计算机科学 1999Vol. 26№. 4

现代软件开发方法中的 use case 技术

The Use Case Driven Approach in Modern Software Development Methods

刘作伟　宁　洪

（国防科技大学计算机学院　长沙 410073）

Abstract　Use case driven approach is used widely in most software development methods. According to UML1.1, this paper introduces some concept about "use case" firstly, then describes the process of analysis, design and realization of use case, especially some details on description, refinement and realization, finally, some noticeable problems appeared in such process are addressed.

Keywords　Use case, Actor, Software model, Object

1　引言

在面向对象的软件开发方法中，客户关心的是软件系统的功能，而开发人员的工作是围绕目标软件系统中的对象来进行的。为了把软件系统的对象模型和功能模型有机地结合起来，在众多的面向对象开发方法中都采用了一种称为"use case"的技术。use case 的分析、设计和实现与面向对象的分析、设计和实现结合起来，提供了一种贯穿整个软件生命周期的开发方式，使得软件开发的各个阶段的工作自然、一致地协调起来。在 UML(1) 系统建模规范中，更把 use case 做为其中的一个重要组成部分。下面以 UML 中的相关模型为例，介绍 use case 技术。

2　use case 及其相关概念

2.1　定义

1995 年，Jacobson 把 use case 定义为，一个 use case 描述在一个系统中一组事务(transaction)执行的先后序列，这组事务的执行将向与该系统交互的使用实体(actor)返回可以度量的结果。此定义涉及到以下几个概念：

· 一个 use case：描述了系统中要发生的一个事件流，其中包括了具体的事件和事件发生的先后次序。

· 使用实体：在使用系统时，某个人或事物所充当的角色。实体可以是与系统相互作用的任何人和事物，同一个人或事物可以充当多个实体角色。一个 use case 可以与多个使用实体进行交互。使用实体可以是类、系统、子系统、另一个 use case 等。

· 事务：系统为完成某个功能而要执行的操作，事务具有原子性。

· 可以度量的结果：意味着事务序列的执行所返回的结果是可以预见或可以计量的，对使用实体来说是可以预计和有意义的。

一个系统中所有 use case 定义的集合就规定了该系统的全部功能。

2.2　use case 之间的相互关系

use case 之间存在两种关系，即扩展关系和使用关系：

· 扩展关系(extend)：A、B 是不同的 use case，A 扩展 B 的含义是，B 的一个执行过程可以引发 A 中定义的行为(在 B 的一个扩展点上且扩展点条件为真时)。一个 use case 可以被多个 use case 扩展(有多个扩展点)。use case 之间的扩展关系对应着 use case 的细化过程，这个细化过程具有两个特点：(1)是增量式的，即系统中已经存在一个解决某个问题的具体途径的定义时，为解决另外的问题可以把更多的系统行为加入到该定义中，从而扩充、细化系统的功能；(2)这种扩充、细化工作并不直接修改系统中已经存在的定义。

(1)UML 是 Unified Modeling Language 的缩写，是由 Rational 公司向 OMG 组织提交的一个面向对象建模语言，目前已被广大软件开发商所接受，有望成为面向对象建模语言的标准。

·使用关系(use):A 使用 B 的含义是,A 可以使用 B 定义的行为,即 A 的执行过程必定包括 B 中定义的行为。一个 use case 可以使用多个 use case。使用有两层意思:使用 use case 的结果(强调被使用 use case 所返回的结果)或使用 use case 中所定义的行为(强调 use case 的执行对系统产生的影响)。使用关系反映了共享的概念,即多个 use case 共享一个 use case 中定义的行为。

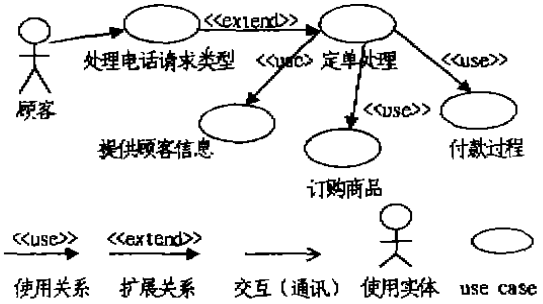

图 1 use case 关系

以一个销售商为例子,采用 UML 规范中的图形表示法,建立的 use case 模型图如图 1 所示。例中,顾客通过电话同销售商进行联系,既可以询问商品的有关情况,也可以在电话中订购商品。图中,"定单处理"扩展"处理电话请求类型";"定单处理"使用"提供顾客信息"、"订购商品"、"付款过程","处理电话请求类型"处理顾客的电话请求,仅当顾客要订购商品时,才会引发"定单处理",而在执行"定单处理"时,其它三个 usse case 都将被执行。

