软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(4):816−830 [doi: 10.3724/SP.J.1001.2012.04041]

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

一种支持多种模型转换技术的组合方法*

何 啸[1,2], 麻志毅[1,2+], 冯 超[1,2], 邵维忠[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)

Approach to Compositing Multiple Model Transformation Techniques

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

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

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

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

He X, Ma ZY, Feng C, Shao WZ. Approach to compositing multiple model transformation techniques. *Journal of Software*, 2012,23(4):816−830. http://www.jos.org.cn/1000-9825/4041.htm

Abstract: Model transformation is the key technique of model driven development. To handle difficult problems, it is necessary to combine smaller transformations into a complex one. Due to the heterogeneity among different transformation techniques, it is difficult to combine them together. The paper analyzes four essential conditions for composite transformations. Next, the paper proposes a composite transformation model, which consists of common type representation, common model representation, common transformation description, and composite transformation definition, in order to realize composite transformations. The paper also introduces the design and implementation of a transformation composition framework. A case study is also presented to illustrate the feasibility of this approach.

Key words: model driven development; model transformation; transformation composition

摘 要: 模型转换是模型驱动开发中的核心技术.为了解决复杂的转换问题,需要将多个相对简单的转换组合起来构成组合转换.目前存在多种转换技术,它们之间存在异构性,阻碍了组合转换的实现.首先分析实现组合转换的必要条件,进而提出一个组合转换模型,其中主要包括公共类型表示、公共模型表示、公共转换描述和组合转换定义语言等部分,用以实现支持多种转换技术的组合.另外,还介绍了一个组合转换平台的设计与实现,并通过一个案例说明所提方法及工具的可行性.

关键词: 模型驱动开发;模型转换;组合转换

中图法分类号: TP311 **文献标识码**: A

模型驱动开发(model driven development,简称 MDD)是一种以模型为核心制品的开发方法[1].模型转换(model transformation)是实现 MDD 的核心技术,它将一个模型转换成另一个模型,或者从一种格式转换成另一

* 基金项目: 国家自然科学基金(60821003); 国家重点基础研究发展计划(973)(2011CB302604); 国家高技术研究发展计划(863)(2007AA01Z127, 2007AA010301); 北京市自然科学基金(4122036), 教育部留学回国人员科研启动基金

收稿时间: 2010-11-12; 修改时间: 2011-01-24; 定稿时间: 2011-04-02

种格式.为了实施模型转换,首先需要定义转换规约,然后再部署在相应的执行引擎上加以运行.

实际的转换通常比较复杂:一方面,复杂的转换往往无法一步完成,因此一个转换过程可能包含多个子过程;另一方面,复杂的转换常常需要完成多个子任务,产生多个输出模型.为了实现这样的转换,一种可行的方案是将其分解成几个相对简单的子转换,每个子转换实现一个子过程或者子任务.之后再将它们组合起来,构成所需的复杂转换.组合转换(transformation composition 或 transformation chain)就是将一组相对简单的转换按照一定顺序组合在一起,从而构成一个复杂转换的方法.

目前存在多种模型转换技术,最常见的两类是基于图的转换技术[2](如 triple graph grammar,简称 TGG[3]和 attributed graph grammar,简称 AGG[4]等)和基于 MOF(meta object facility)[5]的转换技术(如 query/view/transformation,简称 QVT[6],ATLAS transformation language,简称 ATL[7],MOF model-to-text transformation language,简称 MOF M2T[8]等).根据以往的研究和实践,不同的转换技术常常被运用于不同的应用场景,解决不同类型的转换问题.因此,根据所要处理的子过程或子任务的特点,不同的子转换可能使用不同的技术来实现(或者复用已有的实现).

为了能够将一组子转换组合起来,就需要将不同的转换技术结合起来.但是,不同的技术具有一定的异构性[9],它们的理论和技术体系各不相同,这增加了组合转换的实现难度与开销,而目前的组合转换方法往往没有考虑这些异构性问题.同时,组合转换将子转换按照一定顺序组合在一起,即组合流程.为了定义组合转换,就需要描述这个组合流程,而目前也缺乏一种有效的组合转换定义语言,能够定义出复杂的组合流程.

为了实现组合转换,本文首先对其中存在的问题进行分析,找出实现组合转换的必要条件.之后,针对基于图的转换技术和基于 MOF 的转换技术,提出一种组合转换模型(composite transformation model,简称 CTM),并根据 CTM 实现了一个支持组合转换模型的转换平台(model transformation infrastructure,简称 MoTIF)用以支持组合转换.

1 组合转换问题分析

1.1 一个例子

为了说明组合转换中存在的问题,本文首先介绍一个 MDD 中存在的复杂转换.在数据库系统的开发过程中运用 MDD 方法,可以根据设计人员所建立的系统模型(用 UML 描述)转换成所对应的关系数据库模型(relational database model)及相应的文档.

