返回总目录

目　　录

第 7 章　UML 的扩展

为避免增加 UML 语言整体的复杂性，UML 并没有吸收所有面向对象的建模技术和机制，而是给 UML 设计了扩展机制，使它很容易适应某些特定的方法、机构或用户。通过扩展机制，用户可以定义和使用自己的元素，也就是说，CASE 工具必须同时支持语言原有的元素和用户定义的扩展。

扩展的基础是 UML 的元素，然后给这些元素的一些变形加上新的语义。新的语义可以有三种形式：重新定义，增加新的或者对某种元素的使用增加一些限制。UML 有三种扩展其核心的机制，这些机制也定义了一些标准扩展以及某些元素的变形。本章将介绍所有标准的 UML 扩展，并说明用户自定义机制将如何扩展 UML。

UML 的三种扩展机制是：加标签值（性质），约束和版类：

- 加标签值是附属于 UML 元素的性质，类中的操作所附加的前置条件和后置条件是一种标准的加标签值。
- 约束是 UML 中限制一种或多个元素语义的规则（如图 7-1）。约束可以附加在类或对象上，并且经常附加在关系上，约束参与关系的类或对象。

```
<<Stereotype>>
Name
(Constraint-String, Keyword=Value, …)
```

图 7-1　约束字符串和关键字－值对

- 版类是最复杂的扩展机制（如图 7-2）。它是一种附加在已有模型元素的语义，如果版类附加于某种元素，则覆盖该元素的语义，该元素就成为一种新的元素。典型的版类如为类定义的元类（metaclass），为包之间的相关关系定义的导入（import）等。在这种情形中，版类在原来的元素（类或相关关系）中增加了新的或另外的语义。

```
<<Stereotype>>
Name
(Properties)
```

图 7-2　版类

有许多原因需要对 UML 进行扩展，例如，如果所使用的方法可能会有些概念 UML 不能直接支持的，但是扩展 UML 就能够表达；或者应用领域或机构可能有些重要的普通概念需要定义在建模语言中；又或者建模人员想通过扩展 UML 来定义一些更为精确和清晰的模型。

7.1　UML 的架构

UML 的定义是在一个建模概念框架下定义 UML 的，如图 7-3 所示，由以下四个不同的抽象层次或级别组成：

- 元元模型层由最基本的元素组成，它们是 UML 的基础——“事物”的概念，表

示任何可以被定义的事物。这一抽象级是用来形式化概念的表示，并为指定元模型定义语言。

- 元模型层包括所有组成 UML 的元素，其中有来自面向对象的概念和面向组件的范例。这一层的每个概念都是元元模型概念“事物”的实例。这一抽象层是用来形式化范例概念、并为指定模式定义语言的。
- 模型层由 UML 模型组成。这一层为问题、解决方案或系统建模。这一级的每个概念都是元模型层的概念的实例。这一抽象层是用来形式化概念、并根据一给定个体定义表达沟通的语言。这一层的模型通常叫做类或类型模型。
- 用户模型层由 UML 模型的例子组成。这一级别的每个实例都是模型层和元模型层概念的实例。这一抽象级别用来根据特定的个体形式化特定的表达，这一抽象级的模型通常叫做对象或实例模型。

在这一框架内，用“元”的概念表示在一组非元概念以及它们的元概念之间的关系。“元”概念不是模型的一个性质，而是模型在与其它模型相关时所扮演的角色：元元模型与元模型之间的关系与元模型与模型之间的关系是一样的。

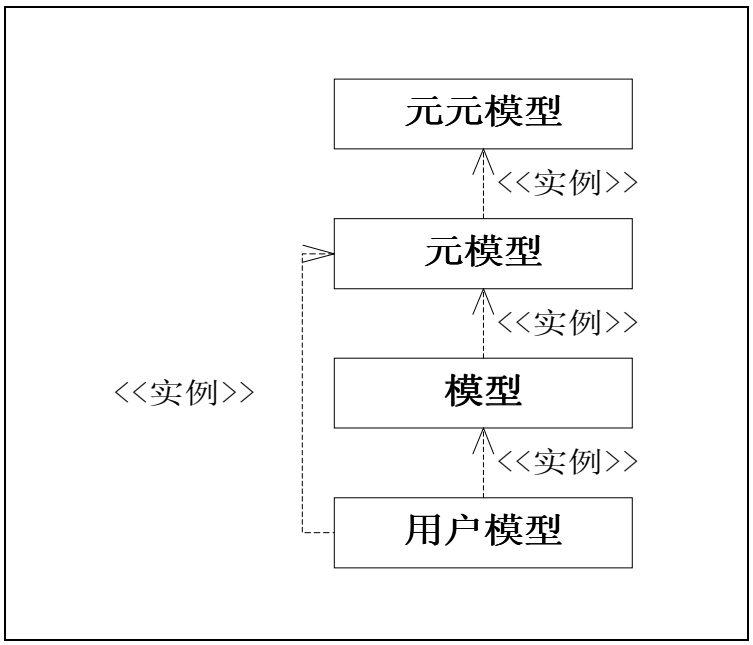

图 7-3　建模的概念框架

7.2　UML 的核心语义

要实现用户自定义扩展，必须熟悉 UML 的语义，至少是它基本的核心语义。UML 扩展机制的形式语法和语义都定义得非常仔细，可以扩展 UML 自身。UML 参考手册的“表示指南”定义了语法，元模型定义了语义。元模型描述模型，而 UML 参考手册中的元模型则描述了 UML，并且参考手册中的元模型用的也是 UML 的表示方法。也就是说，

UML 本身也是用 UML 模型描述的。

这一部分将简单地描述 UML 参考手册中给出的形式语义，在定义自己的扩展之前，再次建议大家了解一下基本的 UML 核心语义，这将有助于对 UML 底层模型的理解。

元素是 UML 大多数成分的抽象基类，它是一个基础，在此之上可以附加一些其它机制。元素被专有化为模型元素、视图元素、系统和模型。模型元素是被建模系统的一个抽象，如类、消息、节点、事件等。视图元素是一个映射——单个模型元素或一组模型元素的文字或图形映射。视图元素是文字或图形符号，如矩形代表类。这就是说，模型元素是概念，而视图元素则是用来构建模型（图）的符号。视图元素也被专有化为图，它们是用例图、组件图、类图、展开图、状态图、活动图、协作图以及对象图，所有元素都可以有名字。UML 是用 UML 来描述的，因此 UML 中的所有概念都被定义成类。例如，相关的定义是一个类，具有一些描述一个“相关”是什么的属性。

包“拥有”元素并且引用它们，如图 7-4 所示。包是一种组合机制，可以拥有或引用元素（或其它包）。包中的元素可以有多种，如模型元素、视图元素、模型和系统。因此包是 UML 是最一般的组合机制。

模型元素被专有化后对系统建模非常有用。大多数元素都有相对应的视图元素来表示它们。但是，某些模型元素就没有相应的表达元素。如模型元素的行为，就无法在模型中可视化地描述。图 7-5 显示了如何把模型元素专有化为 UML 所使用的建模概念。模型元素被专有化为以下子类：

- 类型：一组具有相同的操作、抽象属性和关系以及语义的实例的一个描述。类型被专有化为原始类型、类和用例。类被专有化为活动类、信号、组件和节点。所有类型子类都有一个相应的视图元素。
- 实例：一种类型所描述的一个单个成员。类的实例（类型的子类）是对象，对象与实例类似（但通常只是类的实例）。
- 行为实例：行为的实例。
- 笔记：附加在一个元素或一组元素的一条注释。笔记没有语义，模型元素的笔记在相应的视图元素。
- 值：类型定义域里的一个元素。类型定义域是某个类型的定义域，如 42 属于整数的类型定义域。

