ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.1867−1880
DOI: 10.3724/SP.J.1001.2008.01867

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

一种面向图形化建模语言表示法的元模型*

何 啸[1,2], 麻志毅[1,2+], 邵维忠[1,2]

[1](北京大学 信息科学技术学院,北京 100871)

[2](北京大学 高可信软件技术教育部重点实验,北京 100871)

A Metamodel for the Notation of Graphical Modeling Languages

HE Xiao[1,2], MA Zhi-Yi[1,2+], SHAO Wei-Zhong[1,2]

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

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

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

He X, Ma ZY, Shao WZ. A metamodel for the notation of graphical modeling languages. *Journal of Software*, 2008,19(8):1867−1880. http://www.jos.org.cn/1000-9825/19/1867.htm

Abstract: For graphical modeling languages, there are three problems on the notation definition: How to define graphical symbols for modeling elements; How to define the location relations between symbols; How to map the symbols and the location relations to the abstract syntax. For model transformation and code generation, the notation has to be represented as models. This paper proposes the notation definition metamodel (NDM) for metamodeling tools by summarizing and analyzing the notation of UML and UML family. For the three problems on notation definition, NDM is composed of three parts: basic figures and layouts, location relations and abstract syntax bridges. The notation model defined by NDM can be transformed to usable source codes. This paper also makes a comparison between NDM and other methods, and the results show that NDM have some advantages over other methods. NDM has been implemented in PKU MetaModeler, and some practices of NDM are introduced.

Key words: metamodel; metamodeling; notation; graphical modeling language; MDA (model driven architecture)

摘 要: 对于图形化的建模语言,为定义其表示法一般需要解决 3 个问题:如何定义每个建模元素的图形符号,如何定义图形符号之间的位置关系以及如何将表示法映射到抽象语法.为了方便进行模型转换和代码生成,还需要使用模型化的方式描述建模语言的表示法.通过对 UML 及其语言家族中的表示法进行总结、分析和归纳,提出了一种表示法定义元模型(notation definition metamodel,简称 NDM).针对定义表示法所面临的 3 个问题,NDM 被分成基本图元及其布局、基本位置关系和抽象语法桥三部分.使用 NDM 定义好的表示法模型还可以通过代码生成技术生成可使用的源代码.将 NDM 与其他几种定义表示法的方法进行了比较,结果表明,NDM 与其他方法相比具有优势.NDM 已经在元建模工具 PKU MetaModeler 中实现.介绍了 NDM 在实际应用中的几

* Supported by the National Natural Science Foundation of China No.60773152 (国家自然科学基金); the National Basic Research Program of China 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 National High-Tech Research and Development Plan of China under Grant Nos.2007AA01Z127, 2007AA010301 (国家高技术研究发展计划(863))

Received 2008-01-07; Accepted 2008-04-18

个案例.

关键词: 元模型;元建模;表示法;图形建模语言;MDA(model driven architecture)
中图法分类号: TP311 文献标识码: A

模型驱动体系结构(model driven architecture,简称 MDA)是对象管理组织(Object Management Group,简称 OMG)提出的一套运用模型进行软件开发的方法和技术[1].随着模型驱动技术及其相应开发环境的不断发展和普及,模型正逐渐取代计算机程序成为软件开发过程中的主要产品[2−4],这也导致了越来越多的建模需求.统一建模语言(unified modeling language,简称 UML)[5,6]的出现虽然在一定程度上满足了软件系统的建模需求,但却无法满足大量的针对特定领域的建模需求.这就需要定义一系列的领域相关的建模语言(domain specific language)或 UML 外扩(UML profile)[7,8],并建立相应的支撑环境.

元建模(MetaModeling)是一种定义建模语言的活动,它也是组成 MDA 的重要技术之一[9].利用元建模技术不仅可以简化建模语言的定义过程,还可以通过模型转换技术以及代码生成技术根据定义好的建模语言,直接生成相应的建模工具.利用元建模技术生产建模语言及其相应的建模工具,可以满足大量领域建模需求,如文献[10,11]中的需求.元建模技术还可以应对建模语言的不断演化.一旦建模语言的版本发生了变化,就可以利用元建模技术定义并产生变化后的建模语言,同时以增量的方式生成相应的建模工具.

元建模工具是一种支持元建模活动的 CASE 工具.利用元建模工具进行元建模活动的典型步骤是:

a) 利用工具中提供的语言,如 MOF[9],定义目标建模语言的抽象语法(abstract syntax);
b) 定义目标建模语言的具体语法(concrete syntax),即表示法(notation);
c) 描述目标语言的语义,可以使用文本的方式加以说明,也可以使用某些形式化的语言,如 OCL[12];
d) 定义相关的工具配置,设置一些选项;
e) 根据上述信息,利用代码生成技术或其他方式,生成目标语言的建模工具,称为目标建模工具.

其中,a)~c)步是用来定义目标语言本身,d)~e)步是用来定义和生成相应的建模工具.

在元建模的过程中,定义建模语言的抽象语法和具体语法是非常重要的步骤.建模语言的抽象语法规定了这个语言的抽象结构,在元建模过程中通常使用元类(meta-class)及其之间的关系来表示抽象语法.建模语言的具体语法定义了这种语言的具体表现形式,具体语法可以是文字式的,也可以是图形化的,本文讨论的表示法是指图形化的具体语法.

近年来,由于图形化的建模语言已经成为发展的主流,因此,如何在元建模过程中更好地定义表示法也成为一个重要的问题.定义建模语言的表示法并不是一个简单的绘图过程.一方面,建模语言的表示法是一种图形化的语法,包含一定的规则和语义;另一方面,表示法和抽象语法之间存在紧密的关联,需要建立表示法和抽象语法之间的映射[5].在定义抽象语法时,目前已有一些可以参考的标准和规范,如 MOF 和 ECore[13].但如何定义建模语言的表示法,目前却没有一个统一的标准.任何一种可以用来定义建模语言表示法的方法,至少需要能够解决以下 3 个主要问题:

1) 为每个建模元素分别定义自己的图形符号,例如用椭圆形来表示一个 use case 等.
2) 定义不同建模元素的图形符号之间各种可能的位置关系,例如嵌套、附着等.
3) 将上两步中定义的图形符号及它们之间的位置关系映射到建模语言的抽象语法上.这些映射说明了图形符号及它们之间的位置关系所代表的不同含义.

图形符号、位置关系和到抽象语法的映射是构成建模语言图形表示法的最主要的成分,如果一种方法或者实现无法解决上述 3 个问题,它就很难处理一些具有复杂表示法的建模语言,从而影响整个元建模活动的能力.此外,表示法的代码生成也是一个需要考虑的问题.通过代码生成技术,生成的建模工具可以脱离原来的元建模环境独立运行,并且允许人们通过修改源代码增加新的功能.为了方便代码生成,需要以一种结构化的方式,或者说是以模型化的方式来表示一种建模语言的表示法.

