元模型可度量性及度量方法研究

马浩海[1,2], 麻志毅[1], 吉 哲[1], 杨国东[1], 张 乐[1]
(1. 北京大学信息科学与技术学院, 北京 100871; 2. 内蒙古大学计算机学院, 内蒙古呼和浩特 010021)

摘 要: 随着 UML 这样的通用建模语言的广泛使用, 人们开始更多地利用元模型和元建模做为软件设计和开发的手段之一. 首先分析了元模型体系结构的特点及相关的包括元模型度量在内的元信息管理的要求. 以 OMG 的四层元模型体系结构为例, 对其核心成分 UML 元模型的争论一直都是热点问题, 这提供了对其度量的必要性; 元概念和 OO 度量的特点为对其评估提供了可行性依据. 其次, 在引入稳定性和设计质量的量化度量策略并进行了 UML 的五个元模型版本的度量实践之后, 给出了度量结果的评估及与相关主观性研究的对比. 最后, 对元模型度量方法的研究给出了应用展望.

关键词: 元模型; 可度量性; 度量方法; UML
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2004) 12A-2114-04

A Study of Measurability and Metrics on Meta-Models

MA Hao-hai[1,2], MA Zhi-yi[1], JI Zhe[1], YANG Guo-dong[1], ZHANG Le[1]
(1. *School of Electronics Engineering and Computer Science, Beijing University, Beijing 100871, China*;
2. *School of Computer Science, Inner Mongolia University, Hohhot, Neimenggu 010021, China*)

Abstract: Meta models and meta modeling are becoming popular as one of techniques in the main steam of software development, resulting from the fact that general modeling languages such as UML become ubiquitous. Firstly, this paper analyzes the features of meta model architecture and corresponding requirements, including especially meta model assessment, of managing meta information. Taking OMG's four level meta model architecture as an example, UML meta model, as the core part of the architecture, is the focus of debating among researchers and practitioners. The situation proves the necessity of UML meta models measurement. The hallmarks in terms of meta and OO metrics provide the feasibility of the evaluation. Secondly, a quantitative method, measuring the stability from the internal perspective and design quality from external perspective, is applied to assess 5 UML meta models. The results of the assessment are then analyzed in comparison with related subjective comments on UML evolution. The promising application of the method is presented as a conclusion.

Key words: meta model; measurability; metrics; unified modeling language (UML)

1 引言

描述应用系统中运行时实体及其关系的信息组成了系统的模型, 而定义和表示这些模型的模型称之为元模型. 元模型定义了系统建模时用到的元信息, 它包括所有的元概念(建模构造物)的语法、语义及约束信息. 将表示和操作这些元信息的能力称为元建模[1,2]. 例如, 在面向对象(Object Oriented, OO)范型当中, 应用系统按照类、方法、对象等构造物, 以及这些构造物之间的关系如关联、继承等来建模. 在此种语境下, OO 元建模的基本思想就是利用一个更高层的对象模型(元模型)来建模对象模型(应用系统模型)的所有元素. 每个模型元素可以看作是对应元模型构造物的实例, 则模型可以看作元模型的实例. 如果把元模型也看作一个简单的对象模型的话, 那么描述元模型的语言称之为元元模型, 依次类推(称之为元循环), 形成一种元模型体系结构[3], 见图 1. 这种层次的元模型体系结构提供了对元信息的综合组织和开放的定义方法. 例如, 模型可以通过元模型的构造物进行扩展, 元模型又可以通过元元模型的构造物进行扩展.

虽然元建模可以让开发者有能力定义应用领域的概念种类, 但是元模型和元建模长久以来被认为是专属于理论研究者或者工具商的武器, 而不是主流软件开发的一部分[4]. 这种情况正在随着 UML(Unified Modeling Language)这样的通用可视化建模语言的普及而发生改变. 这种语言通用易扩展的特性鼓励了用户按照自己的需求来扩展或者剪裁建模语言(元模型)自身.