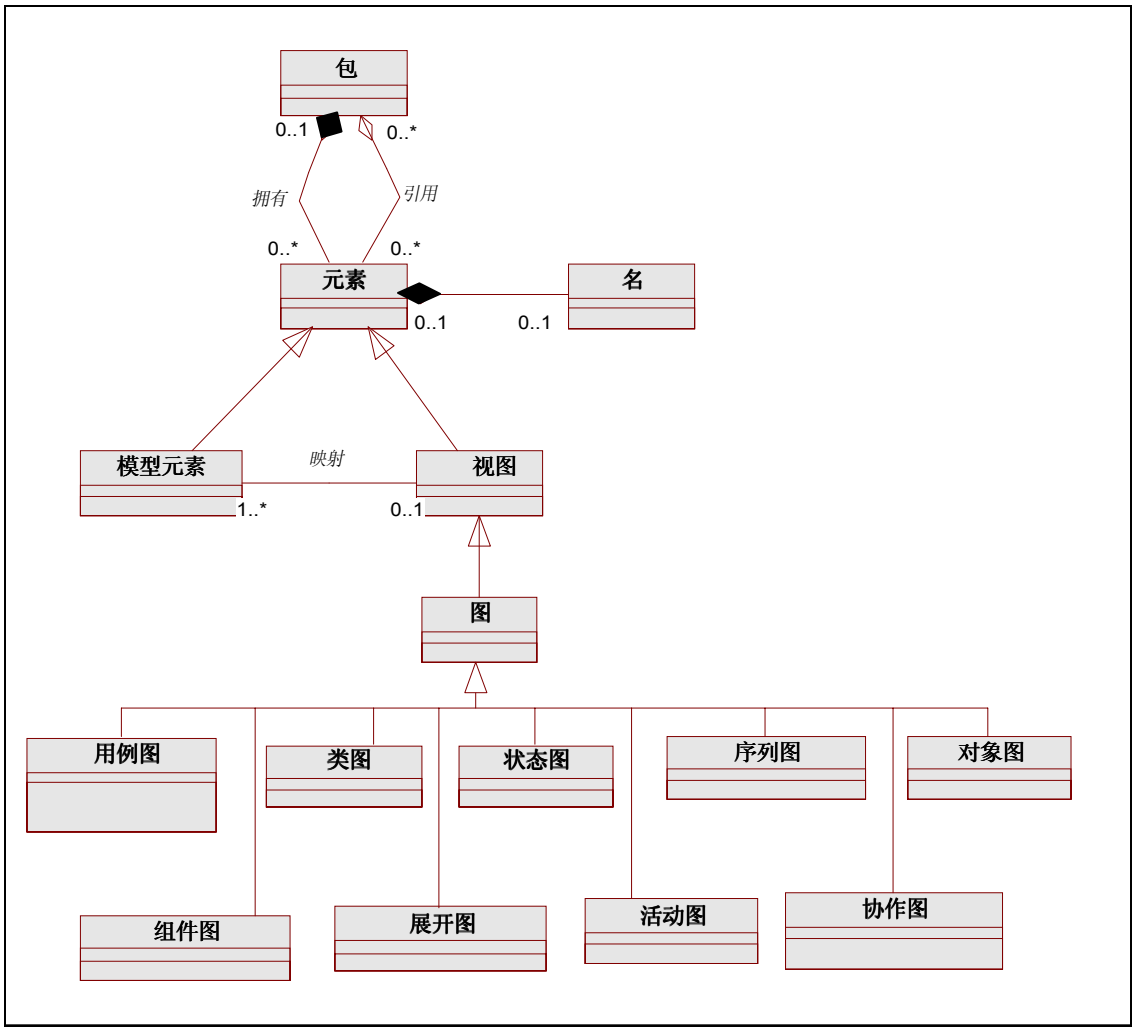

图 7-4　一个包拥有和／或引用元素，元素可以是模型元素或视图元素（或者其它元素如系统和模型）。视图元素是模型元素的映射，视图元素映射一个或一组模型元素。视图元素还被专有化为九种不同的图，这几种图是一组模型元素的映射

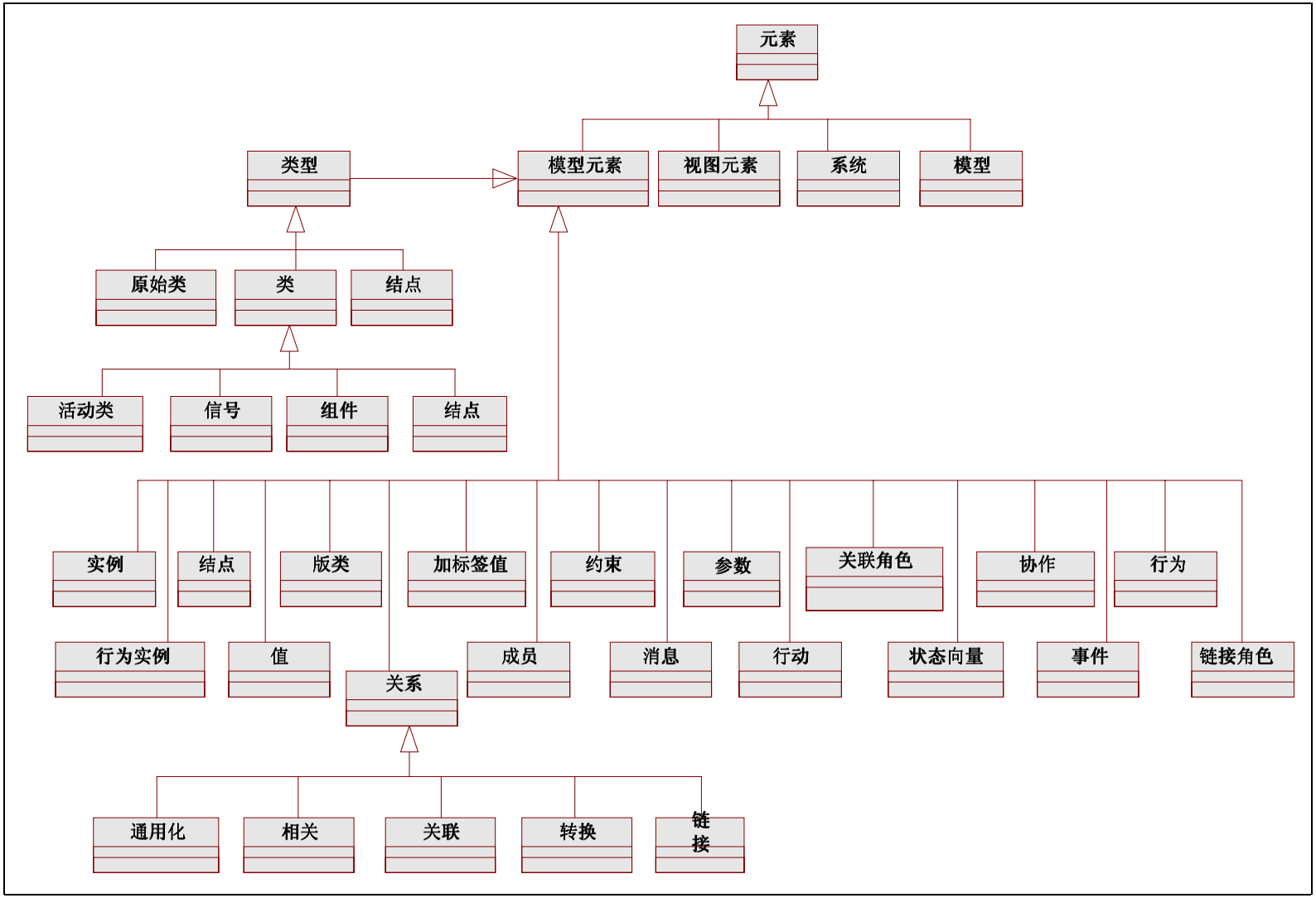
元素
类型
模型元素
视图元素
系统
模型
原始类
类
结点
活动类
信号
组件
结点
实例
结点
版类
加标签值
约束
参数
关联角色
协作
行为
行为实例
值
关系
成员
消息
行动
状态向量
事件
链接角色
通用化
相关
关联
转换
链接

图 7-5　抽象类“元素”被专有化为模型元素、视图元素、系统和模型

- 版类：建模元素的一种类型，扩展 UML 的语义。版类必须以 UML 中已经定义的元素为基础，可以扩展语义，但不能扩展已存在元素的结构。UML 预定义了一些版类，其它是用户定义的。版类有相应的视图元素《版类名》。
- 关系：模型元素之间的一种语义连接。关系被专有化为通用化、相关性、关联、转移和链接。通用化是更通用元素和更专有元素之间的一种关系。专有元素与通用元素完全一致并且还包含其它信息，在所有使用更通用元素的实例的场合，都可以使用更专有化元素的实例。相关性是两个模型元素之间的一种关系，对独立的模型元素的改变将影响相关模型元素。关联是描述一组链接的一种关系。链接是对象组之间的一种语义连接。转移是两个状态之间的一种关系，表示当劳动某件指定的事件发生并且某些指定的条件满足时，处在第一种状态的对象将完成某些指定的动作并进入第二种状态。对于每种关系都有一个对应的视图元素。
- 加标签值：把性质明确定义为一个名-值对。加标签值中，把名称为标签，UML 预定义了一些标签。相应的视图元素是一个性质表。在 UML 中，性质普遍用于与元素有关的任意值，包括类的属性、关联和加标签值。
- 成员：类型或类的一部分，表示一个属性或操作。

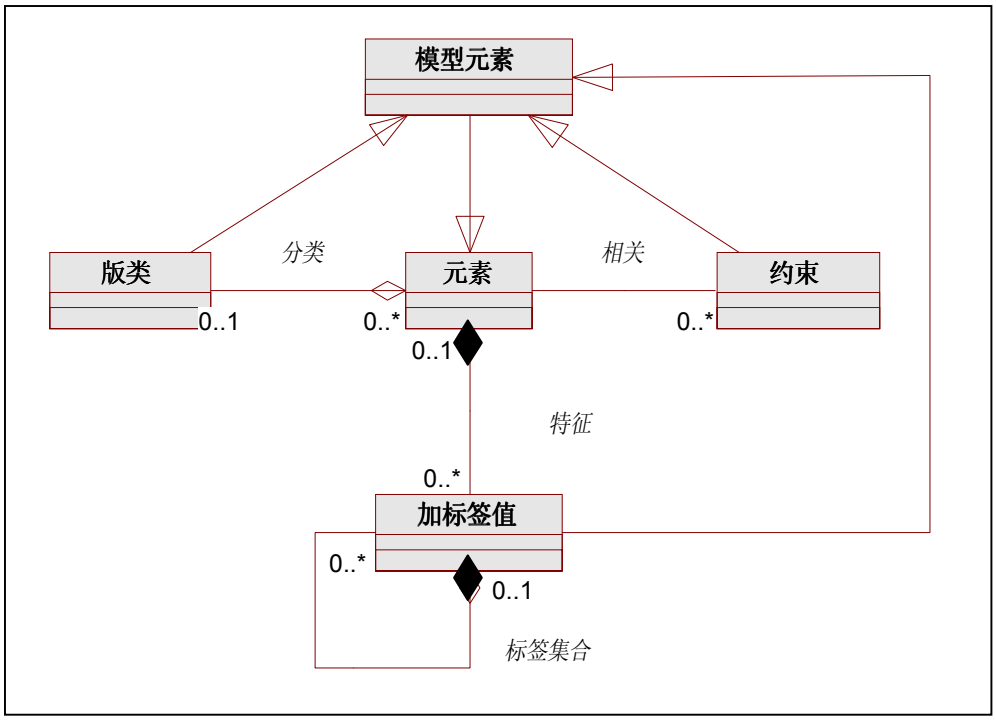

图 7-6　元素可以有 0 个或 1 个版类，多个约束以及多个加标签值。加标签值可以是一组加标签值。版类、约束和加标签值都是模型元素，也就是说它们都是元素

- 约束：一种语义条件或限制。UML 预定义了一些约束，约束也有对应的视图元素。
- 消息：对象之间的一种通信，传递有关将要进行的活动的信息。通常认为接收消

息是一个事件，不同的消息有相对应的视图元素。

- 参数：变量的规格说明，可以被修改、传递或返回。参数可能包含名、类型和指针。在操作、消息和事件中使用参数。
- 动作：是对信号或操作的调用，代表一个计算或算法过程。
- 关联角色：类型或类在关联中所扮演的角色，有对应的视图元素。
- 状态顶点：转移的源或目标。
- 协作：支持一组交互的环境。交互是一个规格说明，它的组成是一组对象为完成某个特定任务而在某个特定环境下的一组消息交换。
- 事件：时间或空间的一个显著发生。事件有对应的视图元素。
- 行为：行为是一个可见的作用及其结果。
- 链接角色：关联角色的实例。