为了更好地解决上述几个问题,本文定义了一种专门为建模语言表示法建模的元模型——表示法定义元

模型(notation definition metamodel,简称 NDM).针对上述 3 个问题来定义 NDM 的结构和元素,使 NDM 可以较好地解决这 3 个问题.此外,利用 NDM 定义出的表示法模型,可以方便地进行模型转换和代码生成.目前,NDM 已经在元建模工具 PKU MetaModeler 中实现,并有一些成功的应用.有关 NDM 的初步设想已经发表在文献[14]中,本文将对这一工作进行更加详细的讨论.

本文第 1 节对定义表示法的问题进行简要分析.第 2 节介绍表示法定义元模型中的主要结构.第 3 节介绍表示法定义元模型的工具实现——PKU MetaModeler,并展示两个实际应用.第 4 节将介绍相关工作,并将 NDM 与其他几种实际应用的表示法定义的方案进行比较.最后是总结和展望.

1 问题的分析

支持图形化建模语言定义的元建模工具都需要定义建模语言表示法.经过分析和总结,目前的解决方法大致可以分成 4 种:1) 使用默认的表示法;2) 通过导入位图定义表示法;3) 通过提供简单的绘图工具定义表示法;4) 通过建立表示法模型来定义表示法.

使用默认的表示法是指,元建模工具中为目标建模语言定义了默认的表示法,例如对于块状建模元素(如类、接口、包,等等)都统一使用矩形来表示,而对于线状建模元素(如关联、继承,等等)则都使用统一的线条来表示.元建模人员无法随意修改建模语言的表示法.这种定义表示法的方式虽然简便、易行,但功能过于简单,对于稍微复杂一些的建模语言都无法有效地定义.因此,这种方法除了在早期的元建模工具和某些原型系统中存在以外,目前可用的元建模工具已经不再使用这种方式定义表示法了.

通过导入位图的方式来定义表示法可以看作是第 1 种方法的一个发展.在这种方法中,最大的变化就是允许元建模人员使用工具外部的位图资源来定义表示法,而不是仅能使用工具内部已经定义好的、不可修改的图形元素.通常,这种方法允许元建模人员为每个块状建模元素都指定一个位图文件,用位图文件中的图片作为块状建模元素的图形符号,而对于线状建模元素仍然使用工具默认的线条来表示.但是,通过导入位图的方法定义表示法的方式仍然存在一些不足.位图资源虽然可以由元建模人员创建和修改,但这种资源毕竟是一种静态资源,这使得建模工具无法通过这种静态资源反映更多的模型信息.

通过简单的绘图工具来定义表示法是指,元建模工具中集成一个简单的绘图工具,元建模人员可以使用这个工具来手工绘制建模元素的图形符号.这种方法可以解决由于过多地使用位图这种静态资源所导致的问题.因为所有的图形符号都是在元建模工具中绘制出来的,所以在绘制的过程中就可以将建模元素的某些信息绑定到图形符号上.此时,元建模工具记录的不是绘图的结果,而是绘图的过程(也称这种图为矢量图).在目标建模工具中,这些绘制过程会被重现,而那些绑定的信息会根据具体的实例动态地显示出来.与前两种方法相比,使用简单的绘图工具来定义表示法具有更大的灵活性,但是这个绘图的过程依赖个人的绘画水平,要想绘制漂亮的图形符号需要较高的技巧.

定义表示法的第 4 种方式是对建模语言的表示法进行建模,建立表示法模型.在这种方法中,表示法被模型化,元建模人员可以像定义抽象语法的模型那样定义表示法的模型,其中包括每个建模元素的图形符号、图形符号模型之间的位置关系以及它们到抽象语法的映射关系.最后,根据定义好的表示法模型,通过代码生成技术可以将这些模型直接转换成代码,从而应用到目标建模工具中.

可以看出,前 3 种方法虽然各有不同,但它们主要关注表示法的图形属性,解决了如何定义建模元素的图形符号的问题,却忽略了如何映射到抽象语法的问题.第 4 种方法与其他 3 种方法相比能够更好地解决定义表示法的问题,也是解决这一问题的发展方向之一.但是,目前应用这一方法的解决方案主要解决了建模元素图形符号的建模,并支持建立少量的从表示法到抽象语法的映射.而且这些方案大多从目标运行平台的图形系统出发来设计它们的建模元素,而并不是针对建模语言本身的特点进行设计,因此使用起来还不够方便和具有针对性.不仅如此,在目前所有的解决方案及其实现中,大部分都无法定义建模元素图形符号之间所有可能的位置关系,将图形符号和位置关系映射到抽象语法的能力也不强,大部分实现都采用默认设置的方式——即元建模人员无法明确定义这些映射.

本文提出一种专门用于定义建模语言表示法的元模型——表示法定义元模型(NDM).从分类上看,NDM 属于第 4 种定义表示法的方法.NDM 中的建模元素都是从 UML 语言家族中的表示法里归纳、总结而来,因此针对性较强.不仅如此,与已有的各种解决方案相比,NDM 可以更完整地解决定义表示法过程中的 3 个主要问题.

2 表示法定义元模型NDM

2.1 基本图元和布局策略

为了定义每个建模元素的图形符号,NDM 提供一组关于基本图元和布局策略的元素,用以定义图形符号的模型.

基本图元是一些常见的简单图形的抽象,例如矩形、圆形、三角形等等,是构造每个图形符号的最基本的元素.基本图元的选取需要一定的策略:如果选取的图元数量过多,则会造成整个元模型的规模过于庞大,并且增加了工具实现和用户使用的难度;如果选取的图元数量过少,则会导致用户无法定义足够的图形符号.因此本文选择基本图元的策略是:首先总结 UML 2.0 规范[6]中所有出现过的图形符号;然后统计其中各种简单形状出现的频数,从中选择最常见的形状作为基本图元,例如矩形、圆形、线,等等;最后再将其余的形状归类,抽象出一些具有一定扩展能力的形状作为基本图元,例如从五边形和六边形等形状中抽象出多边形.虽然矩形也是多边形的一种,但是由于矩形在图形符号中十分常见,因此多边形和矩形都被选择为基本图元.最终确定以下基本图元:矩形、圆角矩形、折角矩形、菱形、三角形、多边形、椭圆形、圆形、直线、弧线、文本对象和图片对象.

仅仅有基本图元还不能满足定义建模语言图形符号的需求,为了构造更加复杂的图形符号,还需要将图元组合起来,这就是布局策略的作用.布局策略反映了基本图元之间的一种组合关系,多个图元按照一定的方式组合起来形成复杂的图形符号.参与组合的图元可以分为两种类型:父图元和子图元.一个父图元可以与多个子图元组合在一起,父图元依据不同的布局策略确定子图元的位置和尺寸.

虽然图元可以按任意的方式组合起来,但是经过总结,对于 UML 表示法而言,图元的组合可以分为以下 3 种模式:

(1) 容器模式

这种模式实质上反映出了图元之间的一种嵌套组合的方式,一个块状图元作为容器(父图元),子图元在容器中按照一定的顺序排列,例如子图元可以按照顺序从上到下排列,也可以从左到右依次排列,还可以按照其他方式进行排列,如图 1 所示.