为了在元模型体系结构下进行有效的元信息管理, 也有必要提供对元模型体系结构进行操作的设施. 当前人们主要关注于: 元信息定义; 元信息维护; 定义、存储和评估元信息中的实体关系; 和元信息交换等[5]. 但是, 对元信息自身质量的度量仍然是空白. 元信息自身质量的好坏, 一方面关系到其下层利用元信息构造物进行建模的能力和精确性, 另一方面对元信息自身的可复用性, 可扩展能力等也有着很大的影响. 本文以元模型体系结构的典型实例-对象管理组织 OMG(Object Manage Group)的四层元模型结构为例, 对其核心成分的 UML

收稿日期: 2004-09-02; 修回日期: 2004-11-22
基金项目: 国家重点基础研究发展规划项目计划(973 计划, No. 2002CB312003); 国家自然科学基金(No. 60125206、60233010)

元模型的变化性和设计质量进行度量.

层次	描述和实例	典型的构造物
元 …元无模型	定义用来描述底层的语言	适当的元 …元构造物来建模底层的实体
…	…	…
元元模型	定义元建模语言, 给出用于描述和解释元模型的结构和语义和构造物, 元元模型的例子如 MOF, CDIF, OIM	例如实体、关联、模块和语义约束
元模型	定义建模语言给出可以应用于定义模型的构造物, 元模型的例子如 RM-ODP, UML, CWM	描述一个模型元素的定义由什么组成, 例如类、接口、方法和属性
模型	定义应用系统的结构和语义, 给出组成应用系统的实体的类型, 模型的例子如一个银行应用系统的模型	表示特定的类、接口、方法和属性定义, 数据类型、关联等
实例层	模型元素的实例, 例如建模的应用系统的运行时实体	对象、方法、属性、数据值、引用等

图 1 元模型体系结构

2 元模型度量

2.1 四层元模型体系结构

虽然图 1 的元模型体系结构理论上可以向上无限扩展. 但实际应用中, 层数却应该是固定的. 其中的关键是如何定义最顶层的元模型. 一般有两种策略: 一是将最顶层定义为由精确的、一致的元模型"原语"[2], 即领域的本体概念组成[6]; 另一种方法用最顶层的元模型的子集来定义其自身, 如 OMG 的四层元模型体系结构[7], 见图 2.

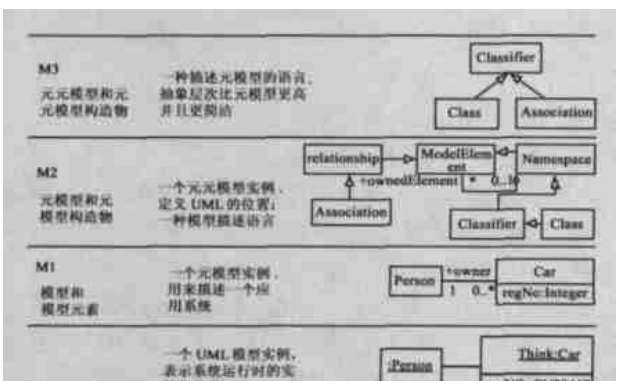


图 2 OMG 的四层元模型体系结构

图 2 中, 最顶层即元元模型层形成了整个体系结构的基础. 该层主要的职责是定义描述元模型的语言. 该层一般被称为 $M3$ 层, MOF(Meta Object Facility) 即是 OMG 定义的一种元元模型. 一个元元模型可以定义多个元模型, 并且比它所描述的元模型更加紧凑. 位于 $M2$ 层的元模型是元元模型的一个实例, 也就是说, 元模型中的每一个元素都是元元模型中一个元素的实例. 元模型的例子有 OMG 的 UML 和 CWM(Common Warehouse Meta model), 他们都被认为是 MOF 的实例. 元模型的职责是定义一种建模语言, 给出用于创建模型的构造物. 特别需要注意的是, OMG 希望元元模型和相关的元模型共享一些公共设计思想和元模型构造物, 从而利用 UML 语言的子集来定义自身乃至元元模型. 模型是元模型的实例, 位于 $M1$ 层. 模型层的主要职责是描述领域的需求, 即, 允许用户对广泛的问题域进行建模, 比如软件系统、业务过程等. 元模型体系结构的底层是 M0 层, 它包含了定义在模型层的模型元素的运行时实例.