具体来说,这一复杂的转换可以描述为以下几个步骤:1) 对 UML 模型进行预处理,删除那些对于转换没有影响的模型元素,例如接口、依赖关系等;2) 按照相应的转换方法,将 UML 类图转换成关系数据库模型;3) 根据用户选项,对关系数据库模型进行重构,使其符合某一范式(本例假设为第三范式);4) 将数据库模型转换为针对特定数据库系统的 SQL 语句,同时生成一个 HTML 格式的说明文档来描述这个数据库模型.

在上述步骤中共包含了 5 个相对简单的子转换,即:

T1. UMLPreprocessing:对 UML 模型的预处理;

T2. UML2RDBMS:类图到关系数据库模型的转换;

T3. RefactorTo3NF:对关系数据库模型的重构操作,使其符合第三范式;

T4. RDBMStoSQL:生成与关系数据库模型等价的 SQL 语句;

T5. RDBMStoHTML:根据关系数据库模型生成 HTML 的说明文档.

其中,T1 对输入模型进行预处理操作,T3 是重构转换.T1 和 T3 通常使用基于图的转换技术 AGG(attributed graph grammar)实现[10];T2 将类图转换成关系数据库模型,输入和输出的抽象层次相同,可以使用基于 MOF 的 QVT 实现[6];T4 是模型到文本的转换,可以使用基于 MOF 的 M2T 实现[8];T5 将模型转换成 HTML 文档,可以使用 ATL 实现[11].需要注意的是,T3 是一个可选的转换,用户可以选择是否执行 T3.

由于本文只讨论组合转换的相关问题,因此并不关注这些子转换的具体实现.在文献[6,8,10,11]中可以找到实现这些转换或类似转换的方法.

1.2 存在的问题

组合转换技术可以将 T1~T5 按照一定顺序组合起来,从而构成所需要的复杂转换,前一个转换的输出可能作为后面的输入.由于不同的转换可能使用不同的转换技术实现,因此组合转换技术也需要能够将不同的转换技术结合起来.

然而,不同转换技术之间存在异构性[9],主要体现在表示风格、执行方式、支撑技术等方面.表示风格的差异性是指不同技术中转换规约的表示方式不相同.例如,AGG 是图形化的方式,而 ATL 是文字式;再如,ATL 是命令式的,而 QVT 是声明式的.执行方式的差异性是指基于不同技术的转换在实际执行时的差异.例如,基于 ATL 的转换只能被单向执行,而基于 QVT 的转换可以双向执行.支撑技术的差异性是指不同转换技术所基于的支撑技术各不相同.例如,QVT,ATL,M2T 是基于 OCL[12]的,而 MOF,TGG 和 AGG 是基于图转换技术的.

由于这些异构性,导致组合转换技术中存在以下 3 个问题:

首先,不同的转换技术之间使用不同的格式表示模型,导致难以进行信息交互.在组合转换里,一个子转换的输出可能作为另一个的输入.例如在上面的例子中,UML 模型首先需要经过预处理,然后再将其转换成数据库模型,即 T1 的输出是 T2 的输入.但这两个转换使用不同模型存储格式,前者使用一种图语法来表示,后者则基于 MOF 的存储格式.因此,模型信息无法直接从 T1 传递给 T2.

其次,不同的转换技术使用不同的格式表示元模型,所以无法验证它们之间的兼容性.为了将两个转换组合起来,前者的输出能够作为后者的输入,这就要求该模型必须同时符合前一个转换的输出类型和后一个转换的输入类型,即后一个转换的输入类型必须能够兼容前一个转换的输出类型.因此,为了正确定义组合转换,需要判断不同转换的输入、输出类型是否能够相互兼容.转换输入和输出模型的类型,简称模型类型,通常使用元模型的方式定义.在文献[13]中已经给出了如何判断两个模型类型兼容性的算法,但要求它们必须具有相同的格式.然而,不同的转换技术缺乏统一的元模型描述方式,因而无法验证模型类型的兼容性,从而难以判断组合转换的正确性.

最后,转换规约缺乏统一的封装.组合转换将一组子转换组合起来,所以在定义组合转换时需要引用这些子转换的规约.但转换规约缺乏统一的格式,包含不同的信息.由于缺乏统一的封装,要同时使用这些格式各异的转换规约就会非常困难.

除了上述 3 个问题,还需要一种有丰富表达能力的组合转换定义语言.一方面,组合转换的结构不是简单的线性结构,即子转换并非按照单一的线性顺序依次执行.相反地,组合结构中可能包括选择结构、并发结构等等,如本文的例子便包含这两种结构.因此,在定义组合转换时,需要能够描述这些控制结构;另一方面,组合转换的执行势必要调用不同的转换引擎执行具体的子转换.由于每种引擎需要使用不同的参数.例如,执行 QVT 转换时需要指定转换方向,而执行 MTF(model transformation framework)[14]转换时需要指定起始规则,因此,在定义组合转换时,还需描述能够定义执行引擎的参数.