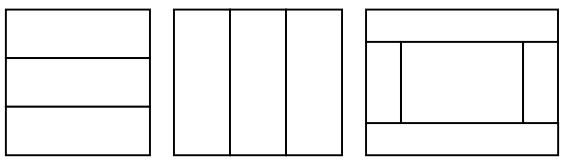


Fig.1 Container pattern

图 1 容器模式

符合“类模式”的表示法有很多,比如 Class,Interface,Package,Action,State,Use Case 等.

(2) 关联模式

这种模式实质上反映出线状图元与其他图元的组合方式,而线状图元作为父图元.这其中有两种情况:第 1 种情况是一个图元作为线端的修饰和线状图元组合在一起,如图 2(a)所示;另一种情况是其他图元排布在线状图元的周围,A,B,C 三个点是可能的排布位置,如图 2(b)所示.

符合“关联模式”的表示法也有很多,比如 Association,Generalization,Composition,Transition,Dependency 等等.

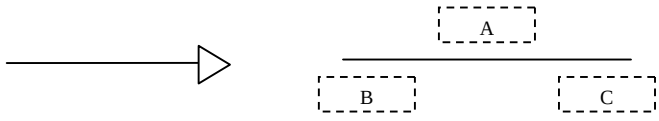


Fig.2 Association pattern

图 2 关联模式

(3) 矢量图模式

这一模式反映了图元之间更加复杂的组合方式,子图元按照各自的逻辑位置和尺寸分布在父图元内部,父图元按照自己的实际位置和尺寸以及子图元的逻辑位置和尺寸计算子图元的实际位置和尺寸.子图元的逻辑位置和逻辑尺寸中的 4 个分量都是在区间[0,1]之间取值的实数.如果假设父图元的实际坐标是(px,py),尺寸是(pw,ph),其某个子图元的逻辑坐标是(cx,cy),逻辑尺寸是(cw,ch),那么根据矢量图模式的布局策略,这个子图元的实际坐标(x,y)和尺寸(w,h)可以按照式(1)~式(4)所示的方式计算出来:

$$x = px + pw \times cx \tag{1}$$

$$y = py + ph \times cy \tag{2}$$

$$w = pw \times cw \tag{3}$$

$$h = ph \times ch \tag{4}$$

可见,这种组合方式相当于一种矢量图,因此这一布局策略是对第 1 节中介绍的第 3 种定义表示法方法的一种兼容.在 UML 里应用了矢量图模式的图形符号中,最典型的例子就是 Use Case 图里的 Actor.

2.2 基本位置关系

图形建模语言通过不同的表示法区分不同的建模元素.不仅如此,它还要通过这些表示法之间不同的位置组合来表现不同的含义.表示法之间的位置关系是表达模型含义的重要手段.比如,在 UML 的组合结构图中,当类 A 和类 B 没有关系时,二者的表示法可能是独立的;但当 B 是 A 的成分类(nested class)时,B 的表示法则要嵌套在 A 中.

基本位置关系与前面讨论的组合布局是不同的,组合布局反映了一个建模元素的图形符号内部不同图元之间的组合关系,而基本位置关系则定义了图形符号之间可能存在的位置关系.它有两个特点:第一,基本位置关系是两个图形符号模型之间的一种二元关系,而多元关系最终可以转化为二元的情况[15];第二,它只描述了一种可能存在,或者允许存在的关系,仅仅给出了图形符号之间位置关系的可能取值.如图 3 所示,是 UML 中出现的位置关系实例.

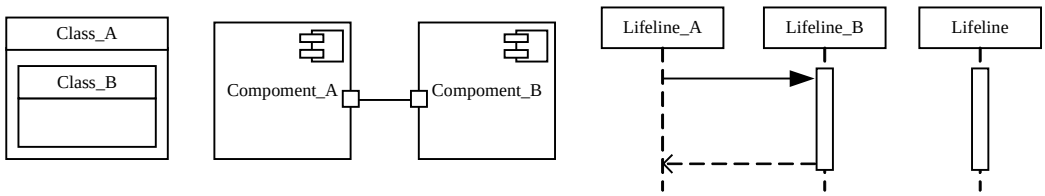


Fig.3 Examples of location relations in UML

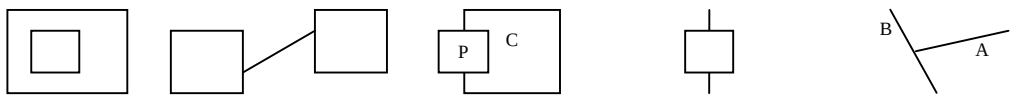
图 3 UML 中图形符号之间的各种位置关系

虽然图形之间的位置关系是多种多样的,但是经过分析发现,图形符号之间的位置关系最终可以归结为 5 种,如图 4 所示.

1) 嵌套型(nested type).如图 4(a)所示,块状元素嵌套在另一个块状元素内部,内层图元随着外层图元的移动而移动,但不会随着外层图元的变形而变形.例如 UML 类图中类的表示法之间的嵌套.
2) 连接型(connected type).如图 4(b)所示,线状元素的一个端点附着在一个块状元素的边界上,这一端点称

为附着端.线状元素的一个附着端将随着块状元素的移动和变形而移动,另一端则不会发生变化.例如,UML 类图中关系的表示法对类的附着.

3) 端口状附着型(port type).如图 4(c)所示,一个块状图元(记为 P)附着在另一个块状图元(记为 C)的边界上,图元 P 只能在图元 C 的边界上移动,P 随着 C 的移动和变形而移动,其大小通常不随 C 的变化而变化.例如 UML 中端口(port)的画法.
4) 结点附着型(node-attached type).如图 4(d)所示,块状图元附着在线状图元上,块状图元只能在线状图元上移动,通常这个线状图元是水平或者垂直的直线.线状图元的延展不会影响到块状图元的大小,但在必要的情况下,会导致它的移动.例如 UML 顺序图中生命线(lifeline)和执行规约(ExecutionSpecification)的画法.
5) 线端点附着型(end-attached type).如图 4(e)所示,线状元素 A 的一端附着在另一线状元素 B 上,A 的附着端只能在 B 上移动.例如 UML 消息和生命线的画法.



(a) Nested type (a) 嵌套型 (b) Connected type (b) 连接型 (c) Port type (c) 端口附着型 (d) Node-Attached type (d) 结点附着型 (e) End-Attached type (e) 线端点附着型

Fig.4 Location relations

图 4 基本位置关系

2.3 抽象语法桥

建模语言的表示法与普通图形的最大区别在于它本身具有特定的约束和语义,因此在定义表示法的时候必须能够指明它所对应的抽象语法和规则.本文使用抽象语法桥的概念来描述这种关系.所谓抽象语法桥实际上是表示法和抽象语法之间的一种映射关系.

抽象语法桥中包含 3 种不同的语法映射:元素映射、属性映射和关系映射.

元素映射用于将利用基本图元和布局建立起来的图形符号模型映射到具体的建模元素上,元素映射的源端是一个图形符号模型,目标端则是目标语言的某个建模元素.一个元素映射表示这个图形符号可以用来表示某个建模元素.例如,UML 2.0 中的 Actor 可以用一个“火柴棍”形的小人表示,此时就可以建立一个元素映射,以说明这种对应的关系.此外,一个建模元素可能同时具有多个元素映射,例如,Actor 也可以使用衍型的方式表示,或者是用一张图片表示,如图 5 所示.