2.2 可行性

在四层元模型体系结构中, UML 元模型扮演着最为重要的角色, 因为它直接提供了用户利用建模机制描述问题域能力的大小. 从语法的视角观察 UML 元模型, 它是由诸如元类、元属性、元关联这样的构造物形成的抽象语法图构成的. 抽象语法图本身也可以看作是一种(元) 类图, 其原因可以归结为: (1) 抽象语法图是由 MOF 中的元素进行描述的. 前面已述 MOF 其实也是 UML 的一个子集, 它有限的元元级构造物可以构造出的 UML 元模型只能是一种(元) 类图[8~12]. (2) 一个模型是否是"元"的, 其实是相对于另外一个模型扮演的角色不同而已[6], 一个元模型相对于模型和一个模型相对于其用户实例从机理上是一样的. 在规范文[7] 中也提到, 四层元模型本身就是一个可以递归应用的过程, 在一种情况下的元模型完全可以看作另一种情况下的模型. 基于此, 我们完全有理由把 UML 元模型做为一种面向对象的设计模型加以度量.

一般地, 基于类的设计始终是 OO 范型的核心[13], 因此, 对 OO 系统类图的度量也是研究的焦点. 这也使得在对 OO 系统进行度量时可以忽略与实现有关的细节, 从而把度量技术应用到软件开发生命周期的早期阶段, 如分析时或者设计时[14,15]. 这一点也恰恰吻合了 UML 元模型的抽象语法类图简洁、与实现无关的特点.

2.3 必要性

除了前述元信息管理的必要需求之外, 对 UML 元模型进行度量和评估也是 UML 自身发展的需求. UML 在 1997 年首次由 OMG 以正式规范的形式发布, 也结束了所谓的 OO 方法之争. 其后它快速成长为一种事实上的标准建模语言[16]. 但是, 如同所有软件一样, UML 在这 7 年时间里也经历了若干次的版本迭代演化过程. 而每一次的迭代都会产生一个新设计或者增强了的 UML 元模型. 这归因于 UML 不断融合新技术, 精化语义和表示法, 以及迎合更充分的使用需求和修订以前的错误. 另一方面, 当一个新版本的 UML 被发布之后, 升级建模工具以适应新的 UML 规范和测试先前版本 UML 生成的模型和新版本 UML 的兼容性问题, 都会造成很大的成本和时间开销. 再加上本身开发和修订 UML 规范的巨大成本(如 UML2.0 的升级过程经历了四年之久仍没有完全结束), 这一切都说明了有必要对 UML 元模型自身质量进行理解和刻画.

一些 UML 的制订者已经撰写了一系列的文章, 例如文[16~19], 来总结 UML 的历史版本并且对新版本(主要是 UML2.0) 做出期望. 然而, 至今还没有一种客观的方法对 UML 元模型进行评估. 这种评估不但要求能够量化的确保 UML 和软件工业的最新发展保持一致[20], 而且可以对 UML 的版本演化进行量化分析以便控制和预测未来版本的开销. 这样就需要对 UML 元模型的评估提供一种度量的方法, 特别地, 该方法可以有效地识别出 UML 元模型的结构上的瑕疵以及设计

质量的下降问题, 为元模型的设计提供可信的建议.

