维普资讯 http://www.cqvip.com

精化 UML 模型*)

杨 静[1] 张明义[2] 刘志明[3]

(贵州大学计算机科学与工程学院 贵阳 550025)[1] (贵州科学院 贵阳 550002)[2]

(澳门联合国大学国际软件研究所 P. O. BOX 3058)[3]

摘 要 本文通过融合 UML 用例图、类图、顺序图和状态图,得到一个软件系统的需求模型和设计模型,给出了需求模型和设计模型的协调性条件及精化规则。这样,我们可以从软件开发的需求分析和设计阶段检查模型的协调性,通过协调地精化模型后生成代码。用这种方法,我们可在软件设计的早期阶段发现不协调问题,减少生成代码后除错所产生的代价。

关键词 UML,用例图,类图,顺序图,状态图,需求模型,设计模型,协调性,精化

Refining UML Models

YANG Jing[1] ZHANG Ming-Yi[2] LIU Zhi-Ming[3]

(Computer Science and Engineering College, Guizhou University, Guiyang 550025)[1] (Guizhou Science Academy, Guiyang 550002)[2] (International Institute for Software Technology the United Nations University, P. O. BOX3058)[3]

Abstract Requirement model and design model are obtained though integrating use-case diagram, class diagram, sequence diagram and state machine in UML. Consistency checking for these models are proposed, and the refinement rules about these models are achieved. Therefore, consistency checking for these models can be performed at requirement analysis and design stages, code can be generated by refining design model steadily. Following this way, we can find inconsistency problem of the software system at earlier stage, and reduce the cost at debugging the software system after code generation.

Keywords UML, Use-case diagram, Class diagram, Sequence diagram, State machine, Requirement model, Design model, Consistency, Refinement

1 引言

在基于 UML 的软件开发过程中,如 RUP[1],几种 UML 模型可以用来描述和分析系统开发过程中每个阶段得到的制品。在 UML 的这种多重视野的观点下,软件开发者可以把一个系统设计分解成较小规模的可控制的模块,因此软件开发者必须保证各种模型相互间在语法和语义上都是相容的。

近几年来,UML 模型的协调性检查及形式化分析是研究 UML 的热点之一[2,3,4,6,7,10],其中绝大部分工作主要是形式化 UML 的单个图(比如状态图)并仅仅处理一种图或二种图间的协调性。

本文目的是构建一个基于 UML 的软件开发的需求模型和设计模型,分析这两个模型的协调性,给出这两个模型的精化规则,通过精化模型从而构建形式良好的需求模型和设计模型。这两个模型涉及到 UML 的用例图、类图、顺序图,状态图、可用于模型融合及代码生成。

2 UML 的用例图、类图、顺序图及状态图

在 UML 中,一个类图(Class Diagram)提供以下信息:

(1)类上的静态信息和它们之间的继承关系:

-*CN*:由所有类所构成的有限集。

-*SUPER*:在集合 *CN* 上定义的直接继承关系。

(2)描述类的结构。对集合 *CN* 中的一类 *C*,用集合 $att(C)$:$\{\langle a_1, T_1\rangle, \cdots, \langle a_m, T_m\rangle\}$表示 *C* 的基本属性,这里 T_i 代表类 *C* 的属性 a_i 的类型。

(3)给出类之间联系的信息。用$\langle C_1, m_1, Ass, m_2, C_2\rangle$代表从类 C_1 到类 C_2 的有向联系 *Ass*,用$(C_1, m_1, Ass, m_2, C_2)$代表类 C_1 和 C_2 之间的无向联系 *Ass*。这里 m_1 和 m_2 仅分别指多重性,如 1|0…1|*|1‥*等等。

(4)对集合 *CN* 中的每一个类 *C*,表示类 *C* 的所有方法的集合为 *method*(*C*)。

一个类图,其形式良好的主要条件是继承关系不允许引进圈,另一方面主要是类的命名问题。这里我们不考虑多重继承。图 1 就是我们考虑的商店的自动收银(Point-of-Sale Terminal)POST 系统的类图。