所有元素都可能与其它元素有相关关系，所有元素的子类，包括子类版类、约束和加标签值等也都将继承这种相关关系。元素具有 0 个或一个版类，0 个或多个加标签值，对约束有一种派生相关关系。元素和约束之间的派生相关关系的重数在元素一端为 1，在约束端可以是 0 个以上（见图 7-6）。

并不是所有的元素都可以被专有化或通用化，只有可通用化的元素才可以被专有化或通用化。可通用化的元素包括版类、包和类型。可通用化元素的子类也是可通用化的。类型有原始类型、类和用例等子类，类有活动类、信号、组件和节点等子类，因此这些类都是被专有化或通用化的。注意版类可以被专有化（见图 7-7）。

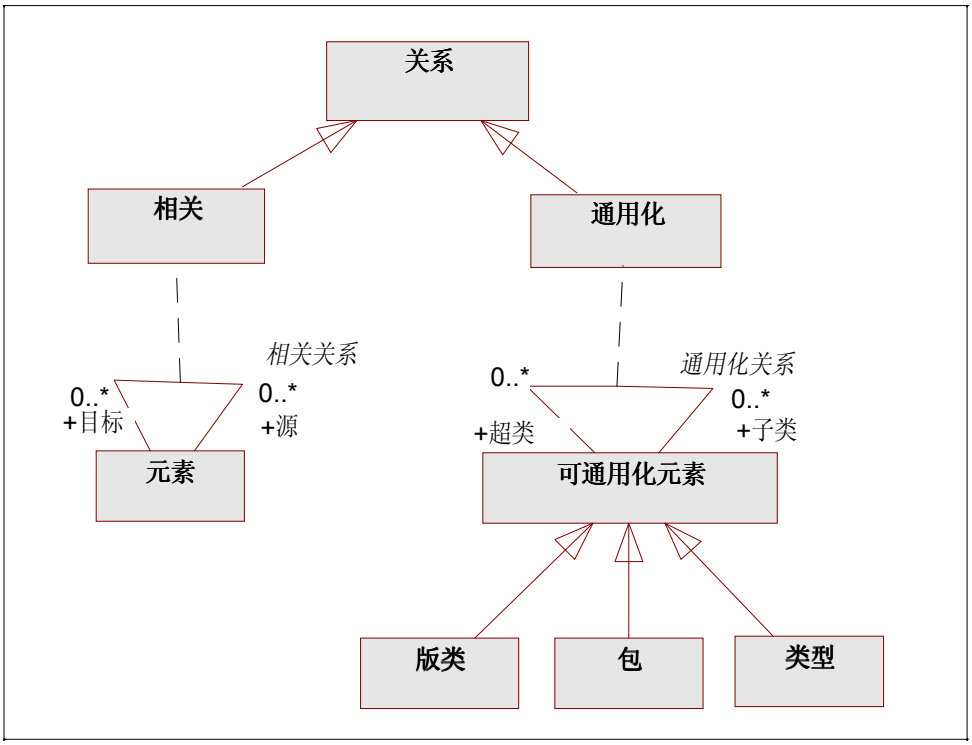

图 7-7　UML 中的两种关系之一是相关性和通用化。相关关系可以连接两种模型元素，但通用化关系只能

连接可通用化元素，它们是版类、包和类型（以及它们的子类）。其它关系是关联、转移和链接。关联只能连接类型，链接只能连接类型的实例，转移只能连接状态（顶点）

有时对关联进行专有化和通用化非常有用，但 UML 不允许这样，因为关联不是一个可通用化的元素。但两个以上关联之间的约束的子集可以替代这个作用。

如果给一个模型元素附加了加标签值、版类或约束等，则该模型元素的专有化将具有相同的加标签值、版类或约束。如果一个通用化有一个版类，则对该通用化的专有化就不能再有其它版类了，元素只能有 0 个或 1 个版类，这其中包括所有继承的版类。

7.2.1 核心语义小结

元素是 UML 大多数万分的抽象基本类，它是许多机制应用的基础。元素被专有化为模型元素、视图元素、系统和模型，模型元素是被建模系统的一个抽象。一些重要的模型元素包括类型、关系、版类、约束、加标签值、状态顶点、事件和消息。类型和关系模型元素都被专有化到其它子类。视图元素是模型的一个（文字或图形的）映射。一些单个模型元素有对应的视图元素（符号），如类、节点、状态和关联，也有视图元素是与一组模型元素对应的，称为图。UML 中的几种图是：用例图、组件图、类图、展开图、状态图、活动图、序列图、协作图和对象图。

元素可以组合到包内，一个包也可以包含其它包。版类、包和类型（原始类型、类、活动类、信号、组件、节点和用例）都是可通用化元素，也就是说，可以对它们进行专有化和通用化。

版类、约束和加标签值是用来扩展 UML 的机制。UML 预定义了一些扩展机制，用来修改或扩展已有元素。一种元素可以有 0 个或 1 个版类、多个约束和加标签值。如果把一个加标签值、版类或约束附加到某个元素上，则该的专有化也将具有相同的加标签值、版类或约束。如果一个通用化有版类，则对它的专有化将不能再有其它版类。

7.3 加标签值和性质

性质通常用于表示元素的值，增加模型元素的有关信息，机器和人都可以使用。人可以用它来增加模型的管理信息，例如，可以在给模型加上最后一次修改的 5 时间等。机器通过这些信息可以某种方式处理模型，例如，模型中的性质可以是代码生成的参数，告诉代码生成器生成何种类型的代码。注意在最后提交的系统中通常没有性质，因为它们只是有关模型以及模型内元素的信息，最终系统并不处理它们。

加标签值把性质明确地定义成一个名-值对，其中，把名称为标签。每个标签代表一种性质，并且能够应用于一个以上的元素。标签和值都用字符串表示，性质的表示如下：

```
{tag = value} or {tag_1 = val1, tag_2 = val2} or {tag}
```

性质都是用大括号括起来，一个标签和一个值对应。如果标签是个布尔标记，此时如果省略了值，则默认取真。除布尔类型外的其它类型的标签都必须明确写出值，值没有语法限制，可以用任何没有意义的符号表示（常常是个字符串）。标签名定义了一个上下文，给出加标签值的意义。例如：

```
{abstract}
{status = "under construction", system analyst = "Bob Jason"}
```

性质（包括加标签值）可以出现在图中，也可以存放在单独的文档中。在图中，性质由大括号括起来，离它们被定义的元素很近。一些 CASE 工具有个性质窗口，可以显示某个元素的所有性质以及它们的值。UML 为元素、模型元素、实例、操作、属性和组件预定义了一些加标签值（标准加标签值）。另外，使用 UML 的工具或方法可以添加自己的加标签值。

7.3.1 元素的加标签值

文献是给元素实例进行建档的标签，值只是个字符串。通常这个加标签值是单独显示的（而不是与元素放在一起），例如，在某些工具中，该值是显示在一个性质或文献窗口内的。类 Insurance Contract 附加的文献标签的值可以是对该类意图的描述：

```
This class serves as a contract between the customer
(policyholder) and the insurer (insurance company).
```

7.3.2 类型、实例、操作和属性操作和属性的加标签值

有九种加标签值，可应用于类型、实例、操作和属性。它们是：

- 不变性（invariant）
- 后置条件（postcondition）
- 前置（precondition）
- 责任（responsibility）
- 抽象（abstract）
- 持久性（persistence）
- 语义（semantics）
- 空间（space semantics）
- 时间语义（time semanntics）
- 不变性应用于类型，它指定了类型实例在整个生命周期中必须保持的一种性质。这个性质通常对于该类型实例必须有效的一种条件。有些语言如 Eiffel 都嵌入对不变性支持，并在最终的代码中直接实现。不解释该值的语法，通常也不在图中显示。

类 Car 中的颜色属性附加的不变性加标签值的值可以是：

```
The value of the attribute color may not change during the lifetime
of the object.
```

更形式化的不变性标签可以是：

```
Car.Speed >= (meaning that the speed may never be negative)
```

- 后置条件应用于操作，它是在操作结束后必须为真的一个条件，该值也没有解释，并且通常也不显示在图中。

附加在操作类（listOfElements : list）的后置条件加标签值可以是：

```
The number of elements in list must be greater than one.
```