3　元模型度量方法

3.1　方法概述

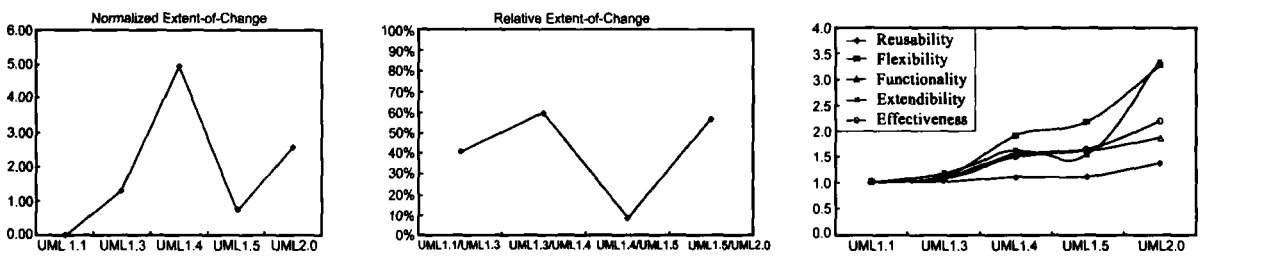
文献[21]中总结了两种看待面向对象度量技术的视点: 一是内部视点, 即, 度量数据按照定义清晰的规则从系统中实体的属性进行收集和计算; 另一种称为外部视点, 即, 通过度量来预测和评估系统开发的质量特性. 从这两种视点出发, 我们从众多的 OO 度量方法中借鉴改编了两种直观有效的策略来作为 UML 元模型度量的基础. 在试验中, 本方法应用到了 UML 五个版本的规范中的元模型上, 包括版本 1.1、1.3、1.4、1.5 和 2.0. UML 1.2 被忽略是因为它没有对 UML 做出任何显著的技术上的提高[16]. 按照内部视点, 本方法对 UML 元模型的稳定性度量借鉴了文[22]和文[23]的方法. 前者从体系结构的层次上进行度量, 后者从独立的类的角度上进行度量. 我们的方法从两种对应的层次上都实现了对 UML 元模型的评估. 首先, 我们需要根据 UML 元模型的特点定义一组度量准则, 如 ANS, AWF, AAP, NAC, NCC, 和 NEK 等(参见文[24]). 之后, 将这些度量准则应用到五个版本的 UML 元模型上并得到一组度量值. 最后, 基于这些值, 我们分别计算(1)标准变化幅度(normalized extent of change), 该值反映了元模型在体系结构上的稳定性; (2)相对变化幅度(relative extent of change), 在各个独立元类的层次上计算相连两个版本的 UML 元模型的变化性. 两种变化幅度(extent of changes)可以彼此互补并作为结果一致性校验的手段.

按照外部视点, 对 UML 元模型设计质量的评估改编自文[14]的 OO 设计质量评估层次模型方法. 虽然对软件质量还没有统一的定义, 但是很多质量评估的方法都是基于这样一种假设: 从产品内部特性可以得到产品外在质量属性的合理的结论[25]. 本文的方法也基于这样的假设, 利用上述稳定性度量中得到的元模型内在度量准则值, 按照文[14]中的计算公式, 可以得出六个元模型质量属性: 功能性(functionality)、有效性(effectiveness)、可理解性(understandability)、扩展能力(extendibility)、可复用能力(reusability)和灵活性(flexibility)的评估结果.

以上评估方法的具体执行过程和度量值可以参考文[24]. 本文下面仅仅对度量结果及其反应出来的 UML 元模型的稳定性和设计质量加以分析.

3.2　度量结果分析及对比

从度量结果图 3 可以看出, 无论是量化的稳定性还是设计质量评估上, 从 UML1.1 到 UML 1.3 的元模型都没有太大的变化, 这个结果和文[16]的观点一致.



(a) 5 个版本元模型的标准变化幅度　(b) 相邻版本元模型的相对变化幅度　(c) 五个版本元模型质量属性变化曲线(可理解性被忽略)

图 3　UML 元模型度量结果图示

从图 3 的(a)和(b)看出二图具有很大的相似性, 说明虽然度量方法不同, 但是结果却是相似的. 两张图都说明了一件事情: 从 UML1.3 到 1.4 是迄今所有元模型版本中发生了最大变化的一次修订过程, 而不像人们普遍认为的这应该是 UML2.0 的修订过程. 相对比, 从 UML1.4 到 1.5 是至今最小的一次对元模型的更新. 其它相邻两个版本的元模型修订幅度对比的大小应该是从 UML1.3/UML1.4 到 UML1.5/UML2.0 依次下降.