综上所述,本文给出实现组合转换的主要条件:

C1. 具有统一的元模型(模型类型)表示,从而能够判断模型类型的兼容性;

C2. 具有统一的模型表示,从而能够实现模型数据交换;

C3. 具有统一的转换规约描述格式,从而能够封装转换规约中的有用信息;

C4. 具有一个组合转换的定义语言,从而能够定义组合转换的结构与执行引擎的配置信息.

2 相关工作

目前,关于组合转换的工作主要包括 MCC[15],TCF[16],UniTI[17].此外,某些转换技术在一定程度上也支持组合转换.

MDA Control Center(MCC)是一个基于 Eclipse 的 MDD 工具,它支持组合转换.MCC 使用 Eclipse 的扩展点(extension point)机制,将不同的转换以 Eclipse 插件的形式组合在一起.通过一种脚本语言,可以支持转换之间的顺序、选择和并发结构.MCC 的不足之处在于,它使用 Java 接口和 Eclipse 接口来统一封装模型转换,因此,针对

每个子转换,都需要编写特定的 Java 代码将其封装并集成进入 MCC.这降低了该方法的灵活性,也增加了实现开销.此外,MCC 使用脚本语言来定义组合转换,这使得组合转换的定义不够直观.

Transformation Composition Framework(TCF)也是一种支持组合转换的工具.它基于构件组装的思想,将转换看成构件及接口,运用现有的构件组装技术将转换组合起来.在 TCF 中,转换和模型同样使用 Java 接口的方式进行封装.TCF 提供简单的转换定义语言,用来实现原子转换.但是 TCF 最大的不足在于,它无法支持不同转换技术的组合.TCF 只能组合用它所提供的转换语言定义的子转换.此外,TCF 使用已有的构件组装模型来定义组合转换,虽然能够清楚地定义子转换之间的组合关系(基于构件和连接子的方式),但无法直观地表达出组合流程中的各种控制结构,如选择和并发.

Unified Transformation Infrastructure(UniTI)是一个统一转换基础设施,它能够将基于不同技术的转换组合在一起.UniTI 采用 Unified Transformation Representation(UTR)模型来封装转换规约和定义组合转换.UniTI 虽然试图解决不同转换技术之间的组合,并提出 UTR 来封装不同格式的转换规约,但 UTR 并没有解决不同转换技术之间的模型和元模型格式不同的问题;其次,与 TCF 类似,UTR 使用了构件组装的思想定义组合转换,无法直观地表达出组合流程中的各种控制结构.

除此以外,有些转换技术,如 ATL 和 QVT,也支持组合转换,但它们不支持不同技术的转换组合.例如,针对 ATL 的 ATLFlow[18]和 Wires*[19],它们利用活动图的语法定义组合转换,但都只能将 ATL 转换组合起来形成复杂转换;而 QVT 也只能支持 QVT 转换之间的组合.

综上所述,现有的研究工作还不能很好地解决异构转换技术的组合问题.大部分的方法只能支持相同或相似的转换技术;或者需要进行大量的编码来封装每个子转换,从而弥补它们之间的异构性.为此,本文针对基于图和基于 MOF 的转换技术提出一种组合转换模型(CTM),它能够解决这两种技术之间主要的异构性问题,包括模型格式不同、元模型格式的不同和转换规约格式的不同.同时,CTM 还提供一种具有丰富表达能力的组合转换定义语言,能够直观和准确地描述组合流程.

3 组合转换模型 CTM

由于不同转换技术之间存在异构性,为了将它们组合起来,就需要根据不同技术的特点,抽象出一个公共模型,用来弥补各种差异.对于实现组合转换的 4 个主要条件(C1~C4),本文针对基于图的转换技术和基于 MOF 的转换技术,提出一个组合转换模型 CTM,以此来解决组合转换中出现的问题.

CTM 主要包括以下 5 个主要部分:基础结构、公共类型表示、公共模型表示、公共转换描述和组合转换定义语言,如图 1 所示.图中分别以包的形式表示这 5 个部分,下面分别加以阐述.

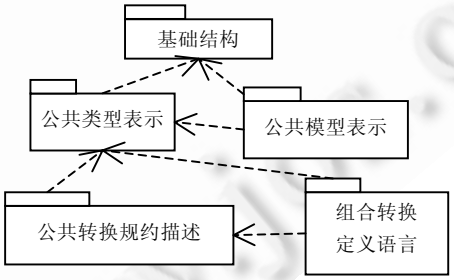


Fig.1 Packages of CTM

图 1 组合转换模型基本结构

3.1 基础结构

基于图和基于 MOF 的转换技术分别以图结构和 MOF 作为基础,因此,CTM 的基础结构包中首先定义相应的基础结构,如图 2 所示.