一个类图定义了系统的状态空间,并且每个系统状态包含一些对象和对象之间的联系。因此,类图类似于程序中类的申明、类型及变量。一个系统状态其实就是这些变量定义良好的状态。UML 完全通过用例机制来处理需求,但没有用例规格说明的任何形式化的结构。在我们的框架下,每一个用例 U 被模似成一个用例控制类 U-controller(见图 2)。

*)国家自然科学基金项目(No. 10161005);贵州省科研基金项目(No. 3086)。**杨 静** 博士研究生,副教授,研究方向:软件工程与形式化方法;**张明义** 博士生导师,研究方向:计算机科学与人工智能;**刘志明** 联合国大学国际软件研究所研究员,研究方向:形式化方法、面向对象和基于组件的软件、实时和容错系统。

维普资讯 http://www.cqvip.com

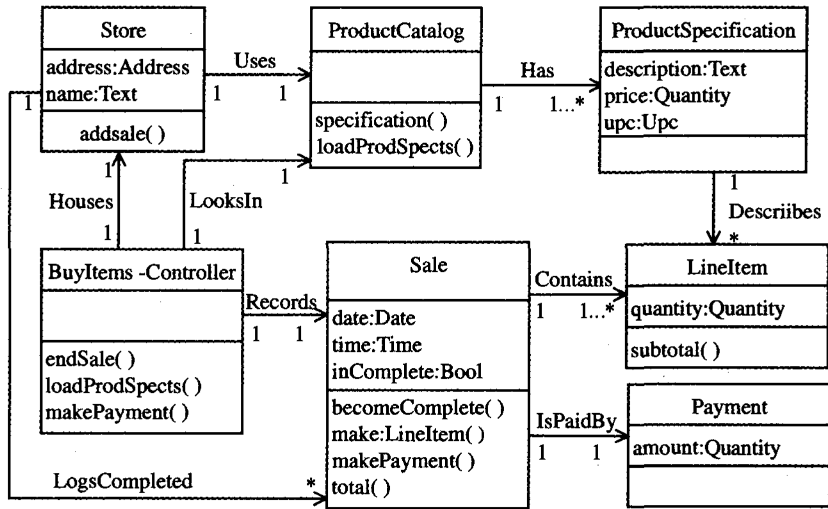

图 1 POST 系统的设计类图

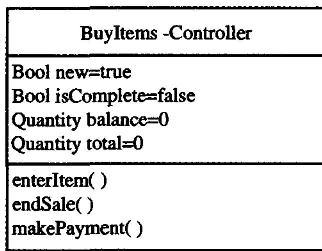

图 2 用例控制类 BuyItems

顺序图(Sequence Diagram)是用来描述为完成某项任务的对象间相互通讯的模式。顺序图(如图 3 和图 4)的基本元素是信息,一条信息具有这种形式〈Ns, $Action$, Mt, ord〉,其中

-Ns 是一个类型为 N 的对象 s,它是信息发送者。

-Mt 是信息的接收者,是一个类型为 M 的对象 t。

-$Action$ 有两种形式:$g\rightarrow t.m$ 和 $*g\rightarrow t.m$,g 是作用于信息发送者 Ns 的属性及公共变量上的布尔表达式,m 是类 M 中的一个方法,$*$ 是循环符号。