由此, 我们可以认为 1.4 版本是 UML 一次并不太成功的发布. 从度量结果文[24]和规范内容文[9,10]可以看出, 在这次修订中, 很多元模型体系结构上的约束都被修订者放松或者忽略了. 例如, 整个元模型的继承层次树在 UML1.4 中竟然有三个根节点之多, 其中竟然还有一个构造物 LinkEndData 也是根节点之一, 这显然是不合理的. 在元模型结构中过多的使用耦合和聚集也是造成体系结构稳定性下降的一个原因. 另外, 由于 UML1.4 第一次引入了动作语义的内容, 它也给整个元模型体系结构上的完整性造成了破坏(这一点在 2.0 中已经得到了很好的解决). 不幸的是, UML1.5 对 1.4 并没有给出多少有意义的修补, 相反过多的使用了元类之间的聚集造成了元模型的扩展能力比 UML1.4 还要差(见图 3(c)). 当然, 我们也应该承认 UML1.4 在设计质量上相对于 1.3 有了很大的提高, 而后者对于 1.1 版的提高却很有限. 这也和文[18]和文[19]评论的对 1.4 版在功能上的增强相一致.

UML2.0 拥有一个更清晰却更复杂的元模型[12], 这应该归因于它试图满足最大多数建模者需求的结果[17]. 但是从图 3(c)看出, UML2.0 在设计质量上相比以前的版本确实有了显著的提高. 元模型的扩展能力和灵活性提高了三倍多, 功能型和有效性提高了近两倍. 尽管复用能力提高不大, 但是从图中可以看出所有版本的复用能力都是近似的, 这意味着 UML 自身并不能涵盖所有的应用领域(这项任务应该留给制订更多的外廓(Profile)来解决).

基于对度量的结果的观察, 我们可以推论出 UML2.0 基本上满足了文[19]和文[18]中提到的期望, 即, 获得适度规模的体系结构、强化和统一已有的特性(例如, 将动作语义无缝的集成到了整个 UML 的元模型中了). 这也造就了一个更灵活和更易扩展的 UML(见图 3(c)), 它对避免"语言膨胀病症"[19]和"第二系统病症"[16,18,26]是至关重要的.

4　结语

本文从元模型体系结构和元建模的特点出发, 研究了元

模型和模型在角色上的相对性. 由此, 使得元模型的度量有了可行性. 借鉴于模型的稳定性和设计质量的度量方法并被应用到 UML 元模型的度量上之后, 本方法得到了比较可信的度量结果. 这种度量方法对于控制和预测类似于 UML 这样的元模型的演化方面的意义在于:

首先, 从对元模型度量上得到的信息对于指导元模型的演化和进一步开发出新的度量技术都是非常有价值的. 例如, 迄今为止指导有效和高效的开发 UML 元模型的信息是很少的. 本文提出的度量方法和结果为今后 UML 的修订过程中如何分配人力物力资源, 如何关注设计质量并保持与先前版本的稳定性上都具有明确的指导意义.

其次, 元模型的度量可以做为是否在元模型中融入新的技术增补的一种判断准则. 例如, 对 UML 试图进行扩展的呼声一直都很大. 那么做为一个最简单的策略就是在 UML 的修订中要剔除那些影响了元模型稳定性和设计质量的增补内容. 而且, 做为 UML 的修订者, 基于度量结果, 可以判断合适的 UML 新版本的发布时机, 避免不符合评估要求的元模型的发布.

第三, 元模型的度量方法可以由建模工具商或者利用元模型建模的用户使用. 在他们接受一个新版本的元模型之前, 按照度量策略加以评估. 基于评估的结果, 工具商或者用户可以预测在升级元模型之后, 由此造成的工具升级或者对已建模系统一致性校验的开销多少. 这样他们有了是否接受元模型更新的决策依据.

最后, 以 UML2.0 为例, 在文[7]中提供了两种扩展机制, 即, 用户既可以通过 Profile 机制来定义 UML 的新的方言, 也可以利用元模型的一阶扩展机制来定义 UML 语言家族的新成员(元模型). 这样不可避免的导致出现众多的元模型. 而元模型的评估方法, 因此有可能被扩展为一种强有力的工具来度量元模型家族.

参考文献:

[1] Mili H, Pachet F, Benyahia I, Eddy F. Metamodelling in OO-OOPSLA'95 workshop summary[A]. In Proceedings of the 10th annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'95)[C]. Austin, TX USA, October 1995. 105-110.

