计算机科学 1997Vol.24№.4

面向对象的元建模技术

An Object-Oriented Meta-Modelling Technique

王 衡　　李景洲　董继润

（山东经济学院计算机系　济南 250014）（山东大学计算机系　济南 250100）

摘 要 For the efficient system specification development, it is very important for developers to compose the development methods suitable for their problem domain and environment before starting their system development. This paper discusses a meta-modelling technique based on object-oriented concept for the method base system. Information hiding and inheritance mechanism in object-oriented concept provide method fragments with high modularity and reusability. Method fragments are modelled and described in object-oriented formal description language Object Z. We also discuss how to customize method fragments and to integrate them to a new method.

关键词 Meta-modelling, Object-oriented, Method engineering, Customization, Method base, method fragments

许多开发方法及其支持工具（称为 CASE 工具）被用来支持系统规范及设计过程，从已应用的方法产生的效果来看，在很大程度上取决于系统开发所基于的问题域，也就是说，某些方法很适合某特定的问题域，而在另外一些问题域中则不然。例如，数据流图（DFD）并不适合实时系统的开发，需要采用把系统状态图合并到数据流图的方法。那么，是否存在适合任何问题域的规范开发方法呢？答案是否定的。我们认为最好的解决办法是由开发者针对他们的问题域及开发环境来构建合适的方法。元建模（Meta-Modelling）技术为我们构造自己的有效的方法提供了一个强有力的手段。

所谓元建模技术是指如何抽取方法构件并将其存到方法库中，如何从方法库中（一个包括现行可应用的方法及方法重要构件的数据库）获取、定制方法构件并将其集成到我们所关心的新方法中去的一种技术。我们之所以讨论面向对象的元建模技术，是为了构造能保存具有高度模块化及定制（Customization）能力的方法构件的方法库，因为面向对象的信息隐藏及继承机制能提供高度模块化和可重用性。方法构件由面向对象的正规描述语言 Object Z 来描述。我们将阐述定制方法构件及如何把方法构件集成到一个新方法中。最后，讨论构建一个基于这种技术的实用方法库的研究前景。

1 基于面向对象概念的元建模

1.1 元建模

元建模是构建系统的开发方法或方法构件。目前，有几种基于实体关系图（E-R 图）的元建模技术，如属性文法，谓词逻辑。实体关系模型（E-R 图），因其简洁性及易于在传统 CASE 工具里实现，得到了广泛的应用。

针对实用的方法库的元建模技术应该具有高度模块化及方法描述的可重用性。高度的模块化对方法构件提供了可修改性，修改其中一个方法构件就不会波及到其它的构件模块，利于产生可定制的方法构件。当我们从现有的方法构件组成新方法时，必须指出方法构件的约束条件。这样，就要求元建模技术对这种约束条件有充分的表达能力。实体关系模型不能满足这种要求，虽然它能说明关系的基数（一对一，一对多关系），但实体关系模型不能表示复杂的约束条件。因此我们需要像谓词逻辑这样一种工具来说明方法构件的各种复杂的约束条件。进一步说，目前使用的各种建模技术均未考虑到其所描述的方法构件的模块化及可重用性。面向对象的软件开发方法使得软件构件具有高度的可重用性及模块化，因此我们采用面向对象的方法进行元建模并表示元模型。如，用面向对象语言对方法进行描述。我们称所讨论的方法是面向对象的元建模。通过面向

对象的元建模技术，我们对各种面向对象的方法所建立的对象模型建立一种元模型，以最终构造实用的方法库，增加各种CASE工具的可重用性及集成性。进一步，我们使用了一种更具表达力的正规描述语言Object Z，Object Z是基于ZF集合论及谓词逻辑的Z的一种面向对象的扩展版本。谓词逻辑中的逻辑公式对表达方法构件中的各种约束是强有力的。

1.2 面向对象的元建模

建立方法的主要目标是在开发信息系统过程中对我们的开发活动进行导航(Navigation)。方法告诉开发者在开发过程中应产生哪些软件产品(Artifacts，如文档，代码)以及产生这些产品的工作顺序，就象工厂中生产一件制品的工序一样。因此，我们认为一个方法具有两种类型的信息：所产生的产品的结构及产生它们的步骤。在面向对象的元建模中，我们把一个方法或方法构件定义为类，它的对象实例代表方法的一个应用实例。根据方法构件产生的产品结构定义为与方法构件相关的类的属性，被封装在类定义的对象实例的操作表达了产生产品的方法的步骤。我们以Coad & Yourdon的OOA中的对象模型为例。该方法的一步骤"标识对象"(Identifying Objects)被定义为对对象实例的一个操作，该对象实例的类是对象图(Object Diagram)，这些操作改变对象实例的属性值。