通常把不变性、后置条件和前置条件合起来使用实现“根据约定合约编程”。类型实

现符保证在系统执行的整个过程中满足不变性条件；操作的调用方保证在调用该操作之前前置条件为真；操作实现子保证操作执行完后满足后置条件为真。当出现任何破坏这些“约定”的情况时，系统可能发出一个意外（表明系统发生内部错误）。

- 责任标签指定了类型的责任，它的值是一个字符串，表示了对其它元素的义务。责任通常是用对其它元素的义务来描述的。
- 抽象是一个加标签值，表明某个类不能有任何对象，如图 7-8 所示。该类是用来继承和专有化成其它具体的类的。这个值是 Boolean 类型，其默认值为真。通常这个加标签值都显示在类图中，在类名框内类名的下面。

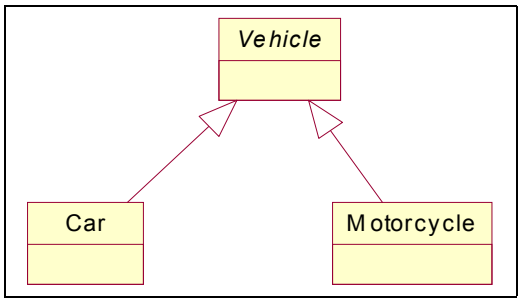

图 7-8　Vehicle 是一个抽象类，它没有对象。只有 Car 和 Motorcycle 类的对象存在

- 持久性应用于类型，把类型定义可持久的说明该类的对象可以存储在数据库或文件中，并且在程序的不同执行过程之间，该对象可以保持它的值或状态。
- 语义、空间、时间语义应用于类型和操作。语义是类型或操作的意义的规格说明，空间语义是相关类型或操作的空间复杂性意义的规格说明，时间语义是时间复杂性意义的规格说明。这些加标签值只在说明 UML 时才使用，不再详细讨论。

7.3.3　模型元素和组件的加标签值

位置是应用于模型元素和组件的加标签值，它说明某个模型元素位于哪个组件（如哪个源代码文件），或在哪个结点（如哪个计算机上）。位置的值是组件或结点。

7.3.4　自定义加标签值

加标签值由名（标签）和值（某种类型）组成，可以连接到任何元素上，用来为这些元素加上一些新的语义。通常用户自定义的加标签值是用来增加有关项目进展或状态的管理信息，或者是模型未来的转换信息。加标签值是有关模型和模型元素的附加信息，在最终系统中是不可见的。可以按照以下步骤定义加标签值：

1. 研究要定义的加标签值的意图；
2. 定义它要附加在其上的元素；
3. 为该加标签值起一个恰当的名字；
4. 定义该值的类型；
5. 弄清楚谁将使用该标签值，是人还是机器，并根据对象的不同，给出合适的定义；
6. 在文档中给出一个以上使用该加标签值的例子。

两个自定义的加标签值的例子是加在一个类里面的某个操作上的 Standard algorithm

和加在任何元素上的 Author。前者指明该操作使用何种算法，后者指明该元素的作者。如：

```
{ Standard algorithm = "Quick sort" }
{ Author = "Peter Smith"}
```

7.4 约　　束

约束是元素的一种语义条件或限制，它应用于元素。一条约束应用于一个种类的元素，因此一条约束可能涉及许多元素，但它们都必须是同一类元素。约束是显示在大括号内（{constrraint}），可以直接放在图中，或者独立出来（有些 CASE 工具中在一个单独的窗口内显示约束）。如果约束应用于一种具有相应视图元素的模型元素（如类，它的视图元素是一个矩形），它就可以出现在它所约束元素的视图元素的旁边。如果一条约束涉及同一种类的多个元素，则要用虚线把所有受约束的元素框起来，并把该约束显示在旁边（如或约束）。在 UML 中有 14 种标准约束，它们是：关联 、全局、局部、参数、自我、完整、不相交、不完整、覆盖、或、有序、投票、广播。用户也可以自定义约束。

7.4.1 对通用化的约束

完整、不相交、不完整和覆盖是四种应用于通用化的约束，它们只能应用于子类。这些约束都是语义的，显示在共享的空矩形内的大括号里，用逗号隔开（如果在该通用化中有几条路径同一个空矩形）。如果没有共享，则用一条虚线通过所有继承线，并在虚线的旁边显示约束（如图 7-9 所示）。

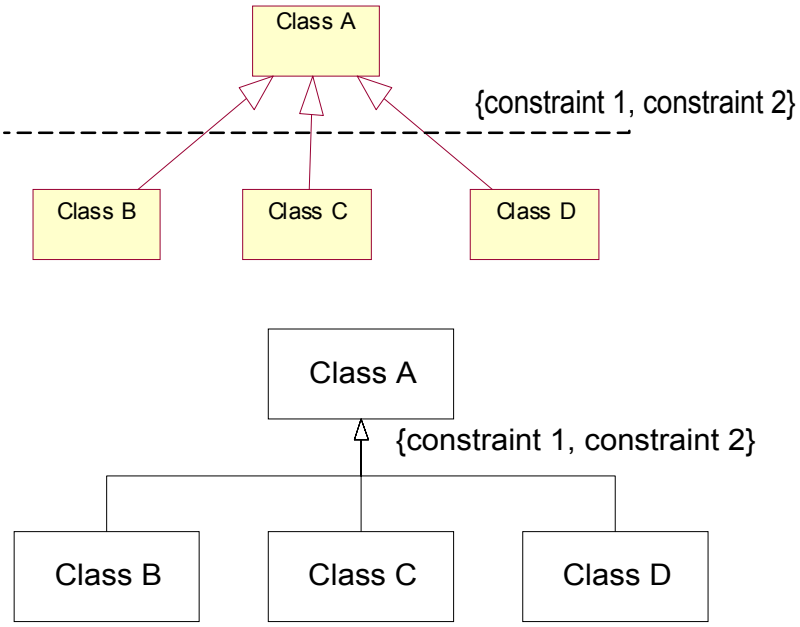

图 7-9　对通用化的约束的两种表示方示

覆盖继承是指在继承关系中，任何进一步继承的子类可以继承一个以上的子类。换言

之，在继承树上的同一个超类，可以有多继承。如图 7-10。

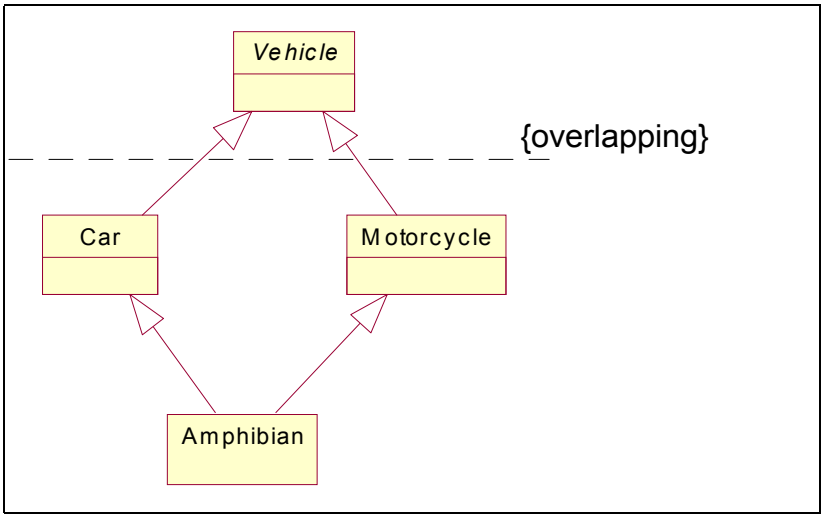

图 7-10　覆盖继承

不相交继承是默认的继承方式，与覆盖继承相反。此时，不允许从同一超类继承而来的子类专有化到相同的子类。

完整通用化约束中指定了一个继承关系中的所有子类，然后就不允许增加新的子类，不完整通用化约束与此相反，以后可以增加新子类。不完整通用化是默认值。

7.4.2　对关联的约束

对关联有两种默认的约束：隐含和或。隐含约束表明关联是概念的，而不是物理的。隐含的关联联结类，但对象之间并没有关系。隐含关联中的对象之间也没有物理连接；而是通过其它一些机制产生联系，比如对象或查询对象的全局名。

或约束指定一组关联对它们的链接有约束。或约束指定一种类的一个对象只连接（或链接）到一个关联类对象（关联的另一部分）。例如，一个人可以有多个（0 个或多个）保险合同，一个公司也可以有多个保险合同，如图 7-11 所示。

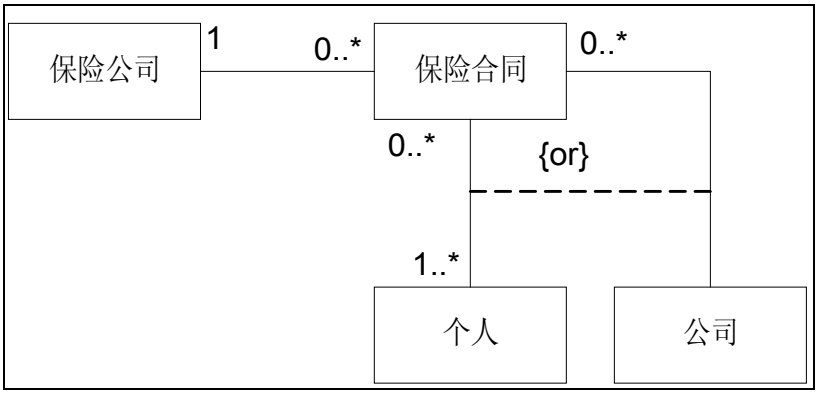

图 7-11　个人和公司可以有 0 个或多个保险合同，一个合同可以由一个或多个人或一个或多个公司拥有。如果没有或关联，一个或多个个人以及一个或多个公司可以拥有保险合同。如果在个人或公司