—ord 是偏序集〈Λ,≤〉的一个元素。其中集合 Λ 中的元素递归定义为

$n::=n|n.n, n\in NAT^+, NAT^+=\{n\in NAT|n>0\}$

偏序≤是标准的字典顺序。

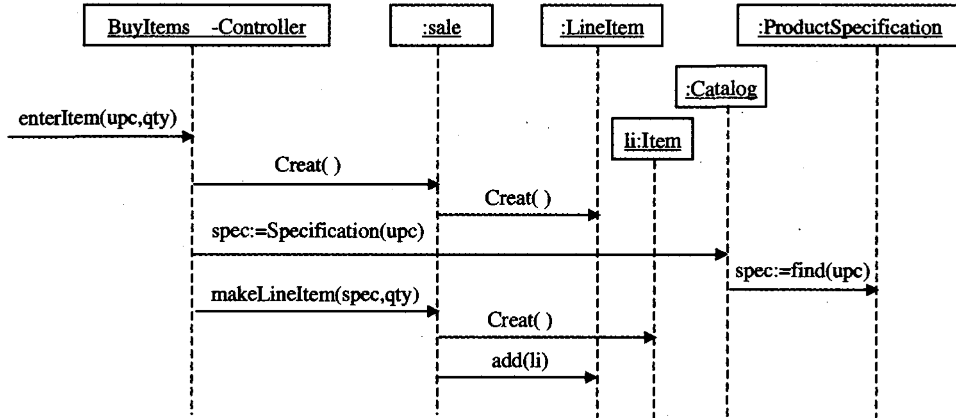

图 3 enterItem()的顺序图

顺序图就是一个满足下述条件的三元组〈Mo,m,MSG〉:

- Mo 是调用方法 m 的接收者,
- m 是类 M 中的一种方法,
- MSG 是一些信息构成的一个有限集。

对每一个类,都用一个状态图(State Machine)来刻画这个类的对象的行为模式及其方法的功能,它包括对象的状态和某些事件(如调用方法和接收信号)的传送,我们只考虑简单的状态图,也就是没有复合状态的状态图(见图 5)。一个状态图是一个四元组〈S,L,s_0,T〉,其中

①S 是一个由状态构成的有限集。

②$s_0\in S$ 初态。

③L 是同形如〈$Event$,$Action$,$Guard$〉构成的有限集,称为标签集。对标签〈$Event$,$Action$,$Gurard$〉而言,$Event$ 是该类的一个方法;$Action$ 是一些命令;$Guard$ 是这个类的属性及 $Action$ 上的局部变量上的布尔表达式,称为警卫。

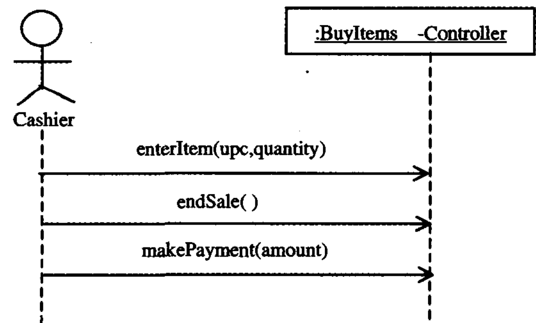

图 4 用例控制类 BuyItems 的顺序图

维普资讯 http://www.cqvip.com

④T是一个关系：$T \subseteq S \times L \times S$，通常$T$中的每一个元素称为一个切换。一个切换$\langle s,t,s' \rangle$把状态从$s$变到$s'$。此时事件*Evnet*发生了并且在状态$s$下*Guard*成立，然后执行程序*Action*。

3 需求模型的协调性检查

正如文[8]指出的一样，一个软件系统的开发周期总是从需求模型的构建开始。在UML中，需求可由概念类图和用例图来描述。概念类图是每一个类都没有方法的类图，而且类之间有的联系是没有方向。因此，需求模型*RM*应包含一些UML模型：概念类图Γ_r、一个用例图U_r、一簇用例顺序图Δ_r(每一个用例对应其用例顺序图，如图4)、一簇状态图Ω_r(每一个用例有自已的一个状态图，如图5)。如果考虑并发行为，那么一个需求模型也应包含一些活动图。在本文中，我们不考虑并发行为。一个需求模型可用一个四元组$\langle \Gamma_r, U_r, \Delta_r, \Omega_r \rangle$来表示。正如前面提到过的那样，每一个用例都可被模拟成一个用例控制类，其余类的所有属性和它们之间的联系对用例控制类而言都是可见的。用例图也提供了用例控制类之间的联系信息。如果用例U_1包含用例U_2，那么从U_1-*Controller*到U_2-*Controller*有一个联系。用例图U_r描述的是参与者和系统间的交互。