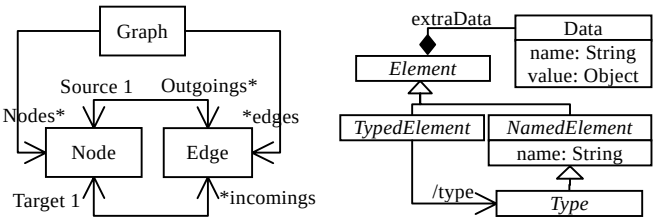


Fig.2 Basic structure

图 2 基础结构

在图转换中,图的概念被定义为四元组 $G=(N,E,s,t)$[2],其中,N 表示结点集合,E 表示边集合,映射 s,t:$E\rightarrow N$ 分别表示边到源点和目标点的映射.根据这个定义,利用模型的方式重新表示这个结构,就是图 2(a)所示的结构.其中定义了图、点和边这 3 个概念以及它们之间的关系.

图 2(b)是 MOF 模型基础结构,它根据 MOF 规范定义了一些与模型(元模型)有关的基础抽象类.其中,类 *NamedElement* 表示一个有名字的元素,类 *Type* 是所有类型的父类,而类 *TypedElement* 表示一个有类型的元素.最后,Data 是 CTM 中增加的元素,表示一个任意类型的数据项,可用来记录一些在 CTM 中没有预先定义的信息.

3.2 公共类型表示

为了统一信息表示格式,首先需要定义一种统一的元模型(即模型类型)表示方法,这样便可以检查这些元模型之间的兼容性.

基于图的转换技术使用“类型图(type graph)”[2]的概念表示元模型,而基于 MOF 的转换技术则使用 MOF[5]来定义元模型.例如,对于一个简单的状态机图元模型,用基于图和基于 MOF 的表示方法可以分别如图 3(a)和图 3(b)所示.通过分析比较可以得到二者的不同之处:首先,类型图中包含的所有概念(包括属性和数据类型)都是用结点表示,关系则用边表示,而基于 MOF 的元模型则使用(元)类和(元)关联等表示元模型,这与类型图的概念体系不同;其次,(元)属性不像类型图中的属性结点那样是一个独立的元素,而是(元)类的组成成分;最后,类型图中所有的边都是单向的,而(元)关联则可表示双向关系.

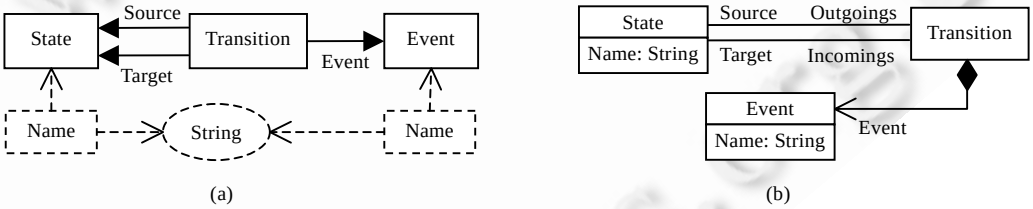


Fig.3 Two representation styles of metamodels

图 3 两种不同的元模型表示形式

为了统一元模型的表示形式,CTM 中定义了一个公共的类型模型,即元模型图(MetaModelGraph,如图 4 所示).它统一了 MOF 与图结构中的基本概念(图 2(a)和图 2(b)中的结构),使得类型图和基于 MOF 的元模型都能够方便地转换成这种表示形式.

公共类型表示的结构如图 4 所示.PrimitiveType 表示基本类型,例如整型、布尔型等,图 4 中略去了这些具体类型的定义.ComplexType 表示复杂类型,也就是元模型中定义的类型.MetaProperty 表示属性,MetaClass 表示元模型中的元素,一个 MetaClass 可以有多个父类(parents).MetaRelationship 表示元模型中的关系,它的 name 属性只表示目标端的角色名.为了兼容类型图中的边,MetaRelationship 只表示单向关系.而为了能够表示基于 MOF 的元模型,可以使用两个相对应的 MetaRelationship 表示一个双向关系.图 4 中的关联 opposite 表示与这个

单向关系对应的反向关系.

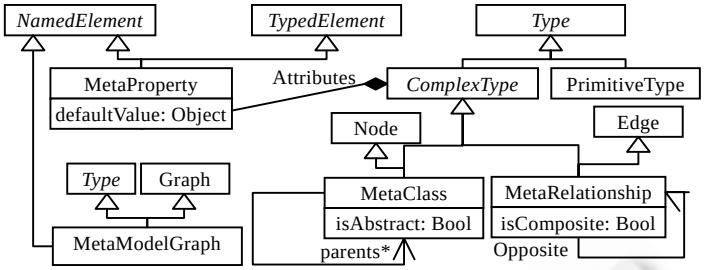


Fig.4 Common type representation

图 4 公共类型表示的结构

MOF 中还定义了 Package 的概念,但其作用主要是用于组织和管理模型元素,并不会影响模型转换的过程,因此,在公共类型表示中没有包含相应的概念.

3.3 公共模型表示

基于图和基于 MOF 的转换技术在表示模型的方式上也不相同,前者用图结构,后者使用基于 MOF 的模型结构.这使得它们之间很难进行信息的交换,为此,还需要定义一种统一的模型表示方法.

