ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.6, June 2009, pp.1457−1469
doi: 10.3724/SP.J.1001.2009.03455

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

UML类图中面向非功能属性的描述和检验*

张 岩[1,2+], 梅 宏[1,2]

[1](北京大学 信息科学技术学院软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

Non-Functional Attributes Oriented Description and Verification in UML Class Diagrams

ZHANG Yan[1,2+], MEI Hong[1,2]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

+ Corresponding author: E-mail: zhangyan@sei.pku.edu.cn

Zhang Y, Mei H. Non-Functional attributes oriented description and verification in UML class diagrams. *Journal of Software*, 2009,20(6):1457−1469. http://www.jos.org.cn/1000-9825/3455.htm

Abstract: Modeling for a system is a very important activity in software development. A model with high quality should not only include the description of functional attributes of the system, i.e., what the system can do, but also the description of non-functional attributes, i.e., what is the quality of the system. Although the de facto modeling approaches and tools adequately support modeling for the functional attributes, they neglect modeling for the non-functional attributes, especially, on how to integrate the description of the functional and non-functional attributes in one model and provide methods to verify some properties about the non-functional attributes. In the paper, UML Class Diagram is extended to describe the non-functional attributes by adding the model elements, i.e., the non-functional attributes notation and the constraints table. An approach is given to verify the consistency and satisfiability of the non-functional attributes in the extended UML Class Diagram. An example is used to demonstrate our proposal and a tool that supports the description and verification of non-functional attributes in UML is introduced.

Key words: non-functional attribute; UML; class diagram; model checking

摘 要: 为系统构建模型是软件开发中的一项关键活动.一个高质量的模型不仅要包含系统的功能属性,即系统能够做什么,同时还应包含系统的非功能属性,即系统的质量如何.目前,通用的建模方法和工具对功能属性建模支持良好,而对如何为非功能属性建模关注得不多,特别是如何将二者统一起来并对描述的非功能属性的有关性质进行检验.通过在 UML 类图中增加非功能属性标注和约束关系表等建模元素来扩展 UML 类图,使其能够描述非功能属性.在此基础上,又提供了对扩展 UML 类图中非功能属性的一致性和可满足性进行检验的方法.通过实例对上述

* Supported by the National Natural Science Foundation of China under Grant No.60773152 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z127 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2005CB321805 (国家重点基础研究发展计划(973)); the National Key Technology R&D Program of China under Grant No.2006BAH02A02 (国家科技支撑计划); the China Postdoctoral Science Foundation under Grant No.20070420006 (中国博士后科学基金)

Received 2008-01-31; Revised 2008-04-15; Accepted 2008-08-11

的面向非功能属性的建模和检验方法进行了说明,并介绍了相应的支持工具.

关键词: 非功能属性;统一建模语言;类图;模型检验
中图法分类号: TP311 文献标识码: A

软件开发过程是一个从抽象到具体的转换过程,即从系统较高层次的抽象模型开始逐步向实现代码的转换过程.在模型驱动的开发方法(model driven development,简称MDD)[1]中,模型作为核心制品,既是实现最终产品的一个“蓝图”,也是各软件开发活动所加工的对象.因此,系统模型的质量直接关系到最终交付使用的实际系统的质量.

一般地,将系统模型所刻画的信息分成两类,即功能属性(functional attribute,简称FA)和非功能属性(non-functional attribute,简称NFA).前者描述了系统必须提供的服务,或者系统在特定的输入或特定的场景下应该做出的反应.功能属性也常被称为能力;后者描述了对系统功能属性、系统整体性质或系统开发过程的某种约束.非功能属性也常被称为质量属性.长期以来,对建模技术和建模工具的研究主要集中在系统功能属性的描述方面.例如,学术界涌现出众多图形化或文本式的工具来支持系统功能模型的建立,如数据流图(data flow diagram,简称DFD)、实体关系图(entity-relation diagram,简称ERD)、消息序列图(message sequence chart,简称MSC)和统一建模语言(unified modeling language,简称UML)[2]等.上述这些工具从结构和行为等不同侧面对系统功能属性的描述提供了有力支持.对于系统非功能属性描述的研究目前主要集中在需求分析阶段,即提供某种手段以便在需求分析阶段能为非功能需求(non-functional requirement,简称NFR)建立模型[3−5].但是,关于在设计阶段如何将非功能属性的描述与功能属性的描述集成到同一个系统设计模型中的研究则相对不足.这就带来一个问题,虽然我们利用某种技术可以在需求分析阶段描述非功能需求,但到了设计阶段却无法在设计模型中继续体现这部分非功能需求.另一方面,随着高可信软件(high confidence software)概念的提出,用户对软件的诸如可靠性、可维护性、适应性、可修复性等非功能属性的关注越来越高,这就必然要求在软件设计阶段能够将这些非功能属性反映出来,即为系统建立的模型中应包含非功能属性的描述.同时,为了保证由系统模型生成的实际系统的质量能够满足用户需求,还应该提供对模型中非功能属性的相关性质(如可满足性,即模型中所涵盖的非功能属性与用户需求一致和一致性,即非功能属性之间不存在矛盾)的检验.这样,尽可能地在开发的早期阶段发现问题,既保证了质量又节约了成本.

基于上述考虑,本文选择一种较为通用的建模语言——UML 进行研究,并将研究重点放在系统设计过程中经常使用的UML 类图(class diagram)上.通过对系统非功能属性固有特点的分析,本文对 UML 类图进行了扩展,从而使其能够描述目标系统的非功能属性.在上述扩展的基础上,本文又给出了面向扩展的UML 类图中非功能属性的检验方法,即非功能属性的一致性和可满足性的检验.本文通过实例说明了在扩展的UML 类图中对非功能属性的描述和检验的有效性,并介绍了支持对非功能属性进行描述和检验的相关工具.

本文第 1 节分析非功能属性的特点,并给出为描述非功能属性而对 UML 类图所作的扩充.第 2 节给出如何针对扩展后的 UML 类图中非功能属性的相关性质进行检验.第 3 节利用一个实例说明如何用扩展的 UML 类图描述系统非功能属性,以及对这些属性进行检验的效果,并介绍相应的支持工具.第 4 节是相关工作的比较.第 5 节总结全文并对今后的工作加以展望.

1 UML 类图中非功能属性的描述

在 UML 类图中使用注释或简单地增加一些标记虽然也可以起到描述非功能属性的作用,但这既不能准确地刻画非功能属性,也不利于对非功能属性的检验.因此,本节首先分析非功能属性的固有特点,从中发现全面描述非功能属性所必须抓住的要素,进而在此基础上给出对类图的扩充方法,以此完成对非功能属性的描述.

1.1 对非功能属性的分析

非功能属性是对系统功能属性、系统整体性质或系统开发过程的某种约束.刻画非功能属性可以分成两部

分:一部分是非功能属性值的表达,即特定非功能属性对系统功能属性、整体性质或开发过程所施加的某种约束的表示;另一部分则是非功能属性间关系的表达.前者相对简单,由于非功能属性值相当于一种约束,因此我们可以将其视为一个逻辑表达式,当该逻辑表达式的值为“真”时,此非功能属性得到满足,否则就不满足.至于非功能属性间关系则相对复杂,通过研究,本文将非功能属性间的关系分为两类,即精化关系(refinement relationship)和约束关系(constraint relationship).