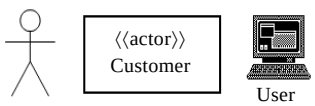
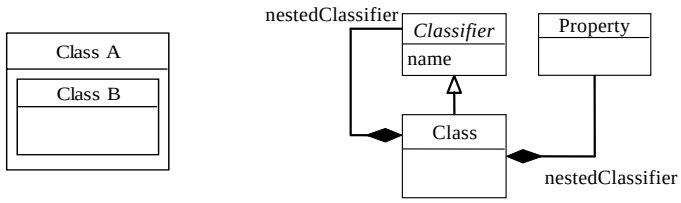


Fig.5 Three graphical representations of Actor

图 5 Actor 的 3 种图形符号

属性映射描述了在一个图形符号中如何表示建模元素的某个属性值,每个属性映射都隶属于一个元素映射,从而限定在属性映射中出现的建模元素和图形符号.例如,UML 中建模元素 Class 具有一个名为 name 的属性,这个属性用来表示类的名字.由于在 Class 的图形符号中,第 1 栏中的文字表示其名字.因此可以建立一个属性映射,将 Class 的图形符号中第 1 栏的文字映射到这个 name 属性上,这表示在第 1 栏中显示类的名字.同理,还可以利用属性映射说明,当类的 isAbstract 属性为 true 时,类的名字使用斜体表示.

关系映射描述了图形符号之间的基本位置关系和建模元素之间的关系的映射.在建模语言的表示法中,图形符号之间的各种位置关系通常表示建模元素之间的某个关系.例如,UML 中 Class 和 Classifier 之间存在一个聚合关系用来表示内嵌的类目,如图 6(b)所示,而这种聚合关系是用图形之间的嵌套型关系来表达的,如图 6(a)所示,因此,当一个类 A 的图形符号嵌套在另一个类 B 的图形符号中时,就表明 A 是 B 的内嵌类(nested class).对于这个例子,Class 的图形符号之间存在第 2.2 节中介绍的第 1 种位置关系,即块状图元之间的嵌套,此时可以建立一个关系映射,将这个位置关系映射到 Class 和 Classifier 之间的聚合关系.



(a) Nested representation (b) The abstract syntax of Class

(a) 嵌套的表示法 (b) Class 的抽象语法

Fig.6

图 6

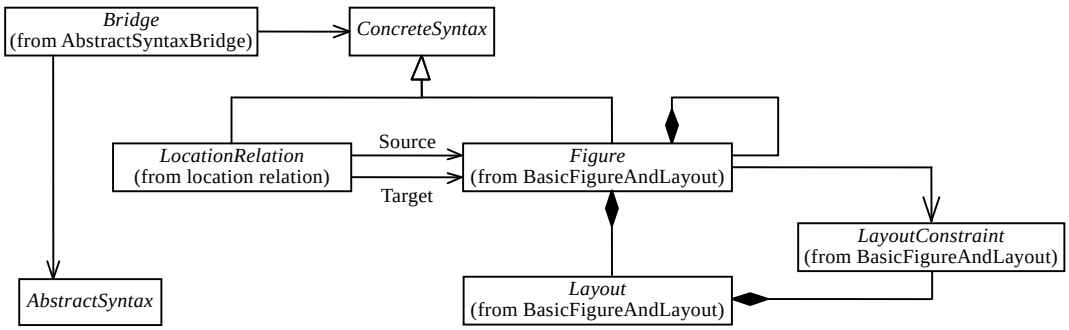
由于关系可以使用属性来表示[15],这就好像数据库中表之间的关系可以使用外码来表示,在某些情况下属性和关系可以相互转化,因此,属性映射和关系映射在某些场合可以互相替换使用.比如,在元模型 Class 中,Class 和 Operation 的关系原本是聚合关系,但在绝大多数的建模工具中,Operation 没有独立的图形符号,它的表示方式在 Class 的图形符号中已经说明.因此,在元建模过程中,可以将 Operation 转换为 Class 的一个属性(数组),然后用属性映射进行绑定.

在实现系统中,所有的抽象语法桥映射都保存在一个映射表中.每个表项都记录了一个指向抽象语法端的指针和一个指向表示法端的指针.在保存元素映射的表项中,抽象语法端的指针指向抽象语法模型中的一个类目(classifier),表示法端的指针指向表示法模型中的一个图形符号模型.在记录属性映射的表项中,抽象语法端的指针指向某个类目中的一个属性,表示法端的指针指向某个图形符号模型中的一个文字对象,此外,还需要一个指针指向该属性映射所隶属的元素映射的表项.在保存关系映射的表项中,抽象语法端的指针指向抽象语法模型中的一个关系(relationship),表示法端的指针指向本文定义的 5 种基本位置关系之一.

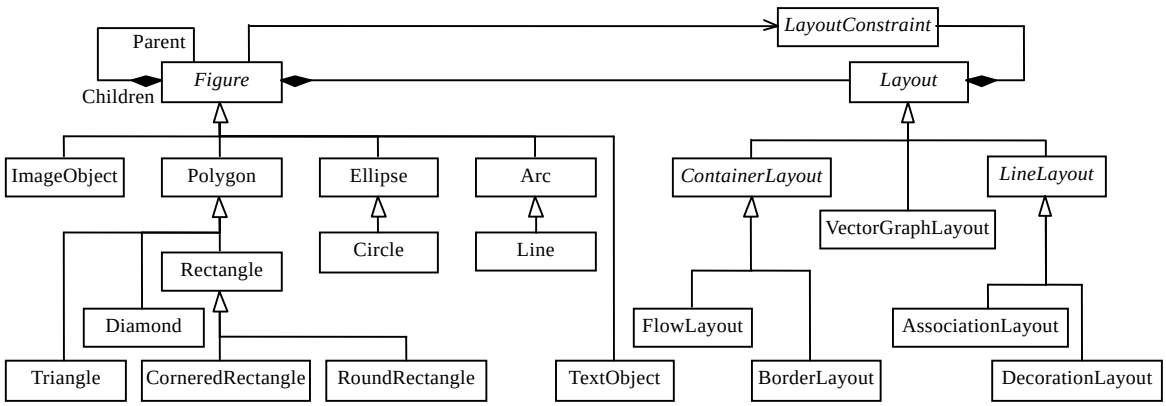
抽象语法桥所描述的表示法模型和抽象语法模型之间的映射关系并非是一种模型之间的静态关系,而是一种动态的实例对象之间的关系.这些关系可以通过代码生成转换成一些控制代码,这些控制代码将在目标建模工具运行时发挥作用.例如,一个属性映射将被转换成一段控制视图刷新的代码.每当视图需要刷新的时候,这段代码就会运行,根据不同的建模元素的实例更新它们所对应的图形符号实例.这样就保证了在目标建模工具中,每个建模元素的图形符号实例都可以动态地显示相应的模型信息.例如,在目标建模工具中创建两个 Class 元素的实例,它们的名字、属性和操作都不相同.根据相应的属性映射所转换的代码,这两个 Class 元素的实例所对应的图形符号实例将动态地显示出不同的信息.