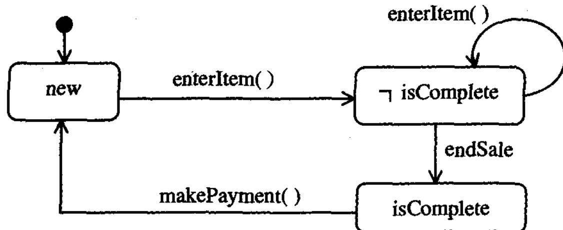

图5 用例控制类BuyItems的状态图

跟随文[8]中提出的控制模式，对用例顺序图中出现的操作，我们在相应的用例控制类中作出相应方法的申明，并且状态图Ω_T应只与用例控制类有关。用例顺序图和状态图用来描述信息传送顺序和用例控制类中方法的行为。所以，对每一个用例U，用例控制类*U-Controller*申明了在用例顺序图中出现的操作，并且类*U-Controller*中每个方法的方法体可用类*U-Controller*的状态图中唤醒该方法后的行为而得到。

从上述观点可以概括出一个需求模型如果满足下述条件便是协调的：

(1)每个用例U的顺序图中出现的信息，其m被申明成*U-Controller*中的一种方法。

(2)每个用例控制类*U-Controller*的状态图中的事件*Event*是相应的顺序图中的一条信息的方法。

(3)状态图中事件发生的秩序与顺序图中信息传送的秩序相同。

(4)类的申明部分是形式良好的。

上述条件保证了每一个在用例控制类中使用的变量或者是基本型的，或者是类图中给出的一个类，在方法体中只能使用相应类的属性。需求模型协调性保证了从概念类图实现用例图的足够信息。

4 设计模型的协调性检查

在UML，一个设计模型*DM*应包含一个设计类图Γ_d，其中所有联系都是设计联系，所谓设计联系就是有方向的联系。图1是一个设计类图；一簇对象顺序图Δ_d，保证对*U-Controller*中的每一种方法至少存在一个对象顺序图，如图3；一簇状态图Ω_d，设计类图中出现的每一个类都有一个状态图。因此一个设计模型是一个三元组

$DM = \langle \Gamma_d, \Delta_d, \Omega_d \rangle$

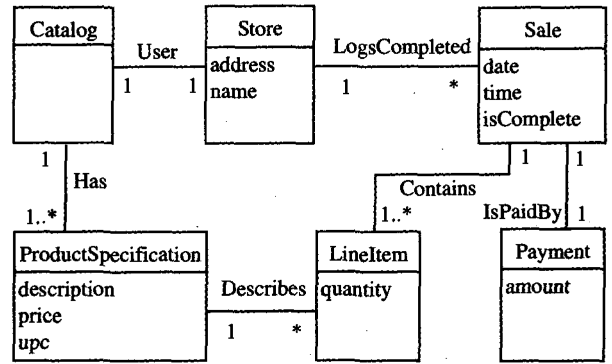

图6 POST系统的概念类图

在软件设计阶段，即在设计模型中，用例图已经被融合到类图中。在我们的框架下，对象顺序图Δ_d在设计类图Γ_d上操作。在以后的工作中我们将提供方法体的形式化规范说明，这种形式化的规范说明将保证顺序图和状态图所要求的行为得以实现。

一个设计模型$DM = \langle \Gamma_d, \Delta_d, \Omega_d \rangle$是协调的，如果满足下述条件：

(1)设计类图Γ_d与顺序图Δ_d是协调的。在顺序图Δ_d中出现的类C方法$m()$，在设计类图Γ_d中的类C中必须声明相同的方法签名；在Δ_d中出现每一条信息，在Γ_d中有相应的设计联系(用JAVA语言实现时，目标对象被申明成源对象的类中的一个属性)。