若非功能属性A_1的值无法直接度量,而需通过非功能属性A_2的值来间接度量,则称A_1与A_2之间存在精化关系,即A_1精化为A_2(或A_2精化为A_1).精化关系实质上刻画了非功能属性间的一种分解关系.一般地,系统中比较抽象的非功能属性要分解(或精化)成一些较为具体的非功能属性来处理,例如,“性能”这一非功能属性就可分解为“计算性能”和“网络传输性能”两个较具体的非功能属性;进而,“计算性能”还可以分解为“CPU速度”、“CPU数量”和“内存容量”等更为具体的非功能属性;同样,“网络传输性能”也可被进一步分解为“网络带宽”和“网卡吞吐量”等非功能属性.因此,我们说“性能”精化为“计算性能”和“网络传输性能”,其中,“计算性能”又精化为“CPU速度”、“CPU数量”和“内存容量”等.若非功能属性A不被精化为其他任何非功能属性,则称A为基本非功能属性(basic NFA);若非功能属性A被精化为其他非功能属性,则称A为导出非功能属性(derived NFA).

若非功能属性A_1的值对非功能属性A_2的值会产生影响,则称A_1与A_2之间存在约束关系,并称A_1为约束的主动方,A_2为约束的被动方.非功能属性间的约束关系既可以是单向的,即一方为约束的主动方,另一方为约束的被动方;也可以是双向的,即双方互为约束的主动方和被动方.例如:“CPU速度”和“计算性能”之间存在的是单向约束,即“CPU速度”的高与低影响着“计算性能”的高与低;而系统的可靠性(reliability)和效率(efficiency)之间存在的就是双向约束,即追求高可靠性会降低效率,追求高效率则会降低可靠性.显然,非功能属性间的精化关系蕴含了约束关系,即若两非功能属性间存在精化关系,则它们之间一定存在约束关系.但注意到,由精化关系所蕴含的约束关系永远是单向的,即若A_1精化为A_2,则A_2会影响A_1,而A_1不可能影响A_2.而对于没有精化关系的任意两非功能属性间的约束关系,则可能是单向的也可能是双向的.

精化关系可以认为是非功能属性间的一种固有关系.在任何系统中,导出非功能属性总是通过一系列的基本非功能属性来展示自己.相反,约束关系则是外界施加于非功能属性间的一种关系.在不同的环境下,同样的非功能属性,它们之间的约束关系可能是不一样的.由于这两种关系的性质有所不同,因此,在描述非功能属性时要区别对待.但不管怎样,非功能属性值以及非功能属性间的关系是描述系统非功能属性的必要成分.

1.2 扩展UML类图描述非功能属性

在类图中对非功能属性的描述分为两部分,分别为非功能属性标注(NFA notation)和非功能属性间约束关系表(简称约束关系表(constraints table)).一个非功能属性标注$NFAN$=[ID,$ValueExpression$,$ScaleType$]由3部分组成,其中:ID是该标注所描述的非功能属性的标识,每个ID均形如$NFAname_0.NFAname_1\ldots NFAname_n$,其中,$NFAname_n$为被描述非功能属性的名称;对任意的$NFAname_i$,$NFAname_{i+1}$($1\le i<n$),有$NFAname_{i+1}$精化$NFAname_i$;没有任何非功能属性精化为$NFAname_0$.ValueExpression是该标注所描述的非功能属性的值,它可以是一个具体的数值,也可以是一个逻辑表达式,前者表示该非功能属性的确切值,后者表示该非功能属性的值要满足的约束.ScaleType是ValueExpression中所涉及的数值的标度类型.不同标度类型规定了具有该标度类型的数值可参与的不同运算.这里重点关注两种运算:其一,当类图中两个有关系的元素对同一个非功能属性均会产生影响时,计算该非功能属性值的运算,称其为相关运算,记作U_d:$V\times V\to V$,其中,V是该非功能属性值的集合;其二,当类图中两个无关系的元素对同一个非功能属性均会产生影响时,计算该非功能属性值的运算,称其为无关运算,记作U_{ind}:$V\times V\to V$,其中,V是该非功能属性值的集合.例如,N_1=[$SysAvailability.SWAvailability$,0.9,$TypeOfAvailability$]中非功能属性名称SysAvailability代表系统可用性,SWAvailability代表软件可用性;0.9是非功能属性SWAvailability的值,TypeOfAvailability是0.9的标度类型.该标注的含义是:软件可用性为0.9,而软件可用性是系统可用性的精化,其值的标度类型为TypeOfAvailability.设TypeOfAvailability所对应的相关运算定义为$U_d(x,y)=x\cdot y$,无关运算定义为$U_{ind}(x,y)=\min\{x,y\}$,其含义是:当N_1标注的是类C_1的软件可用性,N_2=[$SysAvailability.SWAvailability$,0.8,$TypeOfAvailability$]标注的是类$C_2$的软件可用性,且$C_1$与$C_2$之间有关系存在(如$C_1$继承$C_2$)时,则

由C_1和C_2组成的子系统的软件可用性(即SWAvailability的值)为$U_d(0.9,0.8)=0.9\cdot0.8=0.72$;若$C_1$与$C_2$之间不存在任何关系(即$C_1$与$C_2$是相互独立的),则由$C_1$和$C_2$组成的子系统的软件可用性为$U_{ind}(0.9,0.8)=\min\{0.9,0.8\}=0.8$.

在非功能属性标注中,ID不仅标识了非功能属性,同时还刻画了非功能属性间的精化关系.在一个模型中,利用所有非功能属性标注中的ID,我们可以建立起该模型所涉及的全部非功能属性(包括导出非功能属性和基本非功能属性)间的精化关系.如第 1.1 节所言,非功能属性间除精化关系外,还存在约束关系.因此,为全面刻画非功能属性还需要利用约束关系表来描述非功能属性间的约束关系.一个约束关系表$CRL=(c_{ij})_{n\times n}$实际上就是一个$n\times n$维的方阵,其中,n是模型中所涉及的非功能属性的个数;c_{ij}是一个逻辑表达式,它代表了第i个非功能属性与第j个非功能属性间的约束.设NFA_i和NFA_j分别为第i个和第j个非功能属性的名称,V_{NFA_i}代表NFA_i的值.特别地,当$c_{ij}::=\text{TRUE}$时,表示NFA_i和NFA_j间无约束关系;当$c_{ij}::=V_{NFA_j}=f(V_{NFA_i},V_{NFA_s},V_{NFA_{s+1}},\ldots,V_{NFA_t})$时,表示$NFA_j$与$NFA_i,NFA_s,NFA_{s+1},\ldots,NFA_t$的值之间满足函数关系$f$,其中$f$可视为$NFA_j$的一个度量模型.例如,可用性与非功能属性MTTF(平均失效时间)和MTTR(平均修理时间)的约束关系就可以用度量模型*Availability*=MTTF/(MTTF+MTTR)来表示.

图 1 是扩展 UML 类图元模型的核心部分,该图中灰色部分均为标准 UML 类图元模型中的元素.由于标准 UML 类图元模型中的 Constraint 类拥有属性 name 和 sepecification,它们正好对应了非功能属性的名称和值,因此,在扩展 UML 类图元模型中让 NFA 类继承 Constraint 类.NFA 类中的 superNFA 属性表达了非功能属性间的精化关系.ScaleType 类继承自标准的数据类型(DataType 类),但增加了相关运算(dependableOp)和无关运算(independableOp)的定义.