2.4 NDM元模型

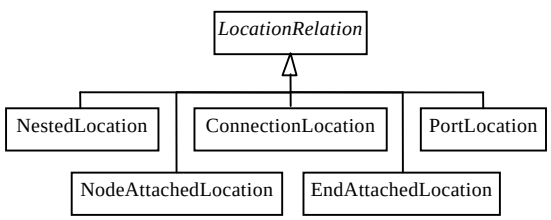
表示法定义元模型(NDM)的总体结构如图 7 所示,总共包含 3 个包:BasicFigureAndLayout,LocationRelation 和 AbstractSyntaxBridge.如图 7(b)所示,BasicFigureAndLayout 包中包含了基本图元和布局元模型元素.抽象类 Figure 是所有基本图元的抽象父类,Figure 的子类及其所有的子孙类就是 NDM 中包含的基本图元.抽象类 Layout 是所有布局的元模型元素,Layout 的子类 VerticalGraphLayout 代表向量图模式的布局策略.Layout 的抽象子类 ContainerLayout 是容器模式的布局策略,它有两个子类 FlowLayout 和 BorderLayout,Layout 的另一个抽象子类 LineLayout 代表关联模式的布局策略.抽象类 LayoutConstraint 代表布局的约束,每种布局都有自己的约束类型,限于篇幅,本文略去具体的约束类型.如图 7(c)所示,LocationRelation 包中包含了基本位置关系的元模型元素,抽象类 LocationRelation 是基本位置关系的抽象父类,其 5 个子类 NestedLocation,ConnectedLocation,PortLocation,NodeAttachedLocation 和 EndAttachedLocation 分别表示文中定义的 5 种基本位置关系.如图 7(d)所示,AbstractSyntaxBridge 包中包含抽象语法桥中的元模型元素,抽象类 Bridge 是所有抽象语法桥的抽象父类,它的 3 个子类 ElementMapping,AttributeMapping 和 RelationshipMapping 是具体的 3 种抽象语法桥.图 7(a)中是各个元素之间的关系.



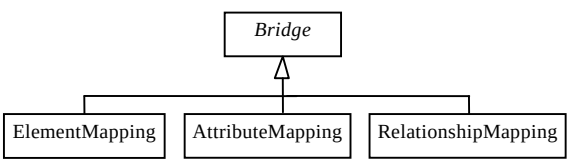
(a) Overall structure
(a) 总体结构



(b) BasicFigureAndLayout package
(b) 基本图元和布局包



(c) LocationRelation package
(c) 基本位置关系包



(d) AbstractSyntaxBridge package
(d) 抽象语法桥包

Fig.7 Notation definition metamodel
图 7 表示法定义元模型

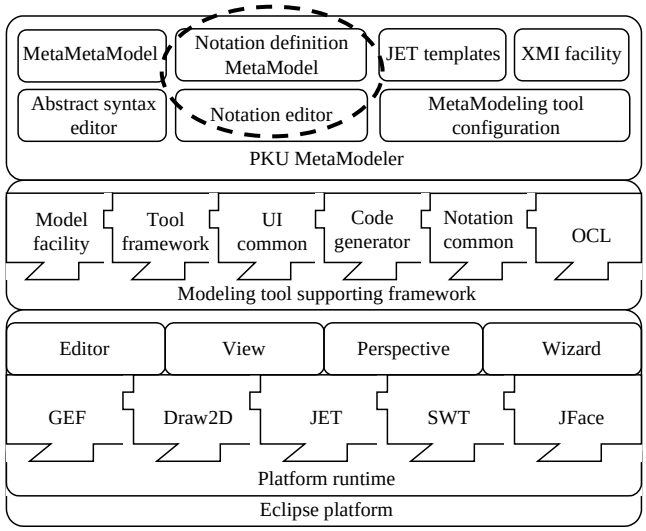
3 工具实现和应用

3.1 PKU MetaModeler

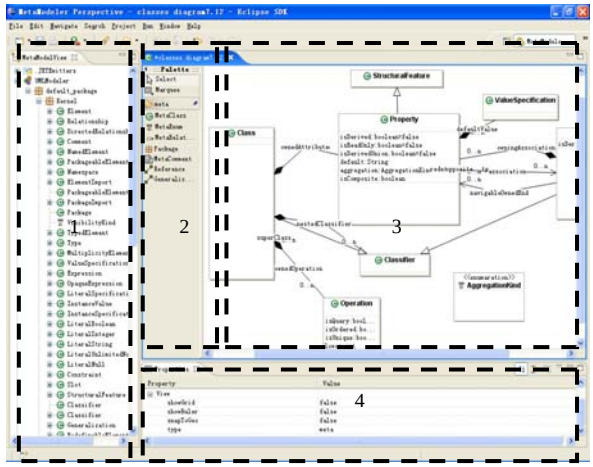
PKU MetaModeler 是一种基于 Eclipse 平台的元建模工具.PKU MetaModeler 使用 MOF[9]作为自己的元-元模型,并提供一套图形化的编辑器供用户建立抽象语法模型.同时,本文提出的表示法定义元模型 NDM 已经在 PKU MetaModeler 中实现.在定义好目标建模语言的基本信息之后,PKU MetaModeler 应用代码生成技术直接生成目标建模工具的源代码,目标建模工具同样基于 Eclipse 平台.

PKU MetaModeler 的体系结构如图 8(a)所示,整个结构分为 3 层.最底层是 Eclipse Platform,PKU MetaModeler 和由它生成的建模工具都是基于 Eclipse 平台,其中包括 Eclipse 上的一些插件,例如,GEF,Draw2D,

等等,以及一些 Eclipse 扩展点(extension point).中间层是建模工具支撑层,这一部分提供了一些建模工具共享的功能,例如有关模型生命周期管理、工具框架、OCL 引擎,等等,PKU MetaModeler 及所有由它生成的建模工具都会复用这一层次提供的功能.最顶层是工具层,对于 PKU MetaModeler 来说,这一层次包括了其元-元模型、抽象语法编辑器、代码生成的模版、XMI 工具,等等.本文定义的表示法定义元模型以及相应的表示法编辑器也都在这一层中实现(图中标注的地方).



(a) The architecture of PKU MetaModeler
(a) PKU MetaModeler 的体系结构



(b) Abstract syntax editor
(b) 抽象语法编辑器

Fig.8
图 8

PKU MetaModeler 的抽象语法编辑器如图 8(b)所示.在这个编辑器中,元建模人员可以使用一些元建模概念,诸如元类、元关系等建立抽象语法模型.图 8(b)中第 1 部分是模型树,这里显示了元建模项目中包含的抽象语法模型;第 2 部分是工具栏,这里提供了工具中支持的元建模元素;第 3 部分是主编辑窗口,这里,元建模人员可以使用可视化的方式建立抽象语法模型;第 4 部分是属性编辑页,对抽象语法模型元素的一些属性,可以在这里

方便地编辑.

