集，模型类型的数量已经被剧减为3个主要的模型类型：

1. 领域模型，识别出系统的主要关注点，然后由UML的类图定义，其中包括对象、属性、关联。
2. 类模型，在类模型中定义类以及他们的属性和操作。
3. 状态模型，状态模型中的每一个状态图都与一个类