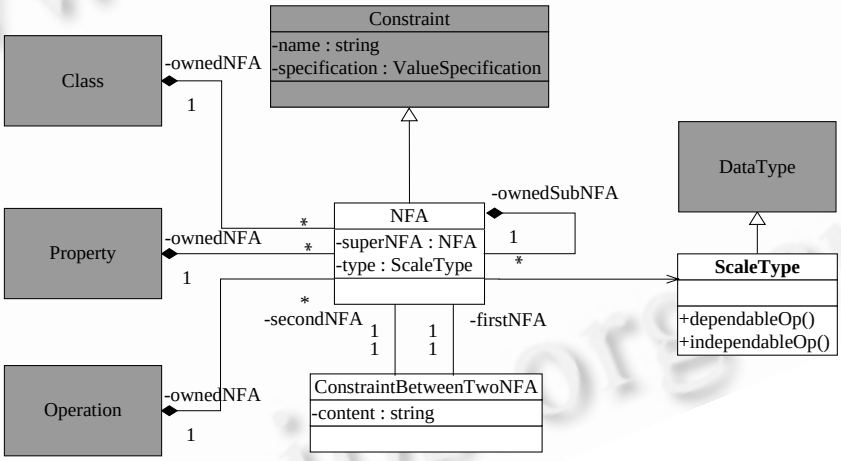
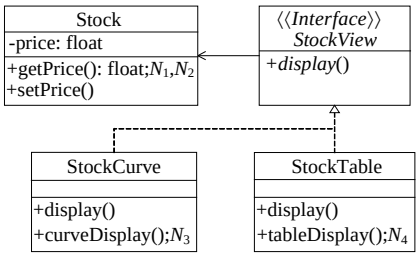


Fig.1 Metamodel of the extended UML class diagram

图 1 扩展 UML 类图的元模型

系统的非功能属性通常是通过具体的功能属性来展示和实现自身的.具体到类图来说,非功能属性是与类中的操作和属性紧密相关的.例如,“操作openfile(filename)的最大延时是 0.5s”、“属性password是为提高安全性而设”等,都说明非功能属性(延时和安全性)是与类中的操作和属性绑定在一起的.因此在扩展UML类图中,非功能属性标注的表示法为:若一个非功能属性标注对应了类中的某个操作(或属性),则在该操作(或属性)的定义结束后,接着写出该非功能属性标注,非功能属性标注与操作(或属性)的定义之间用“;”相连;多个非功能属性标注之间用“,”分隔.当类图用于较高抽象层次的设计时,类中不一定给出操作和属性,那么当类中无操作和属性时,则将非功能属性标注记在类名之后,且两者之间用“;”相连.图 2 给出了一个扩展UML类图的实例.此类图描述了一个股票价格显示系统.该系统可以按照曲线图(StockCurve类)和表格(StockTable类)两种形式显示股票价格(stock类).系统中涉及 3 个非功能属性,即性能(performance)、延时(latency)和可用性(availability),其中,Latency是Performance的精化,两者间的单向约束关系P表示延时越长则性能越差.非功能属性标注N_1表明getPrice()操

作的延时不超过 1s,N_2表明getPrice()操作的可用性为 0.98,N_3表明curveDisplay()操作的可用性为 0.95,N_4表明tableDisplay()操作的可用性为 0.96.



N_1=[*Performance.Latency*,(0,1],*TypeofLatency*]
N_2=[*Availability*,0.98,*TypeofAvailability*]
N_3=[*Availability*,0.95,*TypeofAvailability*]
N_4=[*Availability*,0.96,*TypeofAvailability*]

$$CT=\begin{array}{c} \\ Performance \\ Latency \\ Availability \end{array}\begin{array}{c} \begin{array}{ccc} Performance & Latency & Availability \end{array} \\ \begin{bmatrix} \text{TRUE} & \text{TRUE} & \text{TRUE} \\ P & \text{TRUE} & \text{TRUE} \\ \text{TRUE} & \text{TRUE} & \text{TRUE} \end{bmatrix} \end{array}$$

$$P ::= V_{Lat} > V'_{Lat} \rightarrow V_{Perf} < V'_{Perf}$$

Fig.2 An instance of the extended UML class diagram

图 2 扩展 UML 类图的实例

开发人员利用扩展UML类图为系统建模的过程大致如下:根据用户的需求规约建立系统的功能模型,即标准的 UML 类图.与此同时,将需求规约中与功能相关的非功能需求抽取出来,并转换成相应的非功能属性标注加入到系统的功能模型中.由于一般情况下在用户的需求规约中,功能需求与非功能需求是耦合在一起的,例如响应用户请求的时间不超过 1 秒,其中,响应用户请求是功能需求,而响应时间不超过 1 秒就是非功能需求,因此,开发人员在完成功能设计的同时就在模型中描述非功能属性是非常自然和方便的.当开发人员书写非功能属性标注时,首先要为该标注所描述的非功能属性确定标识.确定标识的过程实质上就是开发人员对系统中所涉及的非功能属性进行分解和归类的过程.一个基本非功能属性会对哪些导出非功能属性产生影响、一个导出非功能属性又通过哪些基本非功能属性来体现,都是在这一步完成的.其次是给出标注中描述的非功能属性的值.非功能属性值一般有如下几个来源:来自于需求规约、来自于以往的度量经验、来自于设计经验等.最后,还要为标注中描述的非功能属性指定标度类型.当所有非功能属性标注书写完成后,从这些标注的 ID 中可以导出约束关系表的结构,开发人员进而需填写约束关系表以定义非功能属性间的约束关系.至此,建模活动结束.需要特别说明的是,对非功能属性的分解是一项重要的活动,它关系到能否确定各非功能属性间的精化关系,并进一步关系到对非功能属性的相关检验能否正确执行.但如何分解非功能属性已超出本文研究范围,本文重点关注的是给定非功能属性及其间的各种关系(包括精化关系和约束关系),能否用所给方法全面而准确地描述它们,其间没有信息丢失且不产生歧义,以利于对这些非功能属性进行检验.

在扩展的UML类图中,非功能属性标注和约束关系表配合在一起,相当于为系统模型中所涉及的非功能属性建立了模型,而该非功能属性模型与系统(行为)模型又是结合在一起的.因此,用扩展后的类图描述的系统模型既包含了非功能属性描述又包含了功能属性描述,并且可以在该模型中对相关非功能属性进行一系列的检验工作.

2 UML 类图中面向非功能属性的检验

上一节介绍了为描述系统非功能属性而对UML类图所作的扩展以及如何用扩展UML类图来描述系统的非功能属性.为了能对扩展UML类图中所描述的非功能属性进行检验,首先需要对该类图进行形式化,为此,本文引入类依赖图(class dependency graph,简称CDG)[6]来完成这一任务.在此基础上,进一步给出对扩展UML类图中非功能属性的一致性和可满足性进行检验的方法.

2.1 UML类图的形式化表示

记扩展UML类图CD中所有类的集合为C_{CD}.CD的类依赖图$CDG_{CD}=\langle X_{CD},E_{CD}\rangle$是一个二元组,其中,$V_{CD}$是结点集合,且存在双射$f:X_{CD}\leftrightarrow C_{CD}$,即$CDG_{CD}$中每个结点代表CD中的一个类.此外,每个结点均带有一个标记,该标记是此结点代表的类中所有非功能属性标注构成的集合.$E_{CD}\subseteq X_{CD}\times X_{CD}$是边集合,对于任意的$(x,y)\in E_{CD}$,若$x\neq y$,

则有$(x,y)\neq(y,x)$,即类依赖图中的边均是有向的.(x,y)的含义是结点x所代表的类依赖于结点y所代表的类.类依赖图中的依赖路径是由类依赖图中结点组成的序列$x_1x_2\ldots x_n$,其中,$(x_i,x_{i+1})\in E_{CD}$,$1\leq i\leq n-1$.若依赖路径$p=x_1x_2\ldots x_n$中不存在任何相同的结点,则称p为简单依赖路径.对于任意的简单依赖路径p,若不存在简单依赖路径p',使得p是p'的子序列,则称p是一条最大简单依赖路径.

由于类图与类依赖图有很好的对应关系,因此很容易从类图CD得到其对应的类依赖图CDG_{CD},其转换规则如图 3 所示[6].