当元建模人员建立好抽象语法模型时,就可以进行表示法编辑了.元建模人员只需在主编辑窗口中选择一个抽象语法模型元素,单击右键在弹出菜单中选择“定义表示法”,就可以调出表示法编辑器.在这个过程中,工具会自动建立一个元素映射,用来描述图形符号和建模元素之间的对应.PKU MetaModeler 的表示法编辑器如图 9 所示.主要结构与抽象语法编辑器类似,只是在工具栏中包含 NDM 中的元模型元素.表示法编辑器是一个图形化的、所见即所得(what you see is what you get,简称 WYSIWYG)的编辑器,元建模人员可以使用工具栏中提供的基本图元和布局直接绘制出建模元素的图形符号,在绘制过程中,工具会自动将其转换成图形符号模型.元建模人员选定一个图形符号后,还可以定义图形符号之间的 5 种基本位置关系.如图 10(a)所示,在弹出的对话框中选择不同种类的位置关系,并进行相应的设置,在对话框左部会出现简单的预览(图中显示的是一个端口型附着的预览),工具则会自动建立适当的关系映射.此外,还可以选择一个文本对象,然后定义属性映射.如图 10(b)所示,在弹出的对话框中选择抽象语法部分的一个属性(通过元素映射找到对应的抽象语法),就可以将这个文本对象映射到这个属性上.在表示法编辑器中,所有的建模过程都是以所见即所得的方式进行.

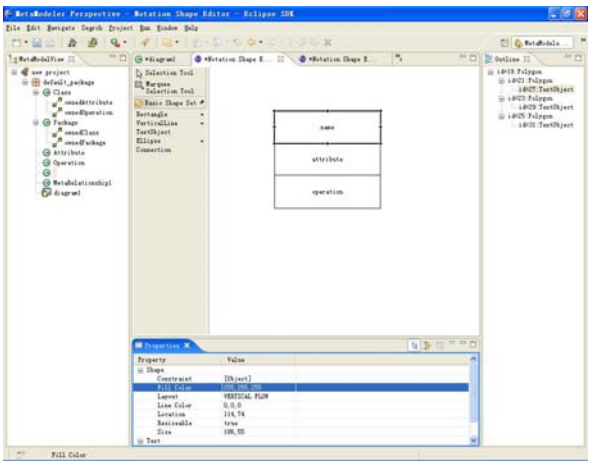
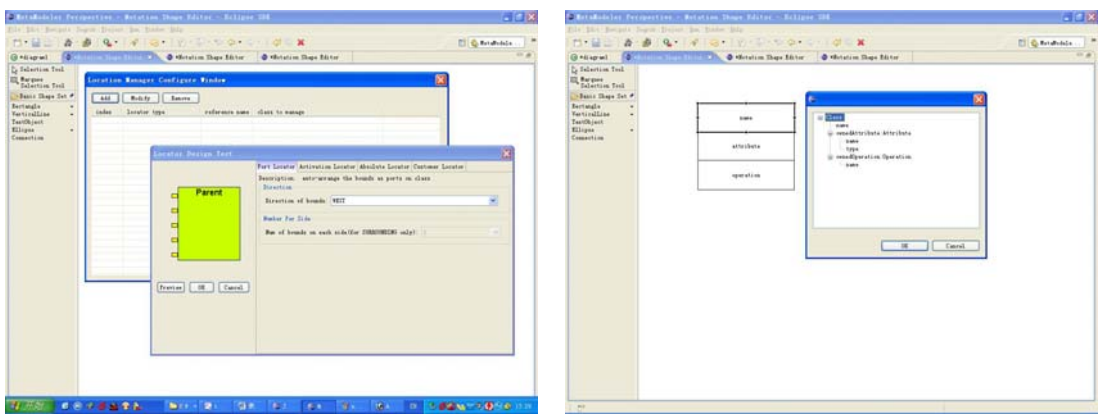


Fig.9 Notation editor

图 9 表示法编辑器

此外,PKU MetaModeler 还支持元建模人员描述建模语言的语义.在定义目标建模语言之后,元建模人员可以在模型树中选中根节点,右键单击后在弹出菜单中选择“代码生成”,然后根据向导进行相应的设置,随后进入代码生成的步骤.最后,PKU MetaModeler 会为元建模人员生成一个目标建模工具,该建模工具编译之后就可以直接在 Eclipse 平台上运行.



(a) Defining location relations between graphical symbols

(a) 定义图形符号之间的基本位置关系

(b) Setting up abstract syntax bridge

(b) 建立抽象语法桥

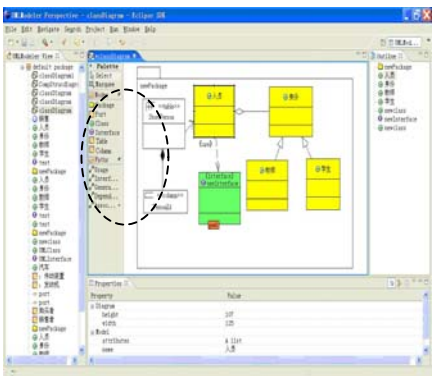
Fig.10

图 10

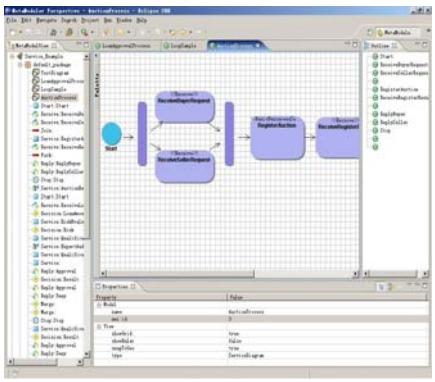
3.2 应用实例

第 1 个例子是一个扩展的 UML 类图建模工具,其中包括标准的 UML 类图元素以及一些为数据库系统建

模而扩充的元素,如图 11(a)所示.首先建立其抽象语法模型,然后分别定义各个建模元素图形符号,之后定义图形符号之间的基本位置关系,再次建立抽象语法桥.最后利用代码生成技术,生成目标建模工具.可以看出,图 11(a)中各个标准的 UML 建模元素的图形符号不仅符合规范中的定义,并且还支持规范中出现的各种基本位置关系.例如,Class 的图形符号可以嵌套在 Package 的图形符号中,Port 的图形符号可以附着在 Interface 的图形符号的边界上,等等,而这些位置关系在其他方法中则很难实现.所有的对于表示法的操作都可以直接反映在模型上,例如,将一个 Class 的图形符号拖拽进另一个 Class 的图形符号中时,这两个 Class 之间会自动建立一个组合关系,等等.此外,这个工具中还支持一些 UML 外扩(UML profile)中的元素,如图 11(a)中椭圆区域内的两个元素.