2.3 use case 类型和 use case 实例

前面提到的"一个 use case"实际上是指一个 use case 实例,它给出目标系统为完成某个功能而要执行的一个具体的动作序列,即目标系统中的一个事件流,如果目标系统中存在多个事件流,并且这些事件流具有相似性(包含的事件和事件发生的先后次序相似)时,则可以定义一个 use case 类型,用此 use case 类型实例化一次即得到一个 use case 实例。因此,标识和定义 use case,实际上就是要定义 use case 类型。

从 use case 的有关概念可以看出,当系统运行时,多个不同 use case 类型的实例和一个 use case 类型的多个实例可以同时存在,从而在软件开发前期从一定程度上反映出目标系统的实际运行情况。

3 use case 的分析、设计和实现

3.1 use case 的分析和描述

use case 的分析工作主要有:(1)确定 use case,包括识别 use case、定义与 use case 相互作用的外部实体、分析各个 use case 之间的关系等等;(2)use case 的精化;(3)use case 的具体化;(4)组织和管理 use case。

3.1.1 确定 use case。通过软件需求分析工作,在识别出 use case 和相关的外部实体的同时,要进行 use case 的描述工作。use case 的描述工作从软件开发者的角度来文档化 use case 的内容,作为软件系统后续工作的基础。use case 的描述至少应包含下面的内容:

- use case name:(use case 的名称)
- actors:(相关的外部实体)
- participants:(包含的参与者,如目标软件系统中的对象)
- parameters:(外部实体与 use case 相互交换的参数)
- pre:(在 use case 实例执行前系统应该满足的条件)
- post:(在 use case 实例执行后系统应该满足的条件)
- relationships:(use case 参与的关系的描述)
- other:(其它方面的描述)

3.1.2 use case 的精化。其本质就是给系统模型中的每个模型元素(如子系统、类等)收集所有的 use case,并模型化这些模型元素之间的协作过程。在软件建模中,采用包(Package)的嵌套层次来组织 use case,在某一层次的包结构中,use case 的精化过程中所包含的每个模型元素的精化工作可以同时进行,因此,描述和设计对象之间的协作过程包含了 use case 的精化过程和实现过程。

精化的工作是根据某一个层次来进行的,精化过程要适可而止,既不要太粗,也不要太细。对于 use case 的描述来说,太粗会使得 use case 显得模糊;太细的话,系统中其它地方的某一个改动很容易影响到它,使得描述 use case 的工作变得更复杂。在精化过程中一个重要的方法就是用下一层次的多个 use case 来细化上一层次的一个 use case,下一层次的某些 use case 之间以"使用实体/use case"(其中一个 use case 作为另一个 use case 的使用实体)的关系相互关联,上一层次的usecase的使用实体一

定是下一层次的多个 use case 中某个或多个 use case 的使用实体。

精化的过程还包括进一步确定 use case 所涉及的范围和 use case 所涉及到的模型元素等。

3.1.3 use case 的具体化。一个 use case 可以描述多个执行场景，一个具体的 use case 实例执行 use case 的一个执行场景中定义的行为。因此，use case 实例化是 use case 具体化的方式之一。这种具体化过程中所做的工作有：use case 中包含的所有可能的执行场景的描述；use case 中包含的分支条件的确定等。

在软件工具 Rational Rose[2] 中提供了一种具体化方法——以面向对象的概念来进行 use case 的具体化工作。其思想是，以 use case 模型为基础，把整个 use case 模型看成是一个类工厂。use case 模型中包含三种特殊的类：system 类、allactor 类和 use case 类。

- 把每个 use case 定义为类 system 的一个操作；
- 每个使用实体是类 allactor 的一个实例；
- 对于类 system 的每一个 use case 操作，创建一个 use case 类；
- use case 包含的每一个执行场景成为相应的 use case 类的一个操作，执行场景名称就是操作的名称。

当使用实体与 use case 交互时，按下面的顺序来执行一个 use case 实例：

- 使用实体通过向 system 发送一个消息来启动一个 use case 实例，这个消息称之为 use case 消息；
- system 根据类 use case 来实例化一个 use case 对象；
- 使用实体向这个实例化的 use case 对象发送另一个消息，消息的名称为要执行的执行场景的名称，这个消息称之为场景消息；
- 执行场景执行时需要的其它普通对象也被实例化；
- 使用实体在同普通对象交互时，也可以发送另外的 use case 消息（向 system 对象）或场景消息（向 use case 对象）。

这种以面向对象的方式具体化 use case 的方式在面向对象的软件开发方法中显得很自然，但应用这种方法的先决条件是，选择执行的场景要事先确定下来。因此当由 use case 边界范围之外的元素来确定要执行的场景时，就应该采用其它的方式。