[2] Odell J. Metamodeling[A]. In OOPSLA'95 Workshop on Metamodeling in OO[C]. Austin, TX USA, October 15, 1995.

[3] Costa F M. Thesis: Combining metainformation management and reflection in an architecture for configurable and reconfigurable middleware[D]. Lancaster: Lancaster University, 2001.

[4] Atkinson C, Hendersorr Sellers B, Kuhne T. To meta or not to metathat is the question[J]. Journal of Object-Oriented Programming, 2000, 13(8): 32-35.

[5] Crawley S, Davis S, Indulska J, McBride S, Raymond K. Meta information management[A]. Proc. 2nd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'97)[C]. Canterbury, United Kingdom, July 1997.

[6] Alhir S S. Extending the Unified Modeling Language[EB/OL]. http://home.earthlink.net/~salhir, 1999.

[7] UML 2.0 Infrastructure Specification[S]. OMG Document: pt/03-09-15.

[8] UML 1.1[S]. OMG Document: ad/97-08-04.

[9] UML 1.3[S]. OMG Document: formal/00-03-01.

[10] UML 1.4[S]. OMG Document: formal/01-09-67.

[11] UML 1.5[S]. OMG Document: formal/03-03-01.

[12] UML 2.0 Superstructure Specification[S]. OMG Document: ptc/03-08-02.

[13] Chidamber S R, Kemerer C F. A metrics suite for object oriented design[J]. IEEE Transactions on Software Engineering, 1994, 20(6): 476-493.

[14] Bansiya J, Davis C G. A hierarchical model for object-oriented design quality assessment[J]. IEEE Transactions on Software Engineering, 2002, 28(1): 4-17.

[15] Reißing R. Towards a model for object-oriented design measurement[A]. Proceedings of the 5th International ECOOP Wokshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2001)[C]. Budapest, 2001. 71-84.

[16] Kobryn C. UML 2001: A standardization odyssey[J]. Communications of the ACM, 1999, 42(10): 29-37.

[17] Björkander M. Model-driven development and UML 2.0. The end of programming as we know it[J]? UPGRADE, European Journal for the Informatics Professional, 2003, 4(4): 10-14.

[18] Kobryn C. Will UML 2.0 be agile or awkward[J]? Communications of the ACM, 2002, 45(1): 107-110.

[19] Selic B, Ramackers G, Kobryn C. Evolution, Not revolution[J]. Communications of the ACM, 2002, 45(11): 70-72.

[20] UML 2.0 Superstructure RFP[Z]. OMG Document: ad/00-09-02.

[21] Purao S, Vaishnavi V. Product metrics for object-oriented systems[J]. ACM Computing Surveys (CSUR), 2003, 35(2): 191-221.

[22] Bansiya J. Evaluating framework architecture structural stability[J]. ACM Computing Surveys (CSUR), 2000, 32(1es): 18.

[23] Mattsson M, Bosch J. Characterizing stability in evolving frameworks[A]. Proceedings of the 29th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS EUROPE'99)[C]. Nancy, France, 1999. 118-130.

[24] Ma H H, Shao W Z, Zh L, Ma Z. Applying OO metrics to assess UML metamodels[A]. Baar T et al, eds. Proc. of the 7th International Conference of UML (UML'04)[C]. Springer-Verlag, LNCS 3273, 2004. 12-26.

[25] Kitchenham B, Pfleeger S L. Software quality: The elusive target[J]. IEEE Software, 1996, 13(1): 12-21.

[26] Brooks F. The Mythical Marr Month, Anniversary Edition[M]. MA: Addisorr Wesley, 1995.

作者简介:

马浩海 男, 1972 年 2 月生于内蒙古自治区呼和浩特市, 现为北京大学信息科学技术学院博士研究生, 研究方向为: 软件工程, 面向对象方法与技术, 模型驱动的语言、方法和技术. E-mail: mahh@sei.pku.edu.cn.

麻志毅 男, 1963 年 4 月生于内蒙古自治区赤峰市, 博士, 副教授, 主要研究方向: 软件工程, 面向对象方法与技术, 计算语言学.