(a) A simple static structure modeling tool
(a) 简单的 UML 静态结构建模工具



(b) PKU ServiceModeler
(b) 北京大学服务建模工具

Fig.11
图 11

第 2 个例子是一个完整的服务建模工具——PKU ServiceModeler,如图 11(b)所示.由于 Web Service 是网络应用程序的主要实现技术之一,因此对 Web Service 进行建模成为一个重要的需求.我们针对 Web Service 的特点建立了一种描述抽象服务流程模型的建模语言(包括抽象语法和表示法).经过定义抽象语法、定义表示法、代码生成之后,就可以得到图中所示的 PKU ServiceModeler.这个工具中使用的表示法是参考 UML 活动图和 BPMN 而定义的,能够表示顺序、循环、分支、并发等各种控制结构以及用于描述服务流程的各种概念.从图 11(b)可以看出,使用 NDM 并配合代码生成技术,可以定义出足够精致的表示法.服务建模人员可以使用这个工具定义出流程图风格的服务流程模型,然后通过模型转换和代码生成得到可以直接运行的 BPEL 文档.

4 相关工作

GME(generic modeling environment)[16,17]是一种较为成熟的元建模工具,它可以通过导入位图的方式来定义表示法.除了可以定义建模元素的图形符号以外,GME 还可以定义某些图形符号之间的嵌套关系.但 GME 中定义表示法的能力比较有限,模型的图形符号除了能够反映它的类型(通过位图)和名字(通过在位图下方增加一行文字)以外,无法提供更多的信息.而在这方面,本文定义的 NDM 与之相比具有更强的能力.NDM 可以通过建立抽象语法桥将表示法模型映射到抽象语法模型上,从而在运行时可以根据这些映射关系动态地显示出更多的模型信息.

PKU MoDEL[18]是从北京大学面向对象建模工具 JBOO4[19]发展而来的一款模型驱动和语言家族的支持环境.PKU MoDEL 同样支持元建模活动,在进行元建模时,PKU MoDEL 就使用一个简单的绘图工具来定义建模语言的表示法,然后可以在 JBOO 中导入目标建模语言的信息,使用定义好的表示法进行建模.除了 PKU MoDEL 以外,元建模工具 MetaEdit+[20]也使用类似的方式来定义表示法,此外,MetaEdit+中还提供了一种称作 Connection Area 的概念来描述各种连接、附着的位置关系.但无论是 PKU MoDEL 还是 MetaEdit+,这些工具都

无法定义图形符号之间所有的基本位置关系,这使得在这些工具中无法定义出有更加复杂的表示法的建模语言,例如 UML.而本文的 NDM 定义了 5 种基本位置关系,这些基本位置关系能够覆盖绝大部分建模语言表示法中出现的情况.在实际应用中,我们已经利用 NDM 来定义诸如 UML 这样大型的建模语言的表示法.

GMF(eclipse graphical modeling framework)[21]是一种 Eclipse 平台下的开元建模工具,它可以看作是不支持图形化表示法的元建模工具 EMF(eclipse modeling framework)[13]的图形化版本.GMF 使用建模的方法定义表示法,它提供一套 GDM(graphical definition model)来定义建模元素的图形符号.GDM 是从 Eclipse 平台下的通用图形库 Draw2D[22]演化而来.GMF 还提供一套 MDM(mapping definition model)来建立表示法和抽象语法的映射.最后,GMF 可以通过代码生成技术生成目标建模工具的源代码.GMF 的 GDM 及 MDM 与本文的 NDM 十分类似,但也有不同.首先,NDM 中的基本图元和布局策略部分的元模型更具有针对性.GDM 虽然包括较为丰富的图形元素,但是表示法中的一些常见图形元素却不在其中,例如注释元素常用的折角矩形等,这导致在使用 GDM 定义图形符号的时候不够方便.而 NDM 的元素都是从 UML 语言家族中的表示法里总结出来的,因此直接包含了那些常用的图元和布局.其次,MDM 无法用来定义图形符号之间的位置关系,而在 NDM 中则可以利用基本位置关系部分的元模型来定义.此外,GMF 在定义表示法模型时没有使用图形化的方式,而是采用 EMF 的树形建模方式,这使得 GMF 在定义表示法时不够直观.

Dong 等人在文献[23]中提出了一种用于跟踪 UML 模型中的设计模式的方法.文献[23]中定义了一个 UML 外廓,其中包含一系列的衍型(stereotype)、标签值(tagged value)和约束.利用这个外廓,建模人员可以显式地定义出模型中的有关设计模式的信息,例如一个模型元素在某个设计模式中扮演的角色等.当模型发生变化时,建模人员仍然可以利用这一方法跟踪模型中的设计模式的信息,并且可以将这些信息动态地显示出来.文献[23]中的 UML 外廓和本文中 NDM 的抽象语法桥有相似之处.文献[23]中提出的方法实际上是在 UML 模型和各种设计模式之间建立一组映射,而本文的抽象语法桥则建立了具体语法到抽象语法之间的映射.利用文献[23]中的方法可以帮助建模人员跟踪并动态地显示模型中设计模式的信息,而利用本文提出的抽象语法桥则可以在建模元素的图形符号上动态地显示出模型元素的信息.文献[23]的工作与本文的不同之处在于,文献[23]的工作主要应用在建模活动中,应用的目标主要是模型;而本文的工作主要应用于元建模活动中,应用的目标主要是元模型.

本节下面的内容将实现了 NDM 的 PKU MetaModeler 和其他几种元建模工具进行对比,以比较这些工具在定义建模语言表示法时能力的强弱.比较分成 3 个层次:第 1 层次是比较单个元素建立图形符号的能力,这个比较又分为建立块状图形符号的能力和建立线状图形符号的能力;第 2 层次是比较定义图形符号之间的位置关系的能力,将分别对 5 个基本位置关系进行比较;第 3 层次是比较将表示法映射到抽象语法的能力,由于元素映射是工具肯定支持的映射,所以这里只比较描述属性映射和关系映射的能力.

参与比较的工具包括:GME,PKU MoDEL,MetaEdit+和 GMF.比较的结果见表 1.

Table 1 Comparison of notation definition functionality between PKU MetaModeler (NDM) and GME, PKU MoDEL, MetaEdit+, GMF

表 1 PKU MetaModeler(NDM),GME,PKU MoDEL,MetaEdit+和 GMF 之间关于定义表示法能力的比较

		GME	PKU MoDEL	MetaEdit+	GMF	PKU MetaModeler
Graphical symbol	Node	Bitmap	Simple graph editor	Simple graph editor	Using GDM	Using NDM
	Connection	Default line	Default line	Configurable line		
Location relations	Nested	Support	Do not support	Support	Support	Support
	Connected	Support	Support	Support	Support	Support
	Port	Do not support	Do not support	Do not support	Do not support	Support
	Node-Attached	Do not support	Do not support	Do not support	Do not support	Support
	End-Attached	Do not support	Do not support	Do not support	Do not support	Support
Syntax bridge	Attribute mapping	Name of the element	All attributes of the element	All attributes of the element	All attributes of the element	All attributes of the element
	Relationship mapping	Nested and connected	Connected	Nested, connected and end-attached	Nested and connected	All location relations

从比较中可以看出,应用了 NDM 的 PKU MetaModeler 在定义建模语言表示法的能力上与其他 4 种工具相

比具有一定的优势.首先,PKU MetaModeler在定义图形符号的能力上比GME和PKU MoDEL要强,因为这两个工具在定义线状图形符号时都只能采用工具默认的图形符号,而 PKU MetaModeler 可以通过所见即所得方式对所有的图形符号进行建模.其次,在定义基本位置关系的能力上,其他 4 种工具都无法支持全部的 5 种基本位置关系,但是 PKU MetaModeler 由于实现了 NDM,因此可以定义图形符号之间全部 5 种基本位置关系.最后,在定义抽象语法桥的能力上,其他几种工具也不能完整地描述表示法到抽象语法之间的映射,而 PKU MetaModeler 则能完整地描述所有的映射.