一连改变了重数（如 0 个或多个），这将影响语义，允许一个保险合同不属于任何人

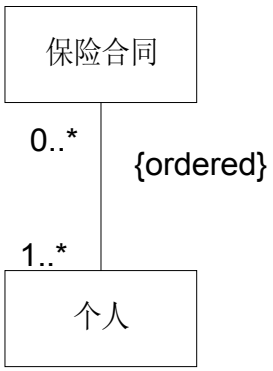

图 7-12　约束{ordered}指定链接之间有明确的顺序。顺序可能显示在大括号内，例如{ordered by increasing time}

7.4.3　对关联角色的约束

有序约束是唯一对关联角色的标准约束。一个有序的关联指定关联里的链接之间有一个隐含的顺序。例如，在屏幕上的窗口有一个隐含的顺序。因此，屏幕和窗口之间的关联是有序的。这一约束显示在关联附近的大括号内 7-12 所示。有序或无序约束都可以明确地显示出来，但对关联来说，默认的是无序，无需指明。

7.4.4　对消息、链接角色和对象的约束

有九个约束应用于交互。可以用顺序图从时间的角度来看交互，也可以用协作图从空间的角度来看交互，或者还可以用活动图从工作的角度来看交互。在所有类型的交互图中（顺序图、协作图和活动图），消息、对象和链接是最重要的概念。对象是通过链接连接起来的（而不是关联），每个链接都有两个链接角色——是关联角色的实例。当对象进行交互时，它们扮演角色（链接的角色）并且通过链接相互发送消息。

应用于这些重要概念的约束是全局的、局部的、参数、自我、投票、广播、创建、注销和临时。

- 全局的应用于一个链接角色，指定相应的实例是可见的，因为它是一个全局量。通过系统都知道的全局名就可以得到该实例。
- 局部的应用于一个链接角色，指定相应的实例是可见的，因为它是操作的局部变量。
- 参数应用于一个链接角色，指定相应的实例是可见的，因为它是某个操作的参数。
- 自我应用于一个链接角色，指定对象可以向它自己发送消息。
- 投票应用于消息，限制一组返回的消息。投票约束指定从这组消息中以大多数投票的方式选出返回的值。
- 广播应用于消息，指定不能以某种顺序发出该消息。
- 创建影响对象生命，将在交互的执行中创建对象。
- 注销影响对象生命周期，对象将在一个交互的执行中被注销。
- 临时是影响对象生命周期的一个约束。临时的对象的创建和注销是在一个操作的同一执行中，它是创建和注销约束的结合。

7.4.5 自定义约束

正像前面提到的，允许用户自定义约束。约束通过条件或对语义的限制来影响元素的语义。因此，当自定义约束时，一定要仔细评估它产生的影响，最好在文档中给出如何使用自定义约束的例子。所以，自定义约束需要做以下工作：

- 描述要约束的元素
- 对该元素的语义影响
- 给出一个或多个使用该约束的例子
- 说明如何实现该约束

7.5 版　　类

版类是把 UML 中已经定义的元素的语义专有化。UML 中元素具有通用的语义，用版类可以对它们进行专有化和扩展。版类不是给元素增加新的属性或约束，而是直接在已有元素中增加新的语义，这种机制可以看作是对已有元素进行某种专有化注意，版类可能扩展已有元素的语义，而不是元素的结构。

版类是可通用化的元素，也就是说它们可以被专有化或通用化。如果一个可通用化元素（类型、包或版类）有一个版类并且被专有化，则版类也被专有化继承，这和加标签值和约束是一样的。

UML 中预定义了 40 种以上的版类，和加标签值及约束一样，也可以自定义版类。版类通常有相应的视图元素，是文字串<<版类名>>。该视图元素通常显示在它所基于的元素的上方或前方。版类通常应用于类、类型、关系、结点、组件和操作。

版类可以有一个图形图标，是可选的，图形图标可以是彩色的。图标和版类可以有几种方法结合，第一种是，把图标显示在版类化的元素内。如果一个类被版类化，就可以把图标放在该类视图元素的类名框内，或者可以显示版类名和图标，或者只显示图标。如图 7-13。

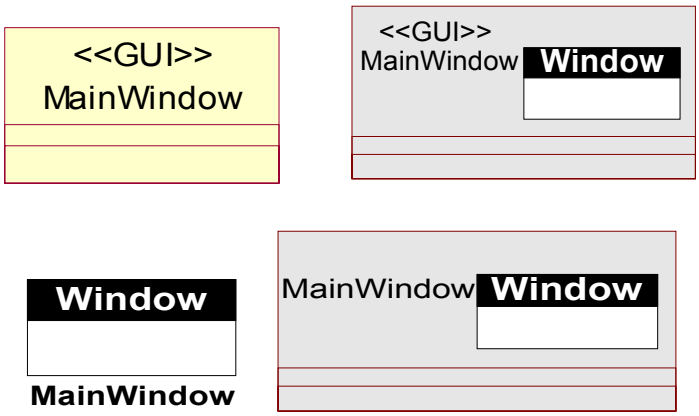

图 7-13　版类 MainWindow 类的几种不同的表示方法。该类的版类是<<GUI>>，具有一个图形窗口图标作为它的可视化表示

7.5.1 版类对类型的应用

7.5.1.1 角色

角色是一种版类化的类型，代表一个抽象，位于被建模系统的外部，指定了外部角色如果与系统交互，以及它们需要系统的什么功能。角色版类是在用例建模中实现的。因为类是类型的子类，因此版类角色通常也被应用于类。因为在一个用例中，角色是一个类，并且可以具有属性、操作以及与其它类之间的关系。但是角色的版类通常是<<actor>>，给出该类的特定语义。角色版类有一个小棍人图标，如图 7-14。

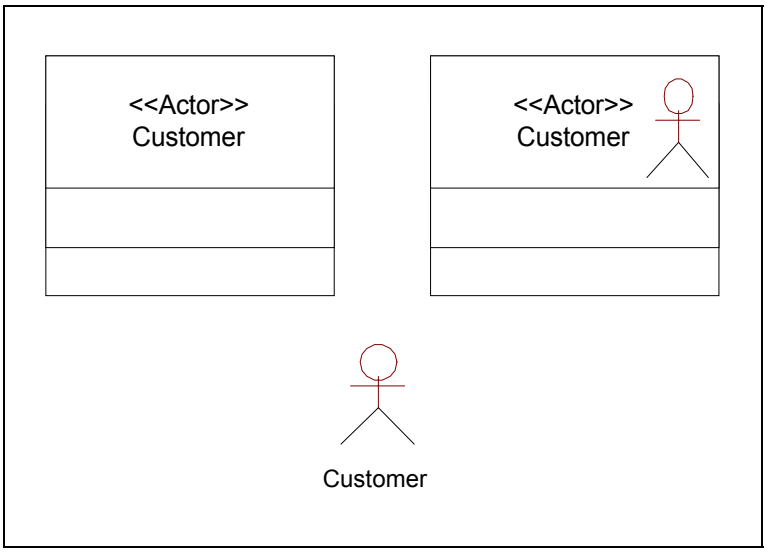

图 7-14 版类角色的不同表达方式

7.5.1.2 接口

接口是版类化的类型，公开类型、类、结点、组件等的行为。当一个包、组件或类等有接口连接时，就说它实现或支持指定的接口（因为它支持接口中定义的行为）。接口是用版类<<interface>>化的类描述的，通常只由抽象的操作组成。接口可以由任何数量的组件、包或类来实现，如图 7-15 所示。

7.5.1.3 元类

元类是一种类，它的实例是类，因此元类是类的类。当你需要在运行时定义类时，元类非常有用。元类是一种版类化的类型，一般应用于类（类是类型的子类，因此版类元类也可以应用于类）。元类可以应用于所有类型（例如类型的子类），如类、原始类型和用例。元类没有图标，只用<<Metaclass>>标记，Smalltalk 已经实现的元类。

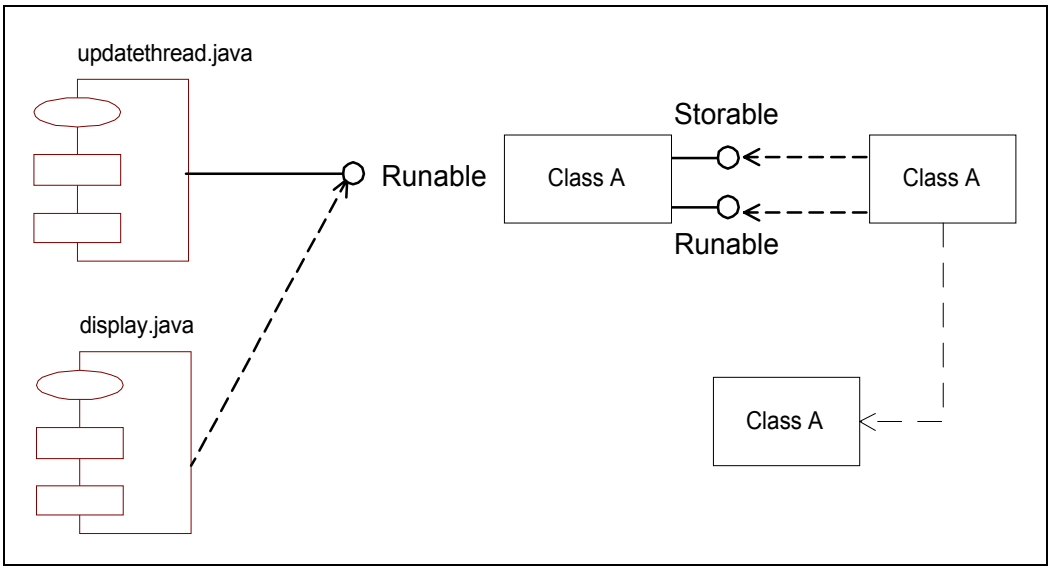

图 7-15　连接到类和一个组件的接口

7.5.1.4　动力类型

当对一个通用进行专有化时，需要一个区别子来指定继承的基础。比如，在一个通用结构中，超类 Purchase 可能有子类 Pusrchase On Credit、Cash Purchase 和 Hire Purchase，区别子可以是买单，因为它区别了子类的实例。