在基于图的转换技术中,模型也是使用图结构表示,称为“有类型的图(typed graph)”[2],其中包括一个图 G 和一个映射 type.G 表示模型的结构,结点和边分别表示模型中的元素与关系;type 将图 G 映射到一个类型图上,从而表示结点和边的类型.

而在基于 MOF 的转换技术中,模型的表示形式则直接依赖于元模型的定义,它们直接使用元模型中定义的概念来表示信息.如,状态图模型中使用“状态、变迁、事件”等概念,构建图则使用“构件、接口、连接子”等概念.也就是说,在基于 MOF 的转换技术中,不同模型的表示方法是不同的,并不像图转换技术那样统一使用结点和边的概念来表示模型.

为了解决这个问题,首先需要找到一种一致的方式来表示所有基于 MOF 的模型,其次才能设计出一种统一的格式来表示基于图的和基于 MOF 的转换技术中的模型信息.

对于所有基于 MOF 的模型,首先需要找到一种一致的表示方式.虽然它们的表示方法依赖于不同的元模型,但这些元模型都是使用 MOF 定义的,因此它们具有相同的基础,即 MOF.在这一前提下,可以使用 MOF 的实例模型(MOF instance model)[5]来表示所有基于 MOF 的模型.按照 OMG 定义的 4 层元模型体系结构,元模型是用 MOF 定义的,它们都是 MOF 的实例;而基于 MOF 的模型都是元模型的实例.在 MOF 规范中,MOF 的实例模型刻画的对象是元模型的实例.因此,我们可以用 MOF 的实例模型表示所有基于 MOF 的模型,这一逻辑关系可如图 5 所示.

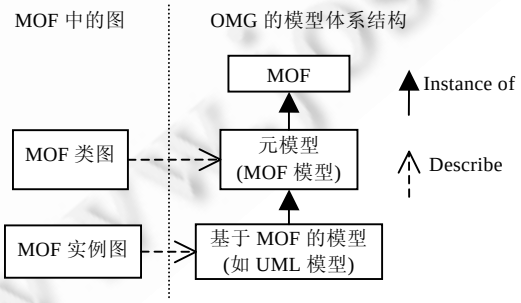


Fig.5 Relationship between diagrams and models

图 5 MOF 视图和基于 MOF 的模型的关系

然后,为了统一图结构和MOF实例模型,CTM中定义了一个公共模型表示,它将图结构和MOF实例模型结合起来,其结构如图6所示.类ModelGraph表示模型图,它与MetaModelGraph相关联,表示这个模型所遵循的元模型.Class 和 Relationship 分别表示模型的元素和关系,在图中分别用结点和边表示,它们都是实例对象(MOFInstance).它们分别关联MetaClass和MetaRelationship,表示其所实例化的类型.MOFInstance元素可以拥有属性,属性用类Property表示,它所关联的MetaProperty元素表示这个属性的声明.一个Property元素都是它所关联的MetaProperty元素的实例.在图转换技术里,有类型的图(typed graph)中的映射type在图6中分别使用关系definition,classType和relationType表示.

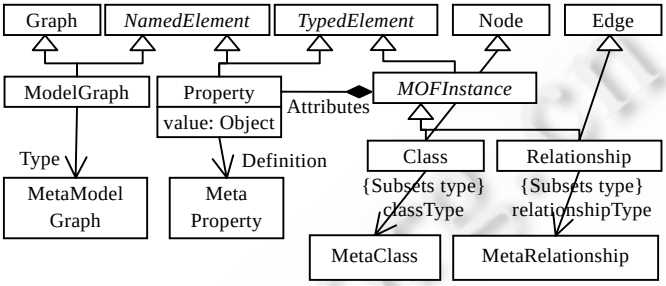
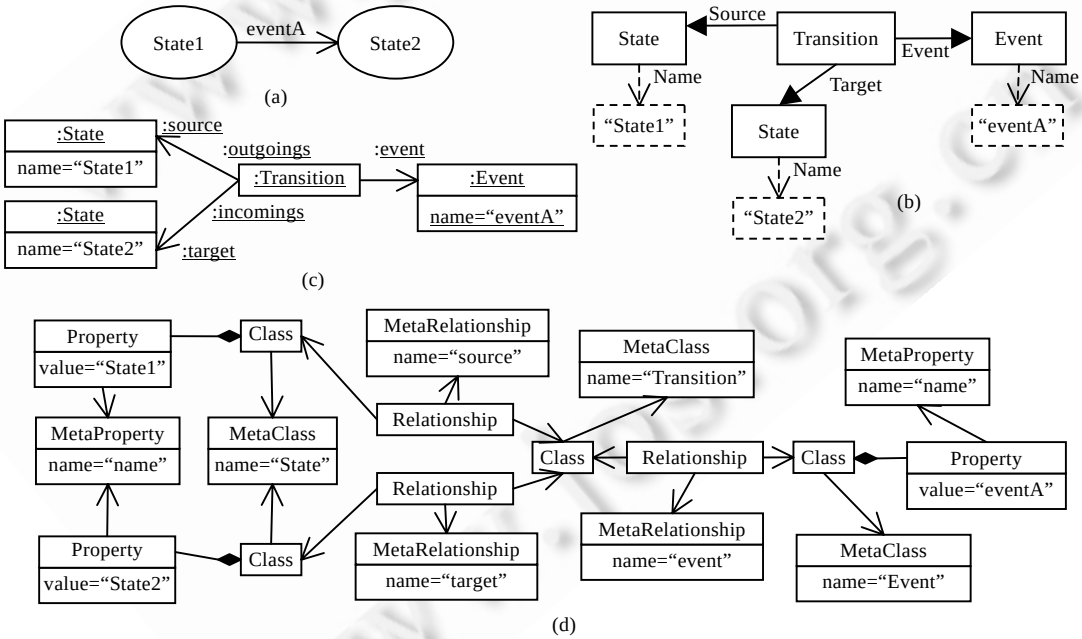