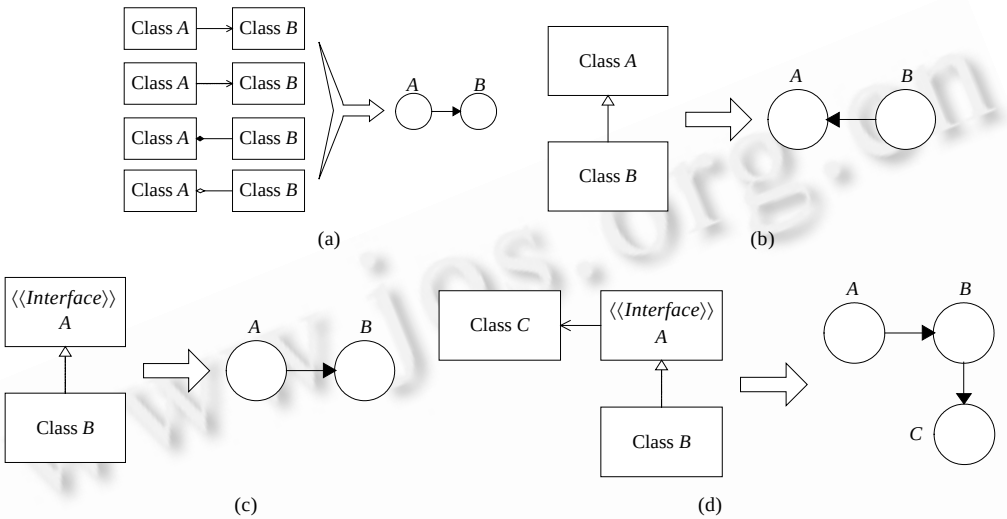


Fig.3 Rules of the transformation from UML class diagram to CDG

图 3 UML 类图到类依赖图的转换规则

图 4 给出了图 2 中所示类图的类依赖图.其中,结点StockView,StockCurve,Stock构成了一条简单依赖路径,同时它也是一条最大简单依赖路径.结点StockCurve上的标记$\{N_3\}$为图 2 中StockCurve类中所有非功能属性标注构成的集合,其他结点均如此.

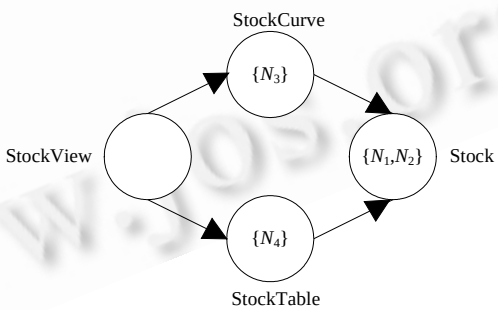


Fig.4 An example of the transformation from a UML class diagram to a CDG

图 4 类图到类依赖图的转换示例

2.2 面向非功能属性的检验

2.2.1 被检验的性质

对于模型中的非功能属性,我们重点关心两方面的性质,即一致性和可满足性.所谓一致性是指模型中所描述的任意两非功能属性均满足它们之间所存在的约束关系.也就是说,非功能属性的描述不存在相互矛盾的情况.所谓可满足性是指由模型中所描述的非功能属性而推知的系统具有的某项非功能属性是满足用户要求的.对一致性的检验,可以保证模型中对系统非功能属性的描述是合理的.对可满足性的检验,保证了为实现系统所

做的设计在特定非功能属性方面是可以满足用户需求的.

由于导出非功能属性依赖于基本非功能属性,因此,要检验导出非功能属性间的一致性就必须通过精化它的基本非功能属性来计算出其具体值之后才能判断.另一方面,对于导出非功能属性的可满足性的检验也需要通过计算精化它的基本非功能属性值来进行.因此,一致性检验和可满足性检验是交织在一起进行的.下一节将给出检验的方法.

2.2.2 检验方法

检验方法的核心是要通过分散在不同非功能属性标注中的非功能属性值合成出每个非功能属性的一个最终值.该最终值代表了类图所描述的系统在该非功能属性上的值,即系统的非功能属性值.进而,可以根据非功能属性最终值来检验非功能属性的一致性和可满足性.

检验是在类图所对应的类依赖图上进行的,检验方法分为如下几步:

首先,以最大简单依赖路径为单位,分别计算各非功能属性的值.在这一步中,需按类之间的依赖关系,根据同一非功能属性的多个值(它们出现在不同的非功能属性标注中)计算出该非功能属性的一个确定值.我们将这一计算过程称为对非功能属性值的依赖路径内更新.注意到,依赖路径内更新的方法与该非功能属性值的标度类型相关,即需要用到特定标度类型所对应的相关运算U_d.以图 4 为例,本步骤要分别对最大简单依赖路径StockView,StockCurve,Stock和StockView,StockTable,Stock中出现的非功能属性Performance和Availability计算其确定值.例如在StockView,StockCurve,Stock中有Availability的两个标注(N_2和N_3),且两个标注中Availability的值是不同的.因此,需要利用定义在TypeofAvailability上的相关运算U_d计算该最大简单依赖路径内的Availability的确定值,即U_d(0.98,0.95).

其次,根据上一步得到的各非功能属性值来计算系统非功能属性值,称这一计算过程为对非功能属性值的依赖路径间更新.注意到,依赖路径间更新的方法同样与非功能属性值的标度类型相关,即需要用到特定标度类型所对应的无关运算U_{ind}.仍以图 4 为例,根据上一步可以分别得到两条最大简单依赖路径内的Availability的值,即U_d(0.98,0.95)和U_d(0.98,0.96),本步骤要根据这两个值计算整个股票价格显示系统的Availability的值,即U_{ind}(U_d(0.98,0.95),U_d(0.98,0.96)).

在以上两步的计算过程中,需要检查各非功能属性值是否满足约束关系表中的约束,一旦有约束不能得到满足,则一致性检验失败.若一致性检验成功,则将系统非功能属性值与用户理想的非功能属性值进行比较,若达到了理想值,则可满足性检验成功,否则可满足性检验失败.

记非功能属性名称的集合为$\mathcal{A}$,基本非功能属性名称的集合为$\mathcal{A}_b$,导出非功能属性名称的集合为$\mathcal{A}_d$.对于任意$a\in\mathcal{A}$,非功能属性a的值记为V_a.约束关系集合$C=\{\psi:\mathcal{V}\times\mathcal{V}\to\{\text{TRUE,FALSE}\}|\exists V_x,V_y\in\mathcal{V}.\psi(V_x,V_y)=\text{TRUE}\wedge x\in\mathcal{A}\wedge y\in\mathcal{A}\}$,其中,$\mathcal{V}$为非功能属性值的集合.用户关于非功能属性$a$的要求可表示成谓词函数$R_a(V_a)$.对类图$S$中的非功能属性进行检验的方法如下:

(1) 求出类图S的类依赖图$G_S=\langle X_S,E_S\rangle$,记$G_S$中每个结点$x\in X_S$上的标记为$L_x$,对任意非功能属性标注$l\in L_x$,记$V(l)$为$l$中的非功能属性值.

(2) 对每个非功能属性$a\in\mathcal{A}$,计算G_S中每个结点x对应的a的值,记为$V_a(x)$.若L_x中无a的标注,则$V_a(x)$不存在;若L_x中存在a的标注,则$V_a(x)$计算方法如下:

a) 当$L_x=\{l\}$时,$V_a(x)=V(l)$;

b) 当$L_x=\{l_1,l_2,\ldots,l_n\}$时,若$l_1$是$a$的标注,则$V_a(x)=U_d(V(l_1),V_a(x'))$,其中,$x'$为标记为 $L'_x=\{l_2,\ldots,l_n\}$的结点;若l_1不是a的标注,则$V_a(x)=V_a(x')$.