假设使用如图1所示的对象图开发一个电梯控制系统，正在构建的对象图的信息被作为属性值存贮起来。当识别出一个对象"紧急按钮(Emergency-Button)，执行IdentifyObjects操作后，"紧急按钮"对象被增加到属性值中。图2是用Object Z描述的对象图的元模型的部分。通过使用Object Z，我们可以把正式的方法步骤标识为操作。定义包括前条件(Preconditions，操作执行前)，后条件(Postconditions，操作执行后)。在"识别结构"(Identifystructures)例中，输入"parent?"及"child?"在操作执行前就被包括在识别出的对象中。换句话说，方法的IdentifyObjects操作应该在执行Identifystructures前识别出对象"parent?"及"child?"。执行Identifystructures后，关系"parent? →child?"被增加到聚集关系(aggregation)中。前、后条件能标识方法步骤的执行次序。我们说明一下图2操作Z模式中的变量约定，变量说明部分"△"符号表示变量值可被操作修改。状态变量说明"?"代表操作后的状态，没有"!"的变量代表操作前的状态。"?""!"分别表示操作的输入和输出，"P"代表幂集。

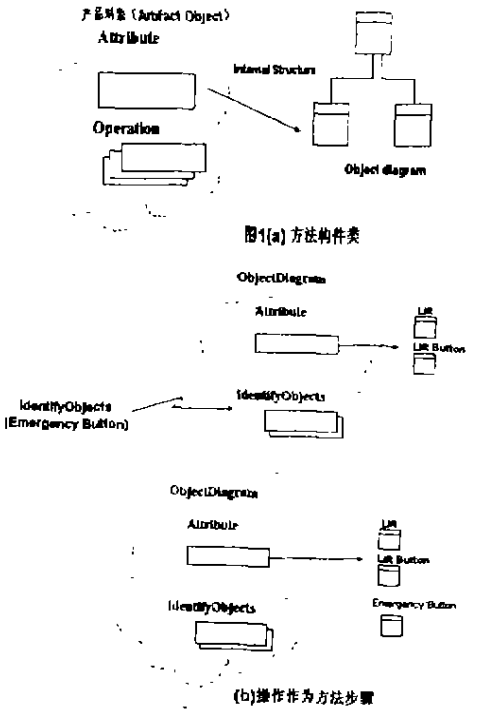
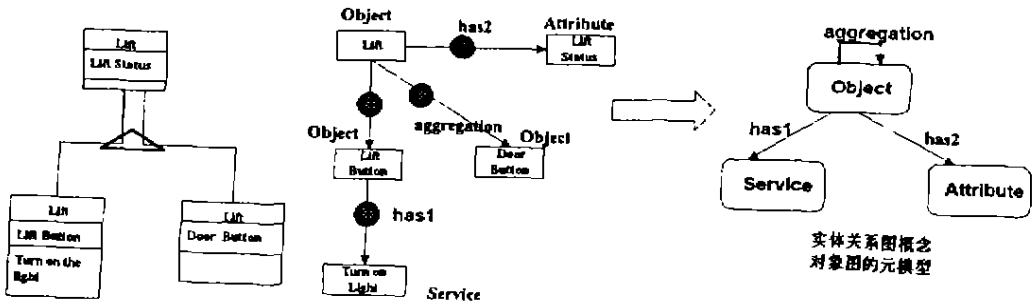