Fig.6 Common model representation

图6 公共模型表示的结构

图7所示为一个状态图模型的例子,分别表示使用基于MOF的模型、MOF实例模型、有类型的图和公共模型这4种方式表现的结果.图7(d)是公共模型表示的结构,其形式看似复杂,这主要是因为图中将模型与元模型的信息及其它们之间的关联也表现了出来.



(a) UML模型形式; (b) 有类型的图的形式; (c) MOF实例图的形式; (d) 公共模型表示的形式

Fig.7 Different representations of a state machine model

图7 一个简单状态机模型的不同表现形式

3.4 公共转换描述

不同的转换技术使用不同的格式定义转换规约,但却缺乏一种统一的描述和封装.为了进行组合转换,我们找到不同的转换规约中的共同点,并据此定义了一套公共转换描述格式,如图 8 所示.其中,主要包括具体转换规约中对外可见及在组合转换中有用的部分,例如转换的名字、转换的形式参数(类型和名字)、所包含的规则等信息,其作用类似于构件的接口描述.有了这些描述,就可以调用和执行转换了.

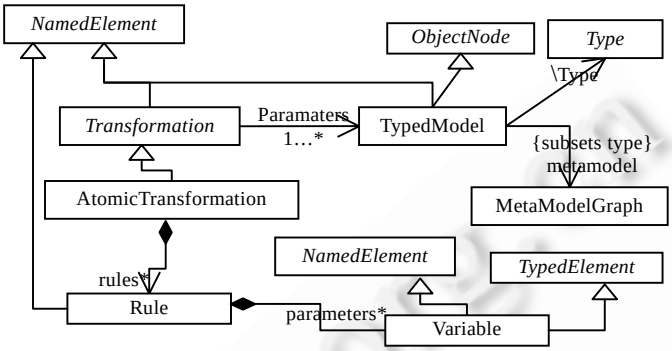


Fig.8 Common transformation description

图 8 公共转换描述格式的结构

抽象类 *Transfomration* 用来表示一个抽象的转换规约.一个 *Transformation* 可以有多个形式参数(parameter),它们都是 TypedModel 对象.TypedModel 的 name 属性表示参数的名字,而 TypedModel 还关联一个 *Type* 对象,用来表示参数的类型.一般来说,这个类型是一个 MetaModelGraph,即一个元模型;但在有些情况下,转换需要一些输入参数来控制转换的执行,此时,其参数的类型可能是一些基本类型.

AtomicTransformation 是 *Transformation* 的一个具体子类,它表示组合转换中的一个原子转换;而关于组合转换的部分,则在下一节中加以定义.当前,无论是基于图的转换技术还是基于 MOF 的转换技术,转换主要由一组规则构成.因此,每个 AtomicTransformation 由多个 Rule 组成,其中,类 Rule 表示转换规则.一个 Rule 元素包含多个规则参数,使用 Variable 表示.虽然本文主要关注以转换为单位的组合,但在统一转换规约中保留与规则有关的信息是因为,某些转换技术,如 MTF[14],在执行转换时需要指定一条初始规则,因此在公共转换描述中需要包含规则基本信息.

3.5 组合转换定义语言

最后,还需要给出一种组合转换的定义语言来描述组合转换的结构,并配置不同的执行引擎.为此,CTM 中定义了一种组合转换规约语言,如图 9 所示,便是这种语言的抽象语法.

在图中,CompositeTransformation 用来表示组合转换规约.为了描述清楚,我们又将组合转换的规约分为静态结构和动态行为两部分.静态结构描述一个转换所必需的信息,如转换的名称、转换参数及其类型.这部分信息与原子转换规约的内容相同(CompositeTransformation 和 AtomicTransformation 都继承自 *Transformation*).

此外,我们还需要描述一个组合转换中都包含了哪些相对简单的转换.因此,CompositeTransformation 和 *Transformation* 之间存在一个聚合关系.组合转换的静态规约也符合第 3.4 节定义的公共转换描述,所以它也可以被当作一个子转换来构造新的组合转换.