(3) 若存在$V_a(x_i),V_b(x_j),a,b\in\mathcal{A},x_i,x_j\in X_S,\psi\in C$,使得$\psi(V_a(x_i),V_b(x_j))$=FALSE,则终止,一致性检验失败,输出$x_i$,$x_j$,$a$和$b$;否则继续.

(4) 按深度优先遍历G_S,找出所有最大简单依赖路径并构成集合P.

(5) 对每个非功能属性$a\in\mathcal{A}$,计算P中每个最大简单依赖路径p对应的a的值,记为$V_a(p)$.若p中每个结点x的$V_a(x)$均不存在,则$V_a(p)$也不存在;否则,$V_a(x)$计算方法如下:

a) 当$p=x_1$时,$V_a(p)=V_a(x_1)$;

b) 当$p=x_1x_2\ldots x_n$时,若$V_a(x_1)$不存在,则$V_a(p)=V_a(p')$,其中,$p'=x_2\ldots x_n$;若$V_a(x_1)$存在,则$V_a(p)=U_d(V_a(x_1),V_a(p'))$.

(6) 若存在$V_a(p_i),V_b(p_j),a,b\in\mathcal{A},p_i,p_j\in P,\psi\in C$,使得$\psi(V_a(p_i),V_b(p_j))$=FALSE,则终止,一致性检验失败,输出$p_i$,$p_j$,$a$和$b$,否则继续.

(7) 对每个非功能属性$a\in\mathcal{A}$,计算P对应的a的值,记为$V_a(P)$,计算方法如下:

a) 当$P=\{p\}$时,$V_a(P)=V_a(p)$;

b) 当$P=\{p_1,p_2,\ldots,p_n\}$时,若$V_a(p_1)$不存在,则$V_a(P)=V_a(P')$,其中,$P'=\{p_2,\ldots,p_n\}$,若$V_a(p_1)$存在,则$V_a(P)=U_{ind}(V_a(p_1),V_a(P'))$.

(8) 若存在$V_a(P),V_b(P),a,b\in\mathcal{A},\psi\in C$,使得$\psi(V_a(P),V_b(P))$=FALSE,则终止,一致性检验失败,输出$a$和$b$;否则继续.

(9) 对每个非功能属性$a\in\mathcal{A}$,计算类图S对应的a的值,记为$V_a(S)$,计算方法如下:

a) 当$a\in\mathcal{A}_b$时,$V_a(S)=V_a(P)$;

b) 当$a\in\mathcal{A}_d$时,若$V_a(P)$不存在,则$V_a(S)=V_a(S)=M(V_{a_1}(S),V_{a_2}(S),\ldots,V_{a_n}(S))$,其中,$a$仅依赖于$a_1,a_2,\ldots,a_n$;若$V_a(P)$存在,则$V_a(S)=U_{ind}(V_a(P),M(V_{a_1}(S),V_{a_2}(S),\ldots,V_{a_n}(S)))$.

(10) 若存在$V_a(S),V_b(S),a,b\in\mathcal{A},\psi\in C$,使得$\psi(V_a(S),V_b(S))$=FALSE,则终止,一致性检验失败,输出$a$和$b$;否则继续.

(11) 若$R_a(V_a(S))$=TRUE,则可满足性检验成功;否则,可满足性检验失败.

在上述检验方法中,步骤(2)、步骤(5)、步骤(7)、步骤(9)分别是根据非功能属性标注计算每个类、每条类的依赖链、由每条类的依赖链组成的类图的某一非功能属性值;步骤(3)、步骤(6)、步骤(8)、步骤(10)是对非功能属性一致性的检验;步骤(11)是根据最终计算得到的整张类图所对应的某一非功能属性值,对该非功能属性的可满足性进行检验.

2.2.3 讨 论

UML 类图中的关系包含了元素间的交互关系,但不是每一个关系都代表交互关系.此外,即便类图中的某个关系是代表交互的,它也仅仅指出两个元素间存在交互,而并没有明确指出交互是由何操作引起,以及其中涉及哪些属性.而在本文给出的方法中,非功能属性描述是与类中的属性和操作相关联的,检验又是基于类之间的依赖关系进行的.因此,检验结果的精确程度受到了类图所含信息量以及这些信息的精确性的影响.实际上,目前的检验结果是最悲观情况的反映.也就是说,当某项非功能属性的可满足性检验不成功时,实际中该类图所对应的系统并不一定不能满足该非功能属性;但当可满足性检验成功时,则说明该类图所对应的系统一定可以满足该非功能属性.但是,这并不会降低本文所给出方法的有用性.众所周知,在软件开发早期,如设计阶段,对系统的细节信息的掌握是较少的,因此,基于这些有限信息来准确推测系统的整体属性势必存在困难.但利用本文所给出方法,可将系统中可能存在问题的地方缩小在一定范围之内.这样,设计人员可以将注意力集中在系统中最可能出现问题的这部分上,通过进一步收集与之相关的准确数据并加以分析,从而准确确定问题所在.本文所给出方法将发现问题的时间提前到了设计阶段,并将确定问题出现的范围尽可能地缩小.与等到软件开发末期掌握了足够多的系统信息后再来发现问题相比,本文所给出方法无论是在降低成本上,还是在保证质量上均要优于前者.

另一方面,UML类图实质上是一种概念模型,它只定义了系统中的元素和元素间的关系.因此,针对扩展UML图中的与交互行为相关的非功能属性(如延时、CPU占用等)所进行的检验就不如在顺序图(sequence diagram)和状态图(state machine diagram)[2]上来进行的效果要好.但是,本文所给方法不是不能对这些与行为相关的非功能属性进行描述和检验,只是利用文中方法在UML类图中对这些与行为相关的非功能属性进行检验所得到的结果,其有用性和准确性会降低.既然UML提供了各种图形,从不同侧面描述同一系统,我们也不应寄希望于将所有非功能属性的检验工作均放在同一张图中进行.因此,对于那些与行为相关的非功能属性,我们应

将其放到系统行为模型中加以描述并进行检验.此外,我们的经验是:在扩展UML类图中对一些抽象层次比较高的非功能属性(如可靠性、可维护性、安全性等)进行建模和检验,比对抽象层次比较低的非功能属性(如内存使用量、功耗等)进行建模和检验的效果要好.

3 实例研究

本节以一个机载防撞系统(airborne collision avoidance system,简称 ACAS)作为实例,对本文的面向非功能属性的描述和检验方法进行分析、说明.图 5(a)给出了 ACAS 的扩展 UML 类图.一架飞机(Aircraft 类)有 4 个关键部件,分别为:飞行监测器(Detector 类)——负责监测邻域内其他飞机的飞行状态(包括经纬度、速度、高度和角度等)并向冲撞管理器(CollisionManager 类)汇报;冲撞管理器——负责根据本机及邻机飞行状态来预测是否会发生碰撞,如果可能发生碰撞,则计算为避免碰撞所应采取的飞行动作并发出警告;飞行控制器(Controller 类)——负责接受从冲撞管理器发来的避免碰撞飞行动作,以此来调整本机的飞行状态;警报器(Alarm类)——负责接受从冲撞管理器发来的警告,并按一定级别发出某种类型的警报.图 5(a)中共有非功能属性标注 14 个(N_1~N_{14}),涉及 4 个非功能属性,即可靠性(reliability)、平均失效时间(MTTF)、性能(performance)和延时(latency).其中,MTTF精化Reliability且两者间的约束关系为V_{Rel}=exp(−t/V_{MTTF}),即平均失效时间和可靠性满足此度量模型;Latency精化Performance且两者间的约束关系为 $V_{Lat} > V'_{Lat} \to V_{Perf} < V'_{Perf}$,即延时越长,性能越差.以N_1和N_2为例对非功能属性标注的含义再作一解释:N_1表明adjustStatus操作的平均失效时间为 1 000(小时);N_2表明adjustStatus操作的延时不超过 0.3s.设TypeofMTTF和TypeofLat上定义的相关运算和无关运算分别为

$$U_d^{MTTF}(x,y)=U_{ind}^{MTTF}(x,y)=\min\{x,y\},U_d^{Lat}(x,y)=U_{ind}^{Lat}(x,y)=\max\{x,y\}.$$

图 5(b)是由图 5(a)导出的类依赖图G_{ACAS},其中有最大简单依赖路径 3 条(p_1~p_3).现按上节所给检验方法对ACAS的类图进行检验.