(2)设计类图Γ_d和状态图Ω_d是协调的。状态图Ω_d的每一个方法，其方法签名被申明在类图Γ_d的相应类中。

(3)顺序图Δ_d和状态图Ω_d是协调的。在状态图中触发事件发生的秩序和顺序图中信息传送的秩序是相同的。

(4)设计类图是定义良好的。

一个设计模型*DM*对于一个需求模型*RM*而言是正确的，如果设计模型的实现蕴涵需求模型的满足。

5 模型的精化规则

在UML框架下，下列精化规则可用于需求模型和设计模型：

(1)添加一个类。这对应于添加一个类到类图当中，添加这个新类的方法到顺序图和状态图中。

(2)对一个存在的类引进一个新的属性。这对应于添加一个基本类型的属性到这个类或添加一个从这个类到其它类的设计联系。

(3)引进一个继承。如果类N属性没有出现在类M和类M的所有超类中，我们可以指定M为N的直接超类。

(4)把一个属性从一个类移动到它的直接超类。如果类N所有的子类都有一个公共属性，把这个公共属性从类N所有的子类中转移到类N上。

(5)数据封装。假定类M有一个公共属性，除M的子类外，其它类的方法不能访问这个属性，我们把这个属性的可见性从公共变为保护。假定类M有一个保护属性，M的子类的方法不能访问这个属性，我们把这个属性的可见性从保护变为私有。

(6)对一个存在的类引进一个新的方法。这样一来允许我们在类图中把一个方法的签名添加到这个类，并且增加一

维普资讯 http://www.cqvip.com

个顺序图。修改相应的状态图,以配合这个新方法。

(7)在一个已申明的类中精化方法 $m()\{c\}$ 的方法体。当然,这将导致与 $m()$ 有关的子顺序图的相应替换,并且在这个类的状态图中精化,把 $m(\ )$ 当成触发事件 *Event* 的切换中的行为 *Action*。

(8)如果一个类的某种方法没有访问这个类的任何保护或私有属性,则把这个方法从这个类移动到它的直接超类。

(9)从一个类复制一个方法到它的超类。

(10)某个类的一些任务分配到和它有联系的其它类。如果这个方法体包含一个能被其它类的方法实现的某子命令,我们用唤醒后者的方法来替换这个子命令,同时精化与这个唤醒的方法有关的顺序图和状态图。

(11)删除无用的属性。一个私有属性如果未出现在其所属类的任何方法体中,则删除此私有属性;一个保护属性如果未出现在其所属类及其子类的任何方法体中,则删除此保护属性;一个公共属性如果未出现在所有类的任何方法体中,则删除此公共属性。

(12)删除无用的方法。一个方法如果未被其它类的任何方法或主程序唤醒,则删除此方法。

6 模型的例子

本节给出需求模型和设计模型的一个简单例子。

例 1 概念类图(见图 6)、用例图(此处只有一个用例控制类,见图 2)、用例顺序图(见图 4)及状态图(见图 5)形成了一个需求模型。

例 2 设计类图(见图 1)、一簇对象顺序图(见图 3、图 7 和图 8)和一簇状态图(设计类图中的每一个类都有一个状态图,此处只显示了一个状态图,见图 5)构成了一个设计模型。对这个设计模型使用精化规则得到类 *BuyItems-Controller* 中方法 *enterItem*()的设计。