但组合转换的静态结构还不足以完整地描述组合转换,因为还需要定义这些转换是按照什么流程组合起来的,所以图 9 中定义了 TransformationActivity 这个元素来表示组合转换的动态行为,即组合流程.如第 2.2 节所述,组合流程中可能包括各种控制结构,如选择、并发等,因此定义的方法必须能够描述这些结构.图 9 中的组合定义语言通过扩展 UML[20]活动图元模型来定义组合流程,但图中并没有展现出完整的活动图元模型,主要显示了扩展部分.选择 UML 活动图作为描述组合转换的基础语言是因为它具有丰富的表达能力:它能够定义各种

控制结构,同时还能描述数据传递的情况.在定义复杂的组合流程时,这些表达能力都是必要的.

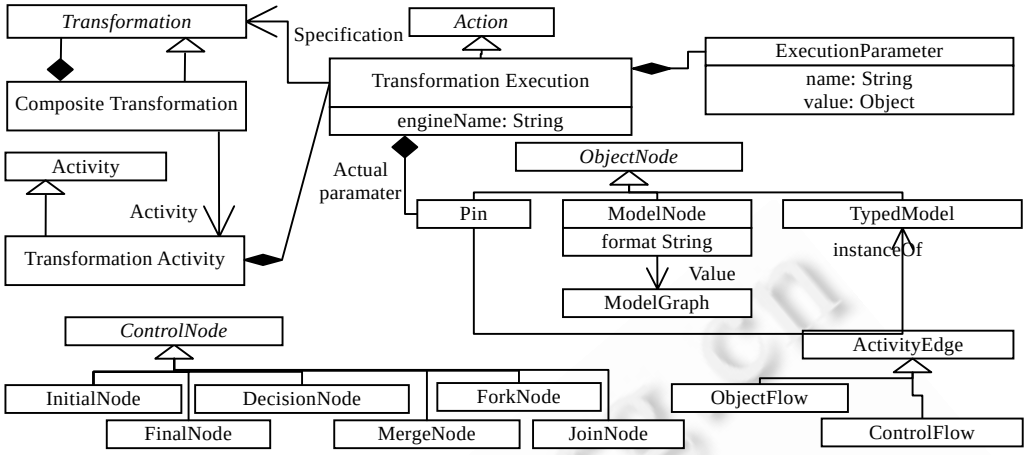


Fig.9 Composite transformation definition

图 9 组合转换规约的元模型

TransformationActivity 扩展自 Activity,表示组合转换的流程.在这个流程中,每个动作(步骤)都是执行一个转换,这里使用 TransformationExecution 来表示转换的执行,它扩展自 *Action*,而它所关联的 *Transformation* 表示所要执行的转换规约;其属性 engineName 表示所要使用的执行引擎的名字.ExecutionParamater 表示执行转换时的一些执行参数,例如执行方向、执行模式等.这个参数在实际执行时会发送给具体的转换执行引擎.

Pin 是 UML[20]活动图中的元素,用来表示 *Action* 的输入和输出数据.在组合转换规约中,Pin 用来表示 TransformationExecution 的实际参数,它们必须和所执行的转换中的形式参数相对应.例如,如果一个转换将 UML 模型转换成关系数据库模型(RDBMS),那么这个转换规约中就有两个形式参数;当执行这个转换时,就必须使用两个 Pin 来表示实际参数,分别表示 UML 模型和 RDBMS 模型.Pin 所对应的形式参数(即 TypedModel)用名为 instanceOf 的关联表示.

Pin 只表示某个转换的实际参数,但参数值的来源只能是其他的 *ObjectNode*.图 9 中定义了一个 ModelNode. ModelNode 表示一个来自文件的模型信息,它的值是一个 ModelGraph 对象,其属性 format 表示文件的存储格式. ModelNode 就可以用来给 Pin 提供实际数据,或者用来表示需要存储的模型文件.TypedModel 也是一种 ObjectNode,表示组合转换的形式参数.

此外,图 9 中还包括一些控制结点(ControlNode)以及控制流和数据流,它们的含义与 UML 活动图中的定义相同.

4 工具支持

为了支持组合转换,我们基于 CTM 设计并实现了一个支持组合转换的基础平台(model transformation infrastructure,简称 MoTIF),其主要结构如图 10 所示,图中还列出了主要的控制流和数据流.

首先,MoTIF 提供一个组合转换编辑器(如图 11 所示)来帮助用户定义组合转换的结构及流程,其中包括两个主要视图:组合转换的静态结构视图,用来描述组合转换及其子转换的转换规约;组合转换的动态流程视图,用来定义组合转换的组合流程.组合转换编辑器定义好的组合转换使用 CTM 中组合转换规约描述,包括静态结构及动态流程.在该编辑器中,用户可以查询转换规约库,选择已有的转换规约来构成组合转换.而转换规约库用于存储已有的转换规约,它们都用统一的转换规约进行了统一的描述.