- 首先计算G_{ACAS}中每个结点所对应的各非功能属性值.以x_1为例,有

  $V_{MTTF}(x_1)$=min{980,1000}=980,$V_{Lat}(x_1)$=max{(0,0.3],(0,0.4]}=(0,0.4],

  $V_{Rel}(x_1)$和$V_{Perf}(x_1)$不存在.其他结点的情况可以类推.
- 接着计算各最大简单依赖路径所对应的各非功能属性值.以p_1为例,因为当$p'=x_5x_1$时,

  $V_{MTTF}(p')$=min{$V_{MTTF}(x_5)$,$V_{MTTF}(x_1)$}=min{1090,980}=980,$V_{Lat}(p')=V_{Lat}(x_5)$=(0,0.6].

  所以最终有$V_{MTTF}(p_1)$=980,$V_{Lat}(p_1)$=(0,0.6).其他最大简单依赖路径的情况可以类推.
- 第 3 步计算由p_1~p_3构成的最大简单依赖路径集合P对应的各非功能属性值,有

  $V_{MTTF}(P)$=940,$V_{Lat}(P)$=(0,1.2].
- 最后计算 ACAS 的各非功能属性值,有

  $V_{MTTF}(ACAS)$=940,$V_{Lat}(ACAS)$=(0,1.2],$V_{Rel}(ACAS)$=exp(−t/940),

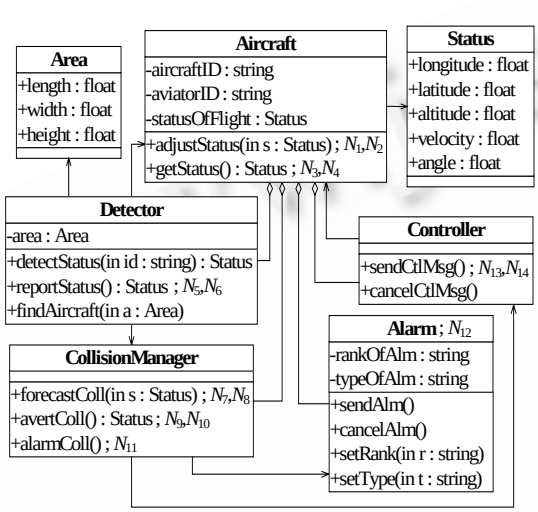
  $V_{Perf}(ACAS)$不存在.在检验过程中没有发现不一致现象,即一致性检验成功.

现在用户给出要求:ACAS 在 24 小时内的可靠性为 0.999.因为 exp(−24/940)≈0.975,所以可满足性检验失败,即当前系统在最坏情况下无法满足用户关于可靠性的需求.通过对 ACAS 的系统模型的分析,我们进而可以发现,造成 ACAS 可靠性不能满足用户需求的关键原因是冲撞管理器中对避免碰撞动作的计算(avertColl 操作)的MTTF 质量属性太差.因此,要提高 ACAS 的可靠性,则需要首先改善冲撞管理器中 avertColl 操作的 MTTF 值.

可靠性和性能是软件系统的两项极为重要的非功能属性,特别是在上述的实例中更是如此.因此,本节所给实例仅涉及了对这两项非功能属性的分析.对于其他非功能属性,如可维护性、适应性和安全性等,利用本文所给方法同样可以进行描述和检验.

我们开发了支持本文提出的在类图中描述非功能属性及对非功能属性的一致性和可满足性进行检验的工具.该工具是在北京大学软件研究所研发的支持 UML2.0 的建模工具 JBOO5.0 的基础上扩展而成的.如图 6(a)所示,该工具分 5 个部分:模型树窗口(model tree)、缩略图窗口(outline)、画板(pallete)、图形编辑窗口(graph editor)和特性编辑窗口(properties editor).用户可以在图形编辑窗口中通过拖拽画板中的建模元素,来构建所需

的 UML 类图.当选定 UML 类图中某个特定的类之后,在特性编辑窗口中会列出该类的相关特性,如类的名称(name)、是否为抽象类(isAbstract)及所含属性(ownedAttribute)和操作(ownedOperation)的集合等.若用户想为类中的某个操作关联非功能属性标注,只需点击 ownedOperation,并从操作列表(operations list)中选择相应的操作,然后选择该操作所对应的非功能属性标注集合(NonFunctionalAttribute),再在弹出的窗口中添加(NFAs list)和编辑(NFAs editor)相应的非功能属性标注即可(如图 6(b)所示).为类中的属性关联非功能属性标注与此类似.约束关系表的定义是通过在图形编辑窗口中单击鼠标右键,并从列表中选择 Constraints Table 项来完成的.用户可以利用工具中预定义的约束关系(在下拉列表中)来定义该表,也可以自定义约束关系(如图 6(c)所示).在模型树窗口中选择相应的 UML 类图并单击鼠标右键,然后选择列表中的 verify 项,即可以对该类图中的非功能属性进行检验.用户可以根据需要选择要检验的非功能属性的性质(如图 6(d)所示).目前,工具中只提供对一致性和可满足性两种性质的检验.检验结果会在特性编辑窗口的位置以文本形式显示.同时,违反一致性和可满足性的非功能属性会以高亮的形式在类图中标出.

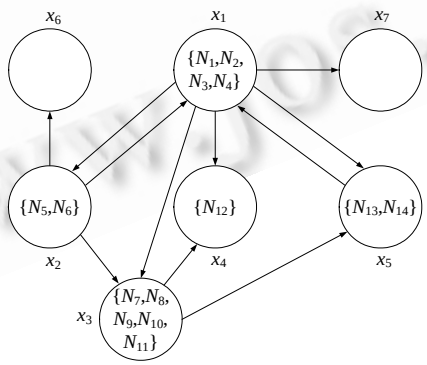


N_1=[*Reliability.MTTF*,980,*TypeofMTTF*]
N_2=[*Performance.Latency*,(0,0.3],*TypeOfLat*]
N_3=[*Reliability.MTTF*,1000,*TypeofMTTF*]
N_4=[*Performance.Latency*,(0,0.4],*TypeOfLat*]
N_5=[*Reliability.MTTF*,1100,*TypeofMTTF*]
N_6=[*Performance.Latency*,(0,0.5],*TypeOfLat*]
N_7=[*Reliability.MTTF*,950,*TypeofMTTF*]
N_8=[*Performance.Latency*,(0,1.2],*TypeOfLat*]
N_9=[*Reliability.MTTF*,940,*TypeofMTTF*]
N_{10}=[*Performance.Latency*,(0,1.0],*TypeOfLat*]
N_{11}=[*Reliability.MTTF*,960,*TypeofMTTF*]
N_{12}=[*Reliability.MTTF*,1150,*TypeofMTTF*]
N_{13}=[*Reliability.MTTF*,1090,*TypeofMTTF*]
N_{14}=[*Performance.Latency*,(0,0.6],*TypeOfLat*]

$$CT_{ACAS}=\begin{array}{c} \begin{matrix} \textit{Rel.} & \textit{MTTF.} & \textit{Perf.} & \textit{Lat.} \end{matrix} \\ \begin{bmatrix} \text{TRUE} & \text{TRUE} & \text{TRUE} & \text{TRUE} \\ P_1 & \text{TRUE} & \text{TRUE} & \text{TRUE} \\ \text{TRUE} & \text{TRUE} & \text{TRUE} & \text{TRUE} \\ \text{TRUE} & \text{TRUE} & P_2 & \text{TRUE} \end{bmatrix} \end{array} \begin{matrix} \textit{Rel.} \\ \textit{MTTF.} \\ \textit{Perf.} \\ \textit{Lat.} \end{matrix}$$