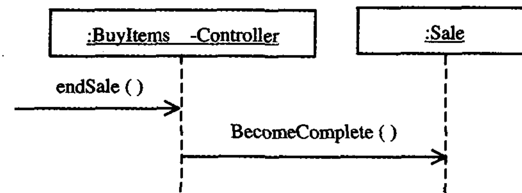

图 7 endSale()的顺序图

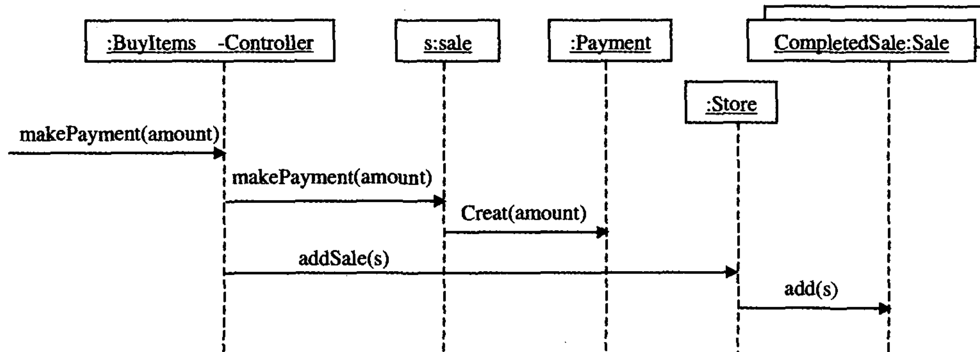

图 8 makePayment()的顺序图

(1)*BuyItems-Controller* 把创建一个新对象 *sale* 的任务分配给类 *Sale*;

(2)*BuyItems-Controller* 把使用 *upc* 匹配查找产品规格说明的任务分配给 *Catalog*,后者进一步把这个任务分配给 *ProductSpecification* 的多重对象;

(3)*Sale* 把创建一个类 *LineItem* 的对象分配给这个类。

结论 在这本文中,我们提供了融合的 UML 模型——需求模型和设计模型,讨论了这两个模型协调性的要求。软件开发需求分析阶段的每一步进化对应相应 UML 模型的进化。在开发过程中,UML 模型的每一步进化必须保证模型的协调性。模型自身的协调性主要是静态的,可通过一个算法做静态的检查,不变量可用模式检测工具检测模型纵向(即精化)的协调性主要来自语义。程序设计的基本技术和数据精化都可用于 UML 模型进化到代码生成。我们这种方法支持基于 UML 模型驱动的软件开发方法[5,9]。

在本文中,我们给出了 UML 模型的精化规则。如果接口(即某些类的方法集)被固定下来,那么这些精化规则相对于系统环境而言是可逆的。因此,完全有可能把一个设计模型变回需求模型,当然这是我们下一步的工作。将来的工作包括把活动图融合到我们的框架下分析并发行为,并用此框架开发相应的需求模型和设计模型的协调性检测的工具。我们已经发表的工作[7]中给出了设计类图和顺序图的规范性说明,但没有处理状态图。

参考文献

1 方贵宾,等译. UML 和统一过程. 北京:机械工业出版社,2003

2 Back R, Mikhajlova A, vonWright J. Class re–nement as semanticsof correct object substitutability. Formal Aspects of Computing, 2000(2):18~40

3 Tyszberowicz S, Litvak B, Yehudai A. Behavioral consistency validatyion of uml diagrams. In: Ist IEEE Internationdl Conference on Software Engineering and Formal Metods(SEFM), IEEE Computer Society, 2003. 118~125

4 Engels G, et al. A methodology for specifying and analyzing consistency of object-oriented ehavioral models. In: The Proc. FSE-10, Austria, 2001

5 Fowler M. What is the point of UML. In: Srevens P, Whittle J, Booch G, eds. ‹‹UML››2003 -The Uni–ed Modeling Language, 6th International Conference, LNCS 2863. San Fancisco, CA, USA: Springer, 2003

6 Kuester J M, Engels G, Groenewegen L. Consistent interaction of software components. In: IDPT2002, 2002

7 Yang J, Long Q, Liu Z, et al. A predicative semantic model for integrating uml models. In: Liu Z, Araki K, eds. LNCS 3407, Verlag Berlin Heidelberg, Springer, 2005

8 Larman C. Applying UML and Patterns. Prentice-Hall International, 2001

9 Mellor S J, Balcer M J. Executable UML: a foundation for modeldriven architecture. Addison-Wesley, 2002

10 Reggio G, et al. Towards a rigorous semantics of UML supporting its multiview approach. In: Hussmann H, ed. Proc. FASE 2001, LNCS 2029, Springer, 2001