图1(a) 方法构件类

(b)操作作为方法步骤

图1 面向对象的元建模



(a) 对象图表示

(b)实体关系图表示

实体关系图概念对象图的元模型

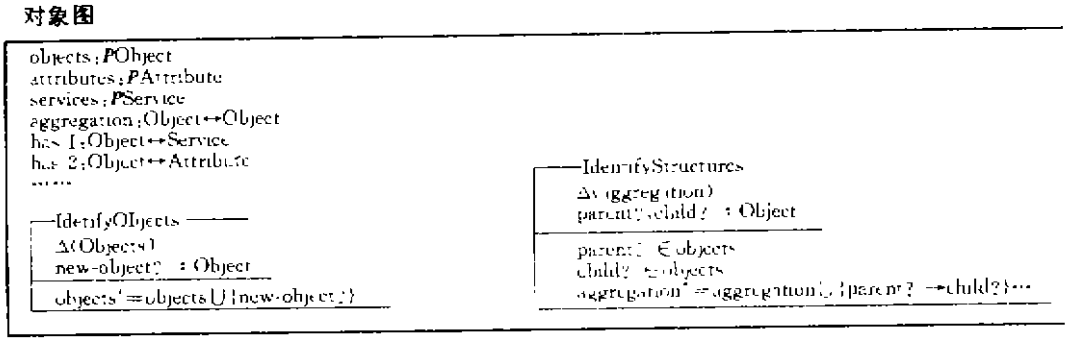


图 2 对象图的元模型及 Object Z 表示

2 方法的定制

面向对象的继承机制允许超类的属性和操作可被子类继承，这使得我们可以递增地定义新的方法构件，帮助我们从现有的方法中派生出新的方法构件以适应不同的需求。我们把新派生出的方法构件，作为该方法构件的子类，并重新描述新方法构件不同于原方法构件的部分。我们看一个简单的例子(图3)如何把实体关系图定制为一个对象图。对象图可看为一个特殊的实体关系图。对象图类定义的首要问题是说明对象图类是实体关系图类的子类。对象图类有两个特殊的关系"聚集"(aggregation)"、"分类"(classification)及概念"方法"(service)，因此在对象图类中必须加以定义。

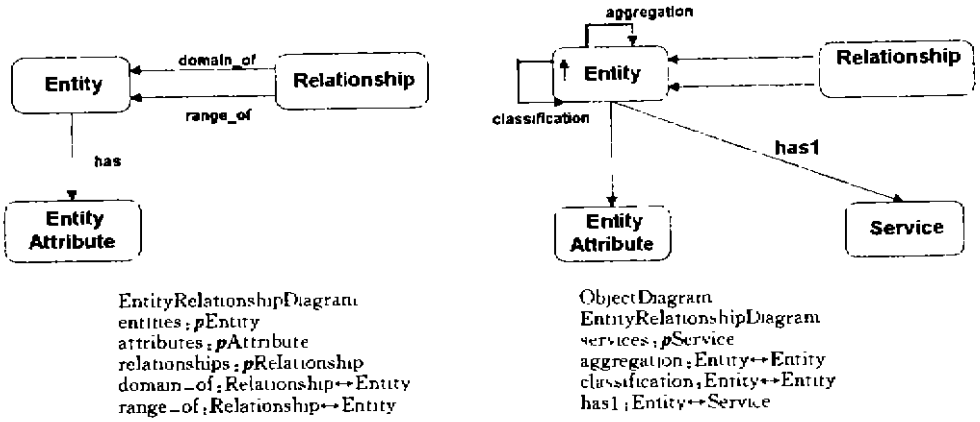


图 3 通过继承把 ER 图定制为对象图

3 方法集成

在方法的集成过程中，自然会产生新的约束。现在看一下集成图 2 所示方法构件对象图与状态转换图(STD，用来标明对象图描述中对象的行为)。我们必须在状态图属性"对象"及状态图类之间建立一种关系。另外，还必须定义相关关系的约束。新的约束可用逻辑公式来表示。为了继承被集成方法的特性，我们使用多继承。新方法也具有被集成的方法的属性。操作：图 4 勾画了方法集成的概貌。图中方法 1 (Method＃1)和方法 2(Method＃2)被集成为新的方法(Integrated Method)。方法 1 类(Method＃1 class)如图中 Artifact＃1，在某些约束下与方法 2 (Method ＃ 2)联接，以保证新方法(Integrated Method)的一致性。这些约束可被看作它们之间的一种关系。新方法(Integrated Methed)由方法 1 (Method＃1)及方法 2(Method＃2)继承而组成。方法 1 和方法 2 的属性和操作被新方法(Integrated Method)继承。方法 1 和方法 2 之间的关系被作为新属性嵌入新方法中。

通过对象图及状态转换图的集成例子，具体看一下，对象图中出现的任何对象都有内部状态，被对象的输入事件改变。这种状态的变化被标识在状态转换图中。本例中，对象图及状态转换图引入的约束为：①一个对象有一状态转换图。②状态转换图中的

任何输入事件，都要在对象的方法中定义。这些约束在 Z 模式 Connection-between-OBJD-and-STD 中定义为一种关系。图 5 是用逻辑公式标识。