区别子是用于区分实例的，它属于实例层。区别子类型叫做动力类型，动力类型属于类型层，表明超类（通用化的类型）是根据什么成子类的。因此，当我们谈到实例时，用区别子来专有化；而当我们谈到类型时，我们用动力类型（powertype）来专有化。图 7-16 是例子。动力类型是一种版类化的类型，通常在类型中用<<Powertype>>显示；它是可选的；只有区别子可以使用（显示区别子的动力类型）。

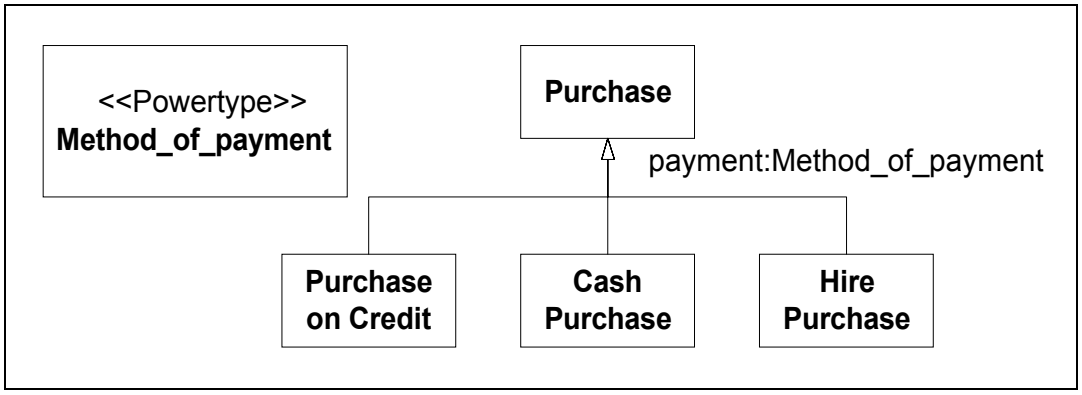

图7-16　区别子（Payment）和<<Powertype>>（Mehtod_of_payment）用来专有化 Purchase

7.5.2 版类对相关性的应用

7.5.2.1 变成

变成是一种版类化的属性，它的源和目标是同一实例，变成版类表明实例将从一个状态（源实例状态）转变（时间和空间上）到另一个状态（目标实例状态）。它可以用于显示是如何从一个结点移动到另一个结点的——也就是说，在一个分布式系统中，对象的位置是如何改变的——或者在其它情况下，它可以明确地显示元素的状态是如何改变的。

7.5.2.2 调用

调用是一种版类化的相关性，连接操作。如果类 A 对类 B 有调用相关性，则类 A 中的操作可以调用类 B 中的操作。

7.5.2.3 复制

复制相关性连接两个实例，其中源实例是目标实例的一份拷贝（但它们仍然有各自的标识符并且是不同的实例）。

7.5.2.4 派生

UML 的规则都是约束或派生的。例如，属性可以从其它属性派生而来，派生是一种版类化的相关性。如图 7-17。

图 7-17 profit 是一个派生属性。派生的规则显示在大括号内

7.5.2.5 友元和导入

友元是一种版类化的相关性，是从一个包到另一个包的相关性。友元相关性扩展了包之间导入的可见性。如果相关性是友元，则源包可以访问目的包中公开、保护和私有元素，但不能看到实现。因此，可以访问友元相关性包的私有或保护的元素，而其它包则不行。

包之间的版类化导入相关性是默认的，只能访问包内的公开性元素。

7.5.2.6 实例

版类化实例相关性是实例及其相应类型之间的关系。源是一个实例，目的是一个类型。它可以明确地显示对象是哪个类的（从对象到类有一个版类化的<<instance>>相关性）。

7.5.2.7 精化

精化是一种版类化的相关性，有自己的图标（也是 UML 的一种视图元素）。它很好地说明了 UML 是如何从很小的模型元素核心扩展而来的，以及它是如何用来定义和扩展它自己的。前面已有详细的讨论。

7.5.2.8 角色

角色是类型和关联角色之间的一种版类化的相关性，前面已有详细的讨论。

7.5.2.9 发送

发送是一种版类化的相关性，其中源是一个操作，目标是一个信号（是一个类，其版类是信号）。操作发送信号。

7.5.2.10 跟踪

跟踪（trace）是从一个模型元素到另一个模型元素的一种版类化的相关性。相互跟踪的元素可以在同一个图中或在不同的图中。跟踪表明源概念上跟踪回目标。跟踪版类可以用于显示一个图是另一个图的详细描述，或者设计图跟踪同一事物的分析图。

7.5.3 版类对组件的应用

一些版类可以应用于组件，通过定义各组件代表了现实世界中的什么事物，使组件图更准确和清楚。

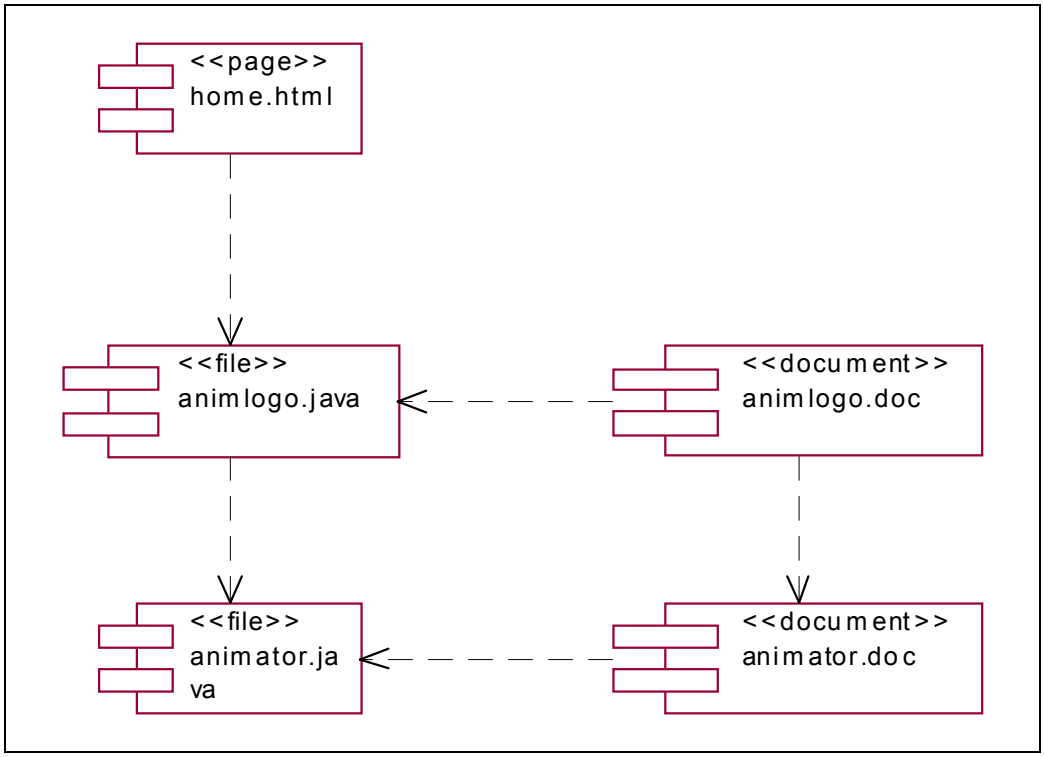

图 7-18　源代码组件之间的相关性

- 版类化组件<<application>>代表一个可执行的程序。
- 版类化组件<<document>>代表一个文档（其中是文档而不是可以编译的源代码）。
- 版类化组件<<file>>代表一个是源代码的文件，如图 7-18。
- 版类化组件<<library>>可以用来表示一个组件是一个动态或静态的图书馆。
- 版类化组件<<page>>代表一个 Web 页面。
- 版类化组件<<table>>代表一个数据库表。

7.5.4 版类对笔记的应用

约束是一个版类化笔记，其中包含了对笔记所连到元素的约束。需求是一个版类化的笔记，指定了笔记所连到元素的责任或义务。

7.5.5 版类对原始类型的应用

7.5.5.1 枚举

枚举是一个版类化的类型，也叫做定义域。它为枚举的原始类型指定了一组允许值的集合，可见性是一个枚举，可以取的值包括：公开、保护、私有和实现。

7.5.5.2 公用类型

公用类型是一个版类化的类型，只含有类范围的操作和属性。公用类型没有任何实例，它是只能通过该类型而不能通过实例访问到的一组类范围的操作和属性。所谓类范围的操作是因为只指定类名和操作名。公用类型通常应用于类，如图 7-19 所示。