精化是用 use case 来精化 use case，具体化是用 use case 所包含的执行实例来进行具体化工作。在软件开发过程中，应通过有效地使用 use case 的精化和具体化手段将 use case 的描述、use case 的设计和实现工作有效地联系起来。

3.1.4 组织和管理 use case。随着 use case 分析工作的不断进行，需要采用一种方式来有效地组织和管理 use case 模型。在 UML 规范中，以包（Package）的嵌套层次结构的形式来组织 use case 图。每个包中至少包含一个 use case 图，包中的内容给出了其中的每个 use case 所涉及的系统环境。通过包组织 use case，并且把软件动态设计模型中的有关部分与相关的 use case 包联系起来，就可以有效地管理 use case 的分析、设计和实现过程。

3.2 use case 的设计和实现

use case 的分析工作从系统的外部角度展示系统的功能，系统内部的对象为实现这些外部功能而要发生的相互作用的行为是由 use case 的设计和实现工作来确定的。use case 的设计和实现方式有许多，下面介绍两种方式，采用的是 UML 规范中的两种动态模型：协作图和状态图。use case 图从用户的视觉，展示了系统的外部功能，是软件需求模型的一部分；而协作图是协作的表现方式，状态图是对象行为的表现方式，它们从系统内部的角度展示系统中的对象在相互作用的过程中所发生的行为，是软件设计模型的一部分。

3.2.1 协作图。协作图中包含了对象、对象之间的关联和消息，若包含了子系统，则表明该协作图还包含了子协作图。其中的消息分为同步和异步两种类型，而且对象之间的协作可以是并发的。软件系统模型中协作图的集合构成了协作模型。采用协作模型在进行动态建模的过程中，也表明了对象之间的关系，这种对象之间的静态关系是通过动态建模来体现的。

协作模型中的一个协作图描述了系统模型中的模型元素（子系统、类）相互协作来实现一个 use case，完成系统的某个功能的过程。模型元素之间的相互协作是通过发送消息来实现的，其中包含了使用实体向 use case 发送的消息。一个消息用协作图

(2)Rational Rose 是 Rational 公司开发的一个基于 UML 的面向对象的软件建模工具。

中一个对象所属类的一个方法(method)来实现。图2中的协作图是在一个学籍管理系统中,use case学生选课的实现。其中课程对象和已开设课程对象为多个实例,表示同一个类的多个实例(多个同一类型的对象)参与这个协作过程。

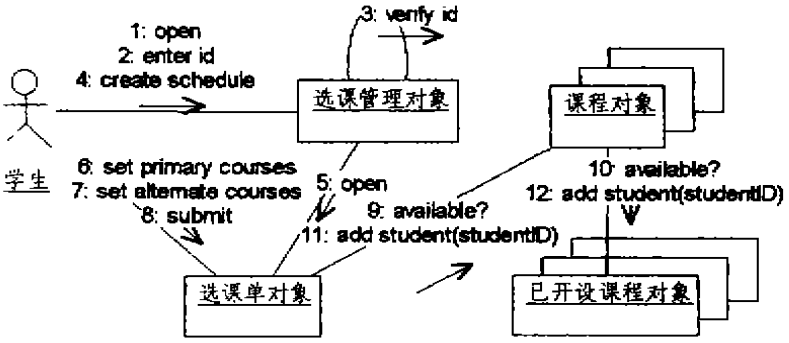

图2 用协作图来实现use case

3.2.2 状态图,状态图包含了一组状态和一组状态转移,状态转移定义了状态转换的过程,软件系统中每个活动的实体都可以建立一个状态图,用以记录系统运行时相应的活动实体的生命历程。当use case中只包含一个软件实体(类)时,就适合用状态图来实现它。

使用实体向use case发送消息,就会产生一个调用事件(call event),从而引发一个状态转换,执行相应的动作、还可以引发新的事件,进而引发新的状态转换,等等。其中,使用实体还可以引发新的调用事件。