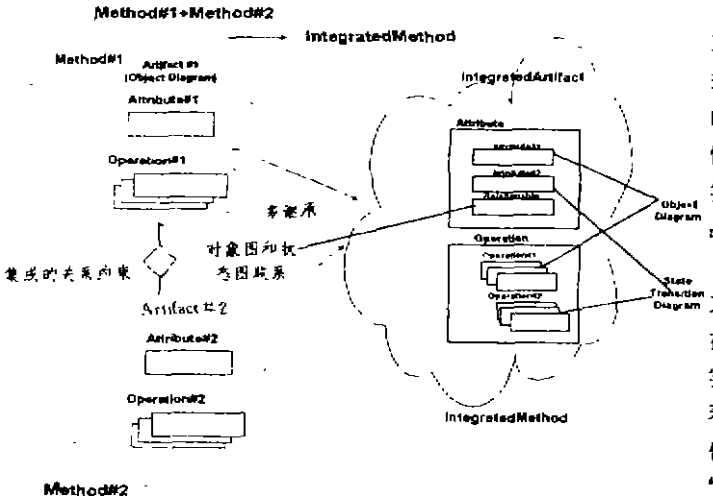


图 4 方法构件的集成

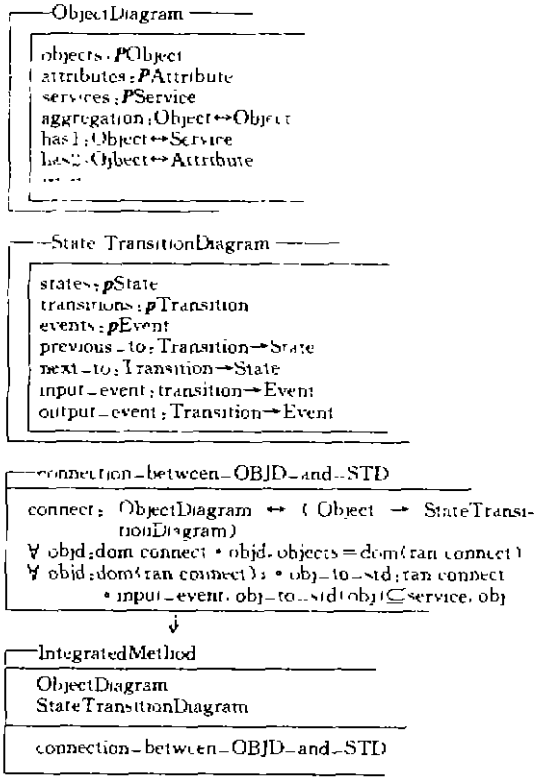


图 5 方法构件集成表示

4 方法库系统的未来方向

方法库系统应该提供存贮方法构件、获取合适的方法构件、集成出新的方法等功能，也能产生出一个支持新方法的 CASE 工具。这里我们讨论一下基于元建模技术的方法库的发展方向。

首先，为了支持方法库开发 CASE 工具，应考虑产品(Artifacts)的表示方式。现有的各种实用的方法有各种各样的表示方式，如图、表以及提高可理解性及用户界面友好的结构化文本。为了实现一个实用的方法库，需要把关于产品表示方式的信息嵌入到元模型中。

其次，如何产生一个支持新的集成方法的新 CASE 工具，其中一个方法是在方法库中既保存方法构件又保存其实现。什么是方法构件的实现？它可被看做是 CASE 工具的一部分，如工具构件。假定我们的方法库有一方法构件“数据流图”，它的工具构件是一操纵数据流图的编辑器。方法库保存一对方法构件及其工具构件。装配集成方法构件使得我们把其工具构件组装为一 CASE 工具。为了在基于装配成的 CASE 工具上的产品中保持一致性，当(图 6 所示)装配工具构件(Tool Fragments)时，我们应该增加一种类似于一致性检查的功能。系统工程师从方法库中选择方法构件并根据实际需要用方法编辑器定制方法，或根据需要集成新的方法。方法库中保存有方法构件的实现及其规范。图 6 中方法构造器检查其一致性。

第三，如何存贮方法构件以便可以重新获取它们，用户使用方法库的信息应该被模型化为方法构件的属性及其之间的关系，这与方法库的设计模式有关。图 7 是使用 PCTE OMS 作为平台的方法库模式的概貌。图中方法构件的属性如“difficulty”，“domain”，“keywords”，“manual”等对于重新获取方法构件是十分有用的信息。“difficulty”表示控制该方法的难易性，“domain”表示域名，如实时系统、商业应用、数据库系统，亦方法构件所应用的领域。“keywords”标识方法构件特征的关键字。“phase”表示在系统开发过程中适用的阶段。如需求分析，数据建模，结构设计。