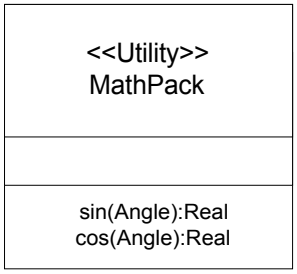

图 7-19 公用类型类，其中数学函数被组合在一个类中，作为类范围的操作

7.5.6 版类对通用化的应用

7.5.6.1 扩展

扩展用例之间的一种版类化的通用化，是用来指定一个用例是从另一个用例继承而来，当定义专有化用例时，可以使用被继承用例的部分。

7.5.6.2 使用

使用是两个用例之间的一种版类化的通用化，描述一个用例被另一个用例使用。

7.5.7 版类对包的应用

7.5.7.1 外观

外观是一种版类化的包，只引用其它包的元素（导入，通过友元相关性等）。外观没有任何元素（它的内容只是引用）。

7.5.7.2 残段

残段是一种版类化的包，代表一个不完全实现的包；它代表另一个系统的一部分，可以有数据库等的残段。

7.5.8 版类对类的应用

7.5.8.1 信号

信号是一种版类化的类，约束该类对象的语义，使它们可以作为信号发送，如图 7-20 所示。信号是一种有名的事件，携带一个对象用在系统中，在不同的活动对象之间发送异步消息。可以建立信号的层次结构来支持多态。

7.5.8.2 控制、边界和实体

可以把类版类化成控制、边界和实体。这些版类是用来增加类的语义，并在建模时使用。它们是根据模型—视角—控制者的概念，其中实体是模型，控制是控制者，边界是视角。模型—视角—控制者是一种常用的方案（模式），用来建立具有一个实体模型的系统，对该系统的操作是通过一个图形用户界面（GUI），如图 7-21。所以，这些版类在处理基于图形用户界面系统时非常有用。

边界版类把类专有化以便于表达和操作。边界类携带信息，从一个系统向另一系统如一个人或机器（就是角色）。边界类也可以允许对信息的操作（通过在实体对象中定义的规则）。边界类典型地是窗口、对话或通信类，如 TCP／IP。

实体版类用于为商业对象（核心概念）建模，如借条、发票、保险单等。它们通常是持久性的，可以保存在系统中。

控制版类用于把边界对象和它们的实体对象联结起来，来处理系统内的一个操作序列。实体对象携带需要由边界对象处理的登记处。控制对象通常负责实体对象内的信息处理问题，并完成一些可能涉及多个实体对象的功能序列。

通过这三个简单版类，可使类图更为精确并且容易理解。因为版类表明了类的意图和责任，并且在图中立即可见。该类更详细的讨论可以在它的文档和责任性质中访问到。而且，类之间的关系也更容易理解，例如，如果一个边界类与一个实体类有关联，则通常实体内的信息将通过该边界类型显示和操作。

用户可以为类、以及特定问题领域的公用类角色定义版类，其中某个特殊类型的类有它们自己的图标和语义。例如，在金融领域，表达一个有价格的项的类，它的版类是<<Asset>>；如果一个类的实例的价格计算受到另一个类实例的价格影响，就用版类化的相关<<Affects>>——例如，股票的价格受到底层股票市场价格的影响。使用版类的机构