P_1::=V_{Rel}=exp($-t/V_{MTTF}$); P_2 ::= $V_{Lat} > V'_{Lat} \rightarrow V_{Perf} < V'_{Perf}$
Rel.: Reliability; *Perf.*: Performance; *Lat.*: Latency

(a) Extended UML class diagram of ACAS
(a) ACAS 的扩展 UML 类图



x_1: Aircraft
x_2: Detector
x_3: CollisionManager
x_4: Alarm
x_5: Controllor
x_6: Area
x_7: Status

$p_1=x_5x_1x_2x_3x_4$
$p_2=x_2x_3x_5x_1x_7$
$p_3=x_3x_5x_1x_2x_6$

(b) CDG G_{ACAS} of ACAS
(b) ACAS的类依赖图G_{ACAS}

Fig.5 Extended UML class diagram and CDG of ACAS
图 5 ACAS 的扩展 UML 类图和类依赖图

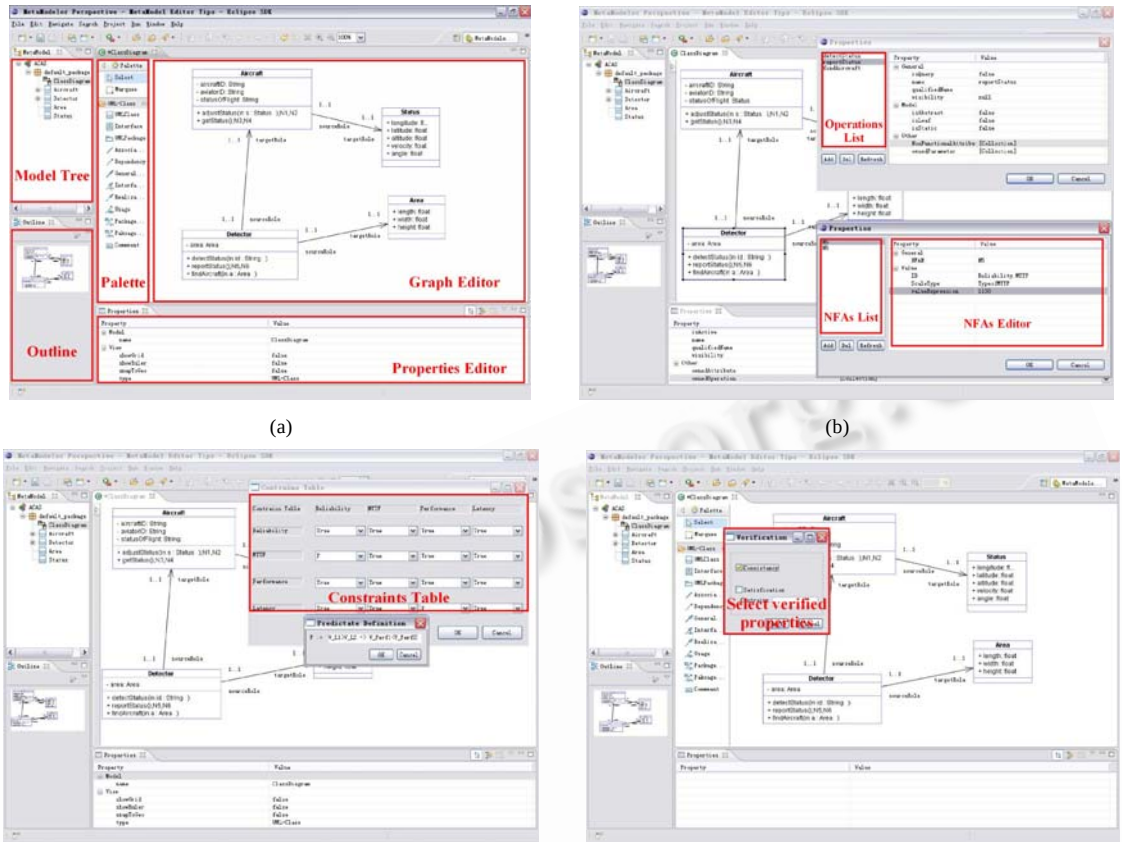


Fig.6 A support tool for the description and verification of NFAs in UML class diagram

图 6 支持 UML 类图中面向非功能属性描述和检验的工具

4 相关工作

本文的工作涉及两个方面:一是为非功能属性建模,二是对模型中的非功能属性的相关性质进行检验.以下就从这两个方面对相关的研究工作进行比较.

首先,在非功能属性建模方面,早期的建模方法和工具(如数据流图、UML1.0 等)重点放在功能属性的描述上,而对如何在模型中刻画非功能属性关注得不多.随着人们对软件质量的要求不断提高,研究人员才开始研究如何为已有的建模方法和工具增加描述非功能属性的能力,所采用的方法可分为如下两类:

一类是提供一种可描述非功能属性的方法,用以支持单独地为非功能属性建立模型[4,7].这样,系统的功能属性模型和非功能属性模型是分开的.然后,通过在两者之间建立一定的对应关系再将它们关联起来[8,9].这类方法存在的问题是:非功能属性与功能属性的刻画不在同一个模型中,这样的设计过程很不自然.我们知道,系统模型是从用户需求导出的,而用户需求中非功能属性的陈述与功能属性的陈述经常是紧密结合在一起的.因此,开发人员在为系统构建功能模型的同时,就为非功能属性也构建起模型是一种既方便又合理的设计过程.此外,该方法不便于对系统非功能属性进行检验.因为非功能属性的检验只能在非功能属性模型上进行.一旦在非功能属性模型中发现问题,则必须通过对应关系映射到功能属性模型中相应的位置才能确定具体问题所在,这样很不方便.最后,文献[7,9]中所给出的非功能属性描述方法均不是图形化的,与本文所给出的基于UML的方法相比,其可理解性和易用性都要差一些,这也影响了对它们的使用.

另一类是通过扩展UML从而增加对非功能属性的刻画能力[5,10−12].这类研究存在的问题是:模型中非功能

属性的描述多是以类似注释的形式出现.它们在形式上虽然与功能属性描述出现在同一模型中,但如何解释这些非功能属性,特别是它们与功能属性的关系,均未给出令人满意的解答.因此,在上述工作中均有意或无意地忽略了如何对模型中非功能属性进行检验的研究.这类工作中最有代表性的是UML2.0 中对描述服务质量和容错特性所做的扩充[11].出于OMG的特殊身份的原因,文献[11]中给出的是一种刻画非功能属性的大而全的一般性方法.可以说,此方法可以适用于对任何领域任何非功能属性的描述.但文献[13]也指出了这一扩充的不足:一方面,要通过两步(即先实例化QoS Catalog中的类模板,由此导出系统所在应用域的Quality Model;然后再实例化Quality Model中的QoS Constraints和QoS Values,并用它们在系统模型中描述非功能属性)才能建立起系统模型,这样既能增加用户负担,又能降低模型的可读性;另一方面,在这一扩展中忽略了对度量数据来源、精确度和时间表达式等与描述非功能属性相关的特性的刻画.