方法构件的层次来源于超-子类层次结构。图 7 中方法构件的关系代表应用超-子类这种层次。层次结构也是需要进一步研究的工作。

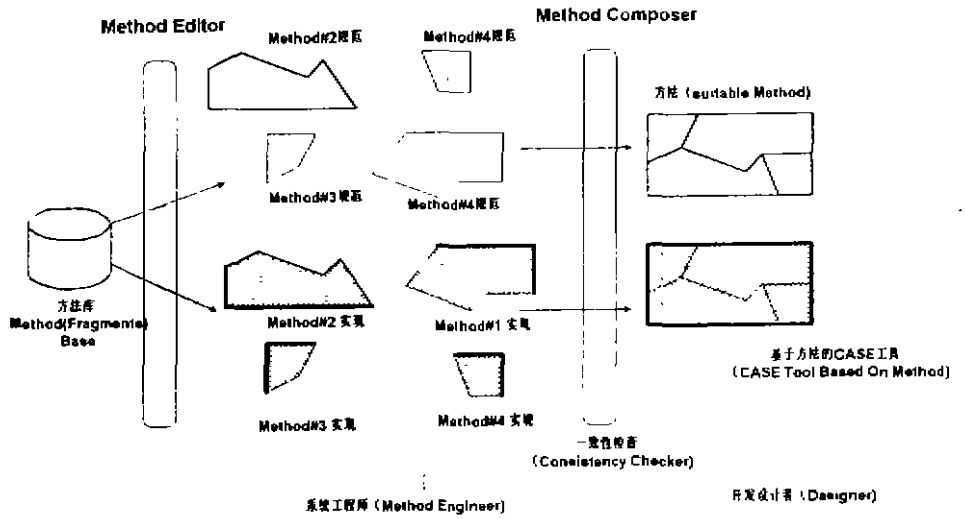


图 6 方法库系统及其支持工具

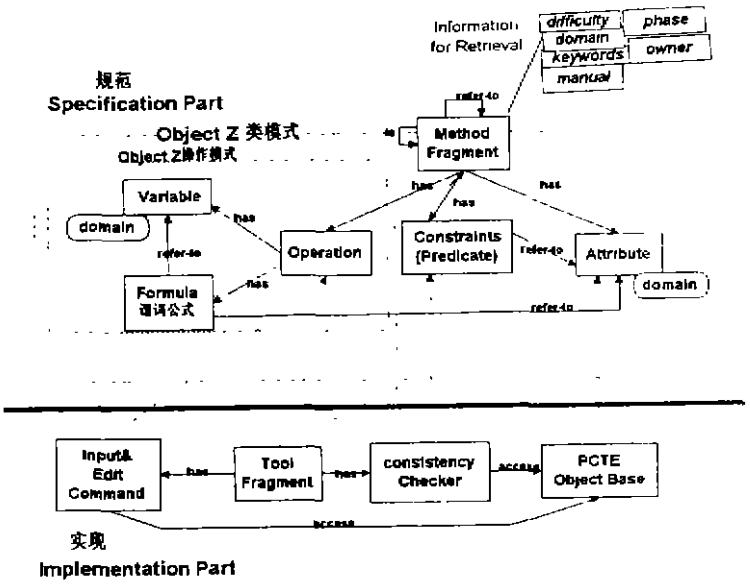


图 7 方法库的模式

参考文献

[1] P. Coad 等,Object-Oriented Analysis,Prentice Hall,1990

[2] G. Smith, The Object-Z Specification Language, Prentice Hall,1992

[3] S. Kelly 等,Support for Incremental Method Engineering and MetaCase,In Proc. of the 5th Workshop on the Next Generation of CASE Tools,1994

[4] K. Slooten, A Method Engineering Approach to Information System Development, In for. Sys. Dev. Proc., 1994

[5] L. Wakeman,PCTE:The Standard for Open Repositories, Prentice Hall 1993

[6] M. Saeki, Specifying Software Specification & Design Methods,Lecture Notes in Computer Science (CAiSE' 94) Springer-verlag,1994

[7] P. Ward, The Transformation Schema, IEEE Trans. on Soft. Eng., 2(12)1986

[8] T. G. Lewis,CASE:Computer-Aided Software Engineering, Van Nostrand Reinhold,1991

[9] Larry E. Towner,CASE—Concepts and Implementation,McGraw-Hill 1991

[10] Albert. Alderson, Meat-CASE Technology, Lecture Notes in Computer Science(Software Development Environments and CASE Technology)