将决定版类的语义的形式化程度和扩展程度。但是，版类的意思必须明确（如图 7-22）。

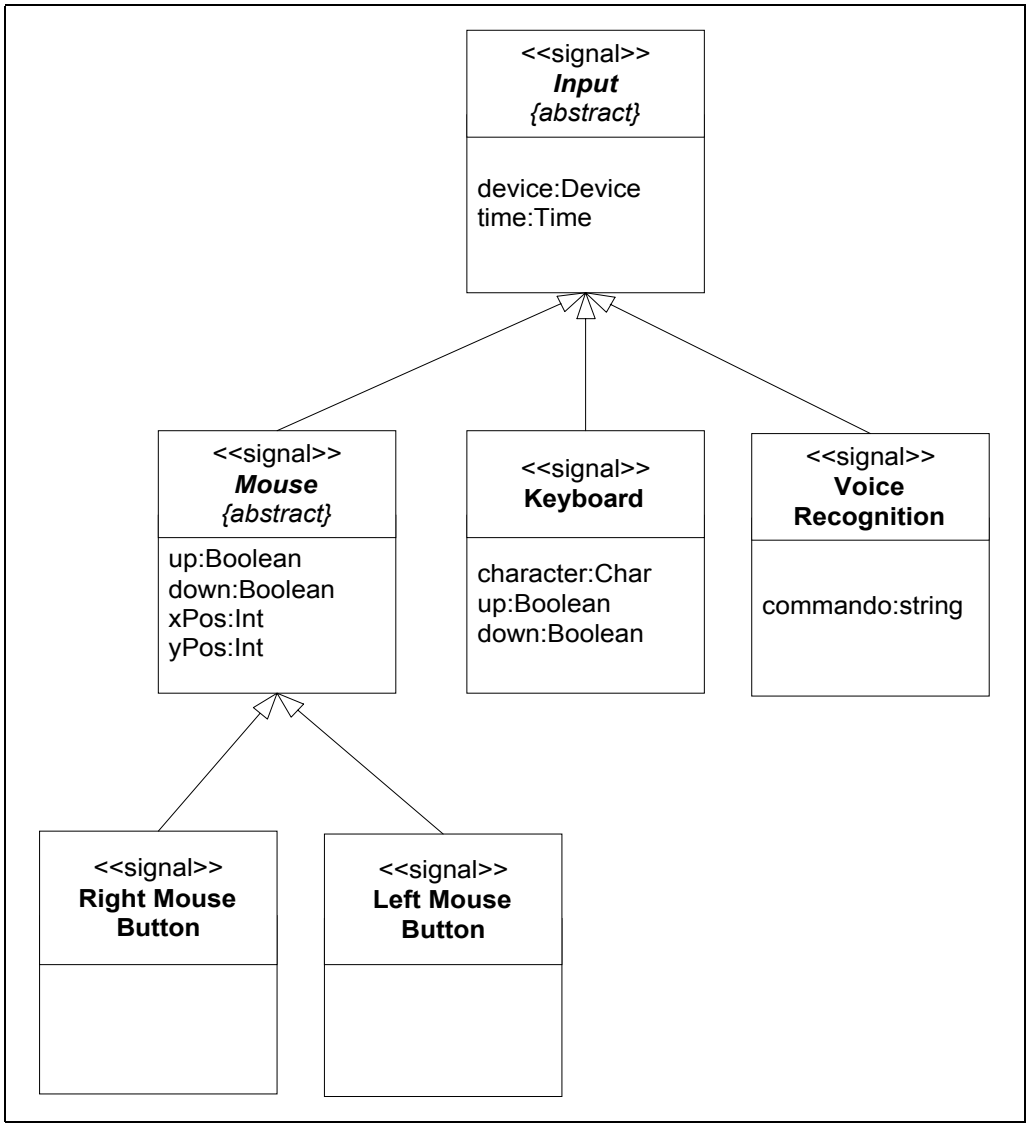

图 7-20　有信号版类化类的分层

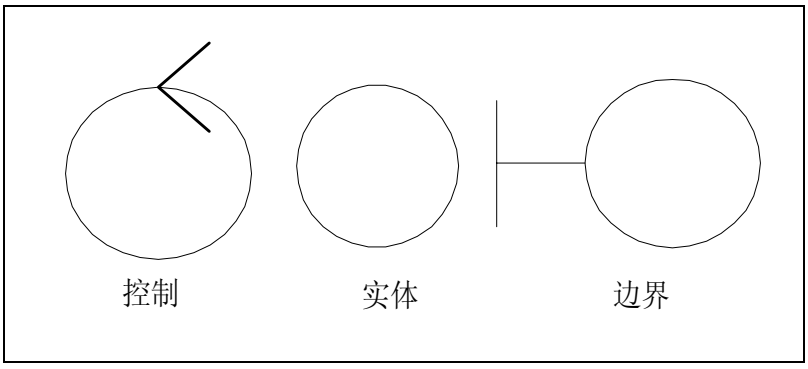

图 7-21　控制、实体和边界版类

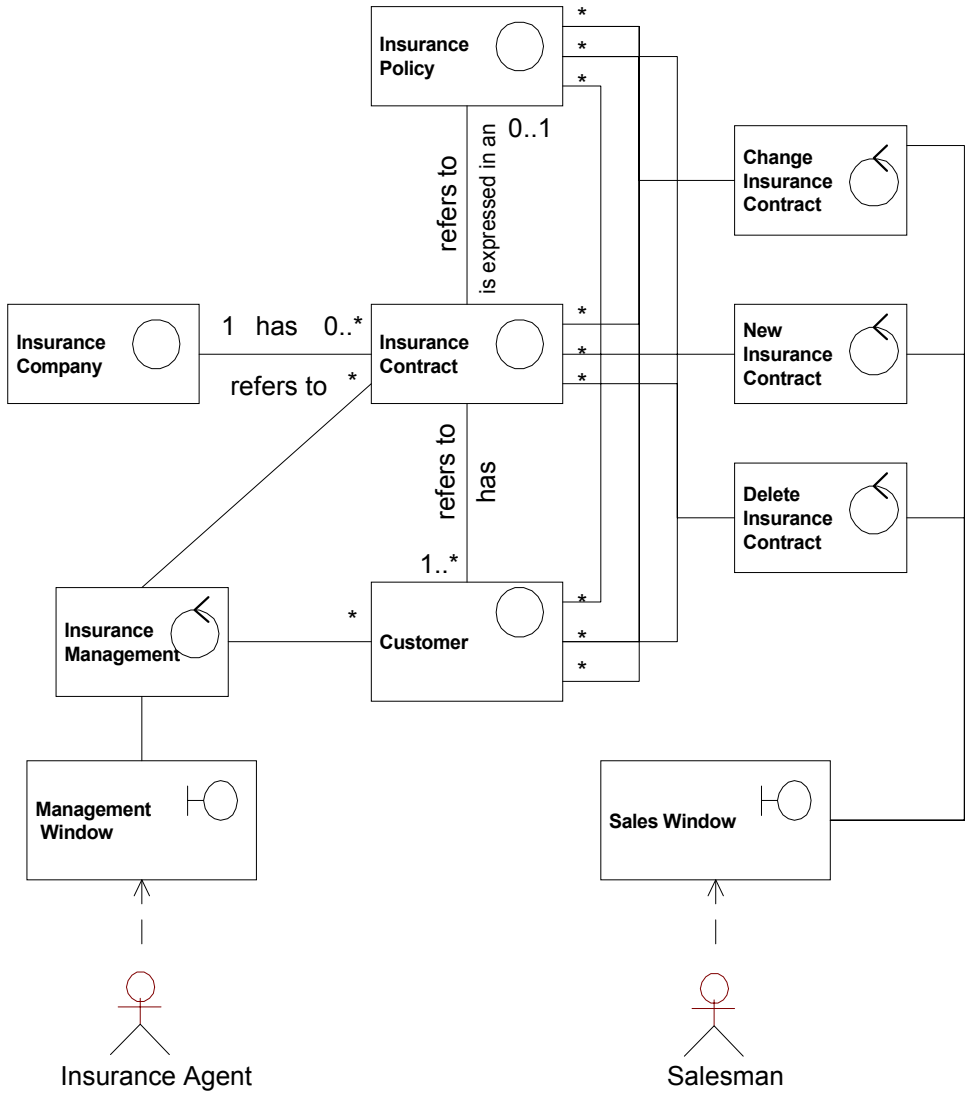

图 7-22　控制、实体和边界版类。在这个例子中，实体类体现了商业的核心保险商业。实体类是 Insurance Policy、Insurance Contract、Customer 和 Insurance Company。控制类是 Change Insurance Contract、New Insurance Contract、Delete Insurance Contract 服务于边界类是 Sales Window。控制类 Insurance Management 用 Customers 和 Insurance Contract 服务于边界类 Management Window

7.5.9　版类对操作的应用

构造子是一个操作，当创建对象时调用。它可以有参数，可以明确调用，也可以有名字，但通常它的名字与类的名字一样（如 C＋＋，JAVA 等要求这样）或简单地只是名字“create(…)”。例如，发票类可以有一个构造子叫做发票，参数是规格和数量。明确的构造子调用是：

```
invoice ( "A dell computer", 2000 $)
```

解构子是一个操作，当对象不再使用或被注销时隐含调用的。所有面向对象的程序设

计语言都支持构造子和解构子。

查询操作不影响对象的状态，典型的查询操作是读取对象的属性。构造子和解构子以及查询都是版类化的操作，如图 7-23 所示。

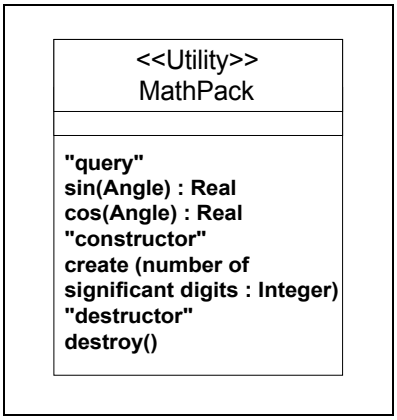

图 7-23

7.5.10 版类对活动类的应用

有两种版类应用于活动类：过程和线程。过程是一种版类化的活动类，表示一个重控制流；线程是一种版类化的活动类，代表一个轻的控制流。这两种版类用来表明活动类是如何实现并映射到底层的操作系统中的。

7.5.11 自定义版类

在用户自定义的加标签值或约束中，都在元素中加入了值或语义，此时定义了一种新的元素类型，尽管它是以原有元素为基础。版类也有自己的视图元素，它可以有一个图标，在图中代表版类元素。

通常，当人们在特定的方法或特定的应用领域中使用 UML，会使用版类。某些概念属于方法或领域所特有、而标准 UML 又不支持的，用户就可以自定义。在定义版类时，需要做以下工作：

- 描述自定义版类的基础是哪个元素
- 对该元素的语义影响
- 给出一个或多个使用该版类的例子

<<Semaphore>><<Time>><<GUI>>就是自定义版类。时间版类应用于关联，指定有一个时间相关性。GUI 版类是边界版类的一个专有化。它与边界类有相同的功能和意图，但只与图形用户接口有关（边界类还涉及其它与用户或系统通信的方式）。因此 GUI 类必须是图形用户接口的组成部分，例如，一个窗口或一个消息框。图 7-24 就是一个例子。

当然，也可以把元素扩展到模型元素以外的其它元素。可以把概念模型（元素的子类）版类化为一组不同的系统，例如：信息系统，技术系统，嵌入式实时系统，分布式系统，商业系统，软件系统等。

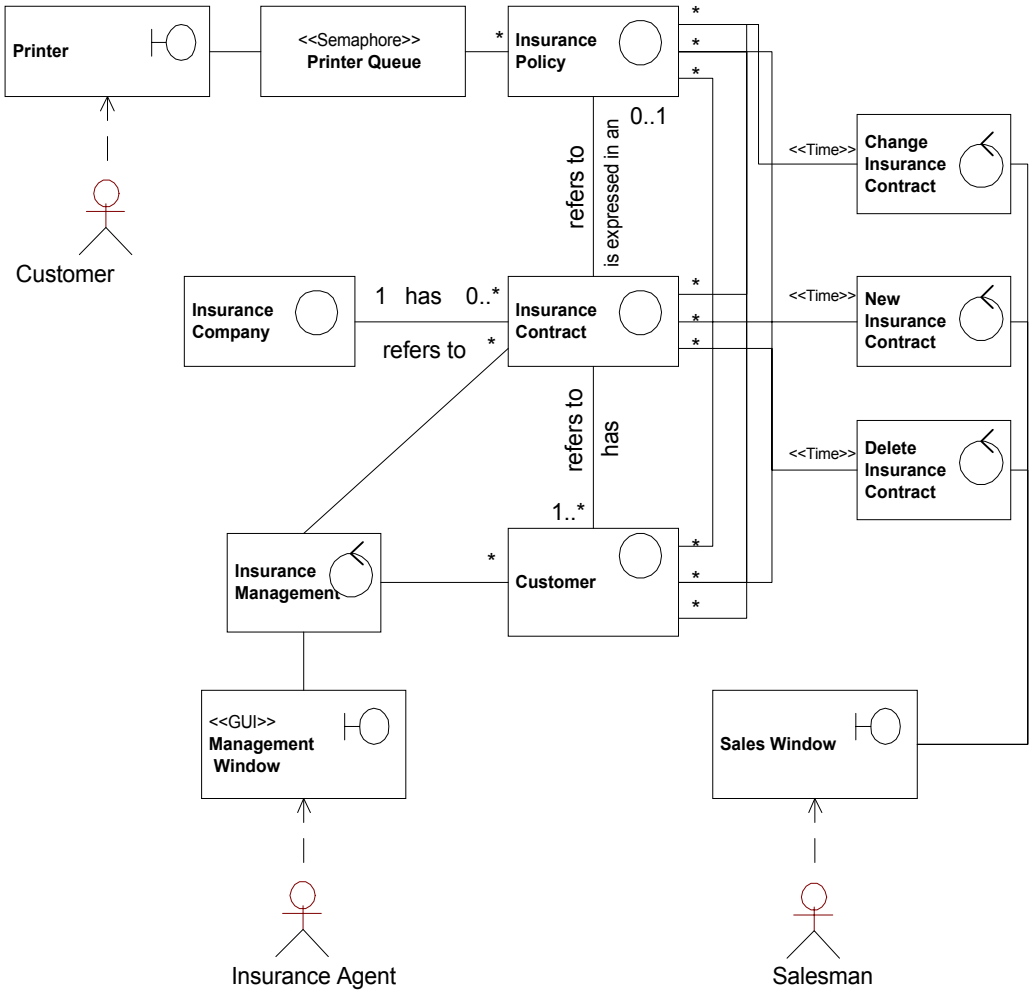

图 7-24　用户自定义版类的例子。Printer Queue 类是一个信号；它控制由 Printer Queue 访问，从而把 Insurance Contracts 类和 Printer 类连接起来。控制类和 Insurance Contracts 之间的关联是用时间版类化的，Management Window 和 Sales Window 是 GUI 版类化的

7.6　小　　结

所有的模型语言都有它的限制。如果语言太通用，虽然它的表达能力很强，但却不实用，在使用起来不方便；如果语言太专有化，那么它的表达能力就不够强。C++可以看成专有化为程序设计用的建模语言，但它的表达能力却不够强，无法用于商业建模。因此 UML 的定义由一组数量有限的通用的元素再加上扩展机制组成。

UML 有三种扩展机制：加标签值，约束和版类。用这些机制定义了一些 UML 的标准扩展，在其上还可以加上新的扩展（如用户自定义的扩展）。通常加上扩展都是为了使 UML 适应某种方法、某个机构或某个特定的应用领域。

扩展机制以元素为起点。在描述 UML 的元模型中，“元素”是 UML 所有组成成份的抽象基类，它是一个基础，其上可加上多种机制；它可以被专有化为模型元素（元素表示的抽象）、视图元素（用来显示该元素的符号）、系统和模型。

加标签值这种扩展机制是在模型和它们的元素上加上更多的信息，这些信息可以给人使用，也可以由机器（如代码生成器和 CASE 工具）使用。

约束是用来限制 UML 元素的语义的。约束不能给已有的 UML 元素增加语义，它只能限制元素的语义。

版类是用来把 UML 元素的语义专有化。用它可以更精确地定义和建模。尽管版类可以取代加标签值和约束，但区分这三种扩展机制仍然是非常实用的。