图3是一个银行存取款的例子中银行帐户对象的状态图,其中包含了一个银行帐户对象所处的可能状态和在存取款业务中要发生的行为。

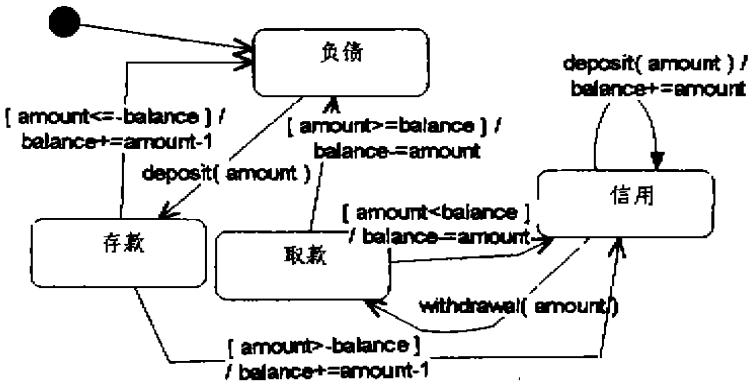

图3 用状态图来实现use case

图中,withdrawal和deposit这两个事件是当使用实体与该状态图所实现的use case交互时,由使用实体发出的请求所激发的事件。

上述两种方式的一个关键区别就是如何对待对象之间的请求(request)。协作图采用操作(operation);状态图采用信号(signal)。在UML中operation和signal均是request的子类型,详细内容请参阅文[2]。当use case涉及一个软件实体(典型情况下,该软件实体本身比较复杂)时,则采用状态图方式;而当use case涉及多个软件实体,并且要反映出各个软件实体之间的关系时,就适合采用协作图方式。另外,也可以采用其它的方式来设计和实现use case,例如,当要强调事件(或操作)的先后顺序、强调时间约束条件时,应当采用UML中的顺序图方式。有关其它的设计和实现use case的方式可参阅文[2]、[3]和[5]。在具体的use case的设计和实现工作中要根据具体情况选择适当的方式。

4 应用 use case 技术时应注意的问题

·在面向对象的软件开发方法中应用 use case 应尽量遵从面向对象开发方法的特点：以对象为中心。use case 技术的具体应用不应破坏方法的内在统一性，而要把它有机地集成到具体的开发方法中去。

·在软件生命周期中采用 use case，应该规范 use case 的定义、描述和设计。

·维持系统分析和系统设计之间的合理的、自然的分离。在分析阶段，use case 主要解决"做什么"的问题，而在设计阶段，主要解决"如何去做"的问题。

·在适当的情况下，采用协作模型来描述和实现 use case，建立系统的动态模型，往往会对软件系统的静态模型产生很有启发的意义。因此，在同等的情况下，更趋向采用协作模型。

·use case 的表示方式会对 use case 的实现方式的选择产生影响。

·在面向对象的软件开发方法中，use case 的分析和设计可以作为对象外部接口的分析和设计的依据，从而有助于信息隐藏。

结束语 use case 在软件系统的开发过程中是一种强有力的辅助开发手段，为问题的提出、问题的理解和问题的解决提供了一个重要的途径。在软件开发过程中，应当把 use case 技术有机地集成到软件开发方法中去，在软件开发的各个阶段应用 use case 技术要注意保持开发方法的特点和内在统一性。组织和管理好 use case，充分发挥 其在软件开发过程中的作用。

关于在软件系统的分析、设计和实现过程中应用 use case 技术目前已有很多方法，如何在实际的开发过程中更好地应用这些方法，还需要做进一步的研究工作。各种软件工具（特别是前期工具和软件测试工具）要为应用 use case 技术提供有力的支持。

参 考 文 献

1 Berard E V. Be Careful With Use Cases. Availble at: URL: http: // www. toa. com/pub/html/use_ case. html

2 UML Semantics, version 1. 1 (1 September 1997), The Object Management Group, doc. no. ad/97-08-04

3 UML Notation Guide, version 1. 1 (1 September 1997), The Object Management Group, doc. no. ad/97-08-05

4 Rational Objectory Process-Introduction Rational Corp. Press, 1997

5 Hurlbut R. The Three R's of Use Case Formalisms: Realization, Refinement, and Reification: [Technical Report: XPT-TR-97-06]. Expertech, Ltd., 1997

（上接第 69 页）

对处理结果的正确性负责。按领域分割保证了协调器的自治，从而降低维护费用。

协调器功能和操作的管理是其所有者的责任，协调器应该做到：

①无论数据源如何变化，协调器都要确保稳定的服务；

②不断改进，以求为客户提供更深层次的服务；

③弥合客户应用和数据源之间的概念差异；

④解决不同数据源之间格式、表达方式和范畴等方面的差异。

在大规模系统中，主要费用在于数据整合与维护，而不是完成所需功能与性能优化。协调器的整合能在多个概念层次上进行，使数据源得到充分利用。

参 考 文 献

1 Kern H. Right-sizing the New Enterprise. Sunsoft/Prentice-Hall, 1994

2 Mediators in the Architecture of Future Information Systems. IEEE Computer, 1992, 25(3)

3 Lientz B P, Swanson E B. Software Maintenance Management. Addison-Wesley, 1980

4 Value-added Mediation in Large-Scale Information Systems. In: Meersman R, Mark L, eds. Database Application Semantics. Chapman and Hall, 1997

5 The Conceptual Basis for Mediation Services. IEEE Expert, 1997, 12(5)