其次,在非功能属性检验方面,最为相关的是对模型检验(model checking)的研究.模型检验是软件研究的一个分支领域,旨在通过对系统的模型进行检查,从而在软件开发的早期就发现软件设计中存在的问题.目前,模型检验技术主要用于验证系统的功能属性,如无死锁性(free dead block)、活性(liveness)、安全性(safety)等[14],而对于非功能属性验证的研究则主要集中在诸如时间和资源一类的基本非功能属性上[15,16].这类方法的优点是精确.从理论上讲,只要设计模型中存在问题,则通过检验就一定能够发现,并能指出出错的位置.但由于该类方法过于强调检验的精确度,使得其在可用性方面受到影响:首先,状态空间爆炸问题就是影响模型检验在实际中应用的首要问题.尽管已有一些解决办法,但面对现实中遇到的越来越复杂的系统,仍需专门设计一些更为有效的检验算法来处理实际遇到的应用问题;其次,对于度量模型不确定,或者根本就不存在统一度量模型的非功能属性(如可维护性)的检验,目前的模型检验方法还无法处理.

本文所给出的方法则弥补了上述方法的不足:首先,将非功能属性描述与功能属性描述集成在同一模型中;其次,对非功能属性的描述不采取大而全和面面俱到的做法,而是只针对非功能属性的本质特征进行描述,并提供对模型中非功能属性进行检验的方法支持;最后,在检验的有效性和可用性两方面进行适当权衡,使得所给检验方法既适用于多数非功能属性,又使得检验结果在一般情况下是有效的.

最后,还有一部分工作是研究在需求分析阶段如何为非功能需求建模,如文献[3]通过扩展特征模型的方法来描述和组织非功能需求,并基于知识库来识别和精化非功能需求.与这部分工作相比,本文主要研究的是在软件设计过程中如何为非功能属性建模.这里的非功能属性是非功能需求在设计模型中的对应物.此外,一般在需求阶段很少对非功能需求的性质进行检验.本文则为设计阶段提供了对系统模型中非功能属性的相关性质进行检验的方法.

5 总结及展望

本文在UML类图的基础上进行扩展,通过增加非功能属性标注和约束关系表,使得类图可以同时描述系统的功能属性和非功能属性.此外,还提供了对扩展 UML 类图中非功能属性的一致性和可满足性进行检验的方法,并通过实例说明了如何用扩展UML类图为非功能属性建模,以及对模型中非功能属性的有关性质进行检验的方法.同时,还介绍了支持上述方法的相关工具.

未来将在以下两个方面进一步深化该项研究工作:(1) 为 UML 活动图、顺序图等刻画系统行为的模型增加描述非功能属性的能力,并提供针对非功能属性的检验方法;(2) 对除一致性和可满足性以外的模型中非功能属性的其他性质(如可优化性)进行检验.

致谢 感谢匿名审稿人对本文提出的宝贵而中肯的意见和建议.这些意见和建议对本文的修改和提高均起到了极大的帮助作用.感谢闫研和冯超对文中支持工具的研发所给予的帮助.感谢柳毅、张乐和麻志毅对文中元模型和实例的设计和修改所给予的帮助.

References:

[1] Selic B. The pragmatics of model-driven development. IEEE Software, 2003,20(5):19−25.

[2] Object Management Group. Unified model language (UML): Superstructure version 2.0. OMG Document: formal/05-07-04, 2005. http://www.omg.org/docs/formal/05-07-04.pdf

[3] Sun LS, Huang G, Sun YC, Chen HJ, Mei H. Towards modeling non-functional requirements in feature models. Computer Engineering and Science, 2006,28(A2):139−141 (in Chinese with English abstract).

[4] Mylopoulos J, Chung L, Nixon BA. Representing and using nonfunctional requirements: A process-oriented approach. IEEE Trans. on Software Engineering, 1992,18(6):483−497.

[5] Cysneiros LM, Leite JCSP. Using UML to reflect non-functional requirements. In: Stewart DA, Johnson JH, eds. Proc. of the 2001 Conf. of the Centre for Advanced Studies on Collaborative Research (CASCON 2001). New York: IBM Press, 2001. 202−216.

[6] Baudry B, Traon YL, Sunyé G. Testability analysis of a UML class diagram. In: Williams AD, ed. Proc. of the 8th IEEE Symp. on Software Metrics (METRICS 2002). IEEE Computer Society, 2002. 54−63.

[7] Rosa NS, Cunha PRF, Justo GRR. Process^{NFL}: A language for describing non-functional properties. In: Jr. Sprague RH, ed. Proc. of the 35th Annual Hawaii Int'l Conf. on System Sciences (HICSS-35 2002). IEEE Computer Society, 2002. 282−291.

[8] Wada H, Suzuki J, Oba K. A model-driven development framework for non-functional aspects in service oriented grids. In: Dini P, Ayed D, Dini C, Berbers Y, eds. Proc. of the 2nd IEEE Int'l Conf. on Autonomic and Autonomous Systems (ICAS 2006). IEEE Computer Society, 2006. 30−38.

[9] Franch X, Botella P. Putting non-functional requirements into software architecture. In: IEEE, eds. Proc. of the 9th Int'l Workshop on Software Specification and Design (IWSSD'98). IEEE Computer Society, 1998. 60−67.

[10] Jürjens J. UMLsec: Extending UML for secure systems development. In: Jézéquel JM, Hussmann H, Cook S, eds. Proc. of the 5th Int'l Conf. of the Unified Modeling Language (UML 2002). LNCS 2460, Berlin, Heidelberg, New York: Springer-Verlag, 2002. 412−425.

[11] Object Management Group. UML profile for modeling quality of service and fault tolerance characteristics and mechanisms. OMG Adopted Specification, ptc/04-09-01, 2004. http://www.omg.org/docs/ptc/04-06-01.pdf

[12] Object Management Group. UML profile for schedulability, performance, and time specification. OMG Adopted Specification, formal/05-01-02, 2005. http://www.omg.org/docs/formal/05-01-02.pdf

[13] Object Management Group. A UML profile for MARTE, Beta 1. OMG Adopted Specification, ptc/07-08-04, 2007. http://www.omg.org/docs/realtime/07-03-14.pdf

[14] Edmund M, Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: The MIT Press, 2000.

[15] Hu J, Yu XF, Zhang Y, Zhang T, Li XD, Zheng GL. Checking component-based embedded software designs for scenario-based timing specifications. In: Yang LT, Amamiya M, Liu Z, Guo M, Rammig FJ, eds. Proc. of the 2005 IFIP Int'l Conf. on Embedded and Ubiquitous Computing (EUC 2005). LNCS 3824, Berlin, Heidelberg, New York: Springer-Verlag, 2005. 395−404.

[16] Hu J, Li XD, Zheng GL, Wang CH. Modelling and analysis of power consumption for component-based embedded software. In: Zhou XB, Sokolsky O, Yan L, Jung ES, Shao ZL, Mu Y, Lee DC, Kim D, Jeong YS, Xu CZ, eds. Proc. of the 2006 IFIP Int'l Conf. on Embedded and Ubiquitous Computing (EUC 2006). LNCS 4097, Berlin, Heidelberg, New York: Springer-Verlag, 2006. 795−804.

附中文参考文献:

[3] 孙连山,黄罡,孙艳春,陈泓婕,梅宏.特征模型中非功能需求建模初探.计算机工程与科学,2006,28(A2):139−141.

张岩(1974－),男,山东武城人,博士,CCF 高级会员,主要研究领域为软件工程,形式化方法,模型驱动开发方法,软件度量.

梅宏(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程与软件工程环境,软件复用和软件构件技术,分布对象技术.