5 总结和展望

为了更好地解决元建模过程中的表示法建模问题,本文提出了一种用于定义表示法的元模型,以建立表示法模型.利用元模型中的基本图元和布局,可以方便地构造出一个建模元素的图形符号.基本位置关系则可以明确地描述图形符号之间所有可能的关系,从而可以更好地定义具有复杂表示法的建模语言.抽象语法桥则可以将定义好的图形符号以及它们之间的基本位置关系映射到抽象语法上.本文还简单介绍了 NDM 在元建模工具 PKU MetaModeler 中的具体实现以及一些应用实例.最后将 PKU MetaModeler 与其他 4 款元建模工具进行了比较,说明 NDM 在定义表示法上的优势.当然,NDM 还存在一些不足,比如,NDM 在定义时主要参考 UML 语言家族中的表示法,虽然这可以覆盖大部分的情况,但还存在一些特殊的表示法,无法利用 NDM 方便地进行描述.进一步增加 NDM 的表达能力,并使其更易于使用将是下一步的工作.

References:

[1] Object Management Group. MDA Guide version 1.0.1. OMG Document, omg/2003-06-01. 2003. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf

[2] Kent S. Model driven engineering. In: Butler M, Petre L, Sere K, eds. Proc. of the 3rd Int'l Conf. on Integrated Formal Methods (IFM 2002). LNCS 2335, Berlin: Springer-Verlag, 2002. 286−298.

[3] de Miguel M, Jourdan J, Salicki S. Practical experiences in the application of MDA. In: Jézéquel JM, Hussmann H, Cook S, eds. Proc. of the 5th Int'l Conf. on UML (UML 2002). LNCS 2460, Berlin: Springer-Verlag, 2002. 128−139.

[4] Booch G, Rumbaugh J, Jacobson I. Unified Modeling Language User Guide. New York: Addison Wesley Publishing Co., 1998.

[5] Object Management Group. Unified modeling language: Infrastructure version 2.0. OMG Document, formal/05-07-05, 2006. http://www/omg/org/cgi-bin/apps/doc?formal/05-07-05.pdf

[6] Object Management Group. Unified modeling language: Superstructure version 2.0. OMG Document, formal/05-07-04, 2005. http://www/omg/org/cgi-bin/apps/doc?formal/05-07-04.pdf

[7] Cook S. Domain-Specific modeling and model-driven architecture. In: Frankel D, Parodi J, eds. Proc. of the MDA Journal: Model Driven Architecture Straight from the Masters, Chap.3. Tampa: Meghan-Kiffer Press, 2004. http://www.davidfrankelconsulting.com/MDAJournal.htm

[8] Eriksson HE, Penker M, Lyons B, Fado D. UML2 Toolkit. Indianapolis: Wiley Publishing, 2004.

[9] Object Management Group. Meta object facility (MOF) 2.0 core specification version 2.0. OMG Document, formal/06-01-01, 2006. http://www.omg.org/cgi-bin/apps/doc?formal/06-01-01.pdf

[10] Yin JF, Lan QG. A variability metamodelling method in MDA context. Computer Engineering and Applications, 2006,42(19): 23−25 (in Chinese with English abstract).

[11] Yuan F, Li MS. MDA-Based model management method and its application for TRISO-model. Journal of Software, 2007,18(7): 1612−1625 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1612.htm

[12] Object Management Group. Object constraint language version 2.0. OMG Document, formal/06-05-01. 2006. http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01

[13] Eclipse.org. Eclipse modeling framework project. http://www.eclipse.org/modeling/emf/

[14] He X, Ma ZY, Li G, Shao WZ. A metamodel for the notation of graphical modeling languages. In: Ceballos S. Proc. of the 31st Int'l Computer Software and Applications Conf. Los Alamitos: IEEE Computer Society, 2007. 219−224.

[15] Shao WZ, Yang FQ. Object-Oriented System Analysis. 2nd ed., Beijing: Tsinghua University Press, 2006 (in Chinese).

[16] Ledeczi A, Maroti M, Bakay A, Karsai G, Garrett J, Thomason C, Nordstrom G, Sprinkle J, Volgyesi P. The generic modeling environment. In: Proc. of the WISP 2001. Budapest, 2001. 34−42.

[17] GME Web Site. http://www.isis.vanderbilt.edu/projects/gme/index.html

[18] Ma HH, Xie Bing, Ma ZY, Zhang NB, Shao WZ. PKU MoDEL: A model-driven development environment for languages family. Journal of Computer Research and Development, 2007,44(4):686−692 (in Chinese with English abstract).

[19] Ma ZY, Zhao JF, Meng XW, Zhang WJ. Research and implementation of Jade Bird object-oriented software modeling tool. Journal of Software, 2003,14(1):97−102 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/97.htm

[20] Pohjonen R. Metamodeling made easy—MetaEdit+ (tool demonstration). In: Glück R, Lowry M, eds. Proc. of the Generative Programming and Component Engineering, the 4th Int'l Conf. (GPCE 2005). LNCS 3676, Berlin: Springer-Verlag, 2005. 442−446.

[21] Eclipse.org. Graphical modeling framework (GMF). http://www.eclipse.org/gmf

[22] Eclipse.org. Graphical editing framework. http://www.eclipse.org/gef

[23] Dong J, Yang S, Zhang K. Visualizing design patterns in their applications and compositions. IEEE Trans. on Software Engineering, 2007,33(7):433−453.

附中文参考文献:

[10] 尹剑飞,兰庆国.MDA 环境下可变性元建模方法研究.计算机工程与应用,2006,42(19):23−25.

[11] 袁峰,李明树.基于 MDA 的 TRISO-Model 模型管理方法及应用.软件学报,2007,18(7):1612−1625. http://www.jos.org.cn/1000-9825/18/1612.htm

[15] 邵维忠,杨芙清.面向对象的系统分析.第 2 版,北京:清华大学出版社,2006.

[18] 马浩海,谢冰,麻志毅,张能斌,邵维忠.PKUMoDEL:模型驱动的开发和语言家族支持环境.计算机研究与发展,2007,44(4):686−692.

[19] 麻志毅,赵俊峰,孟祥文,张文娟.青鸟面向对象软件建模工具的研究与实现.软件学报,2003,14(1):97−102. http://www.jos.org.cn/1000-9825/14/97.htm

何啸(1983－),男,北京人,博士生,主要研究领域为面向对象建模,元建模,模型驱动的体系结构,模型转换技术.

麻志毅(1963－)男,博士,副教授,主要研究领域为软件工程与支撑环境,软件建模技术,面向对象技术.

邵维忠(1946－),男,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程环境,面向对象方法,软件复用,软件构件技术.