为了保证组合转换的正确性,这个编辑器还会验证有关模型类型的兼容性.即,如果一个转换的结果要传递给另一个转换作为输入,那么这个编辑器会验证后者的输入类型和前者的输出类型是否兼容.编辑器会读取相

应的元模型信息,并将其转换成 CTM 的公共类型表示,然后利用文献[13]中的算法检查兼容性.

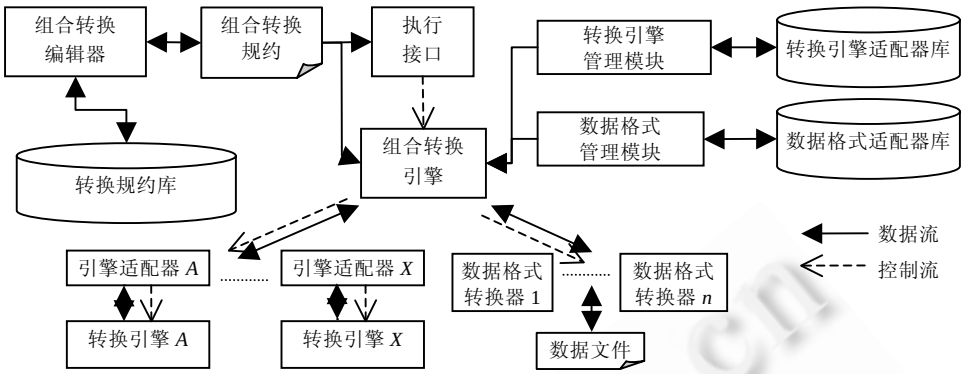


Fig.10 Architecture of MoTIf

图 10 MoTIf 的框架

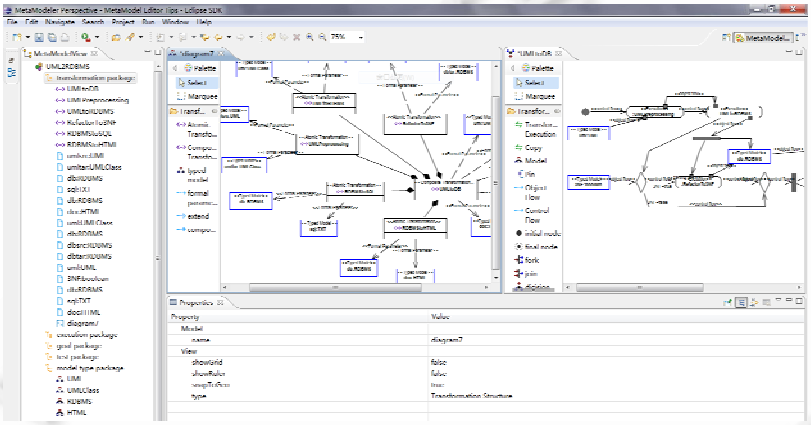


Fig.11 Screenshot of composite transformation editor in MoTIf

图 11 MoTIf 的组合转换编辑器

然后,使用 MoTIF 中的组合转换执行接口就可以读取组合转换规约.这个用户接口只是方便用户使用,而在执行时则会调用组合转换执行引擎.该引擎根据 CTM 中的组合转换流程语义,实现了一个流程执行模块.这个模块能够解释组合转换规约中的组合流程,按照其定义依次执行其中的简单转换.

当执行到某个具体转换时(执行到 TransformationExecution 元素时),就需要调用具体的转换引擎.此时,组合转换执行引擎就会从转换引擎管理模块中查找相应的转换引擎适配器.转换引擎管理模块主要负责管理和维护 MoTIF 中集成进来的具体的转换引擎的信息,通过为每个具体转换引擎实现一个引擎适配器,并将其注册到转换引擎管理模块,便可以将一个具体转换引擎集成到 MoTIF 中.转换引擎适配器封装了初始化、调用一个引擎以及为其设置执行参数、获得返回值等一系列操作,这样从实现层面上解决了转换引擎的封装问题.利用这个引擎适配器,组合转换执行引擎就可以自动调用相应的具体转换引擎,从而执行某个简单转换.

由于具体的转换引擎可能使用了不同的格式表示(元)模型信息,为了保证信息的交互,组合流程执行引擎还会根据需要从数据格式管理模块中查找相应的数据格式转换器.数据格式转换器实现了从磁盘中读取模型或元模型数据以及向磁盘写入模型或元模型数据的具体方法.每个转换器对应一种具体的存储格式,当从磁盘中读入信息时,数据格式转换器会将具体的数据转换成使用 CTM 表示的统一格式;而写入操作则会将用 CTM 表示的统一格式变成具体的存储格式存入磁盘.CTM 中定义的统一格式(公共类型表示和公共模型表示)在组合转换执行时作为数据的中间格式,这样可以方便不同格式的数据相互转换.如果希望为 MoTIF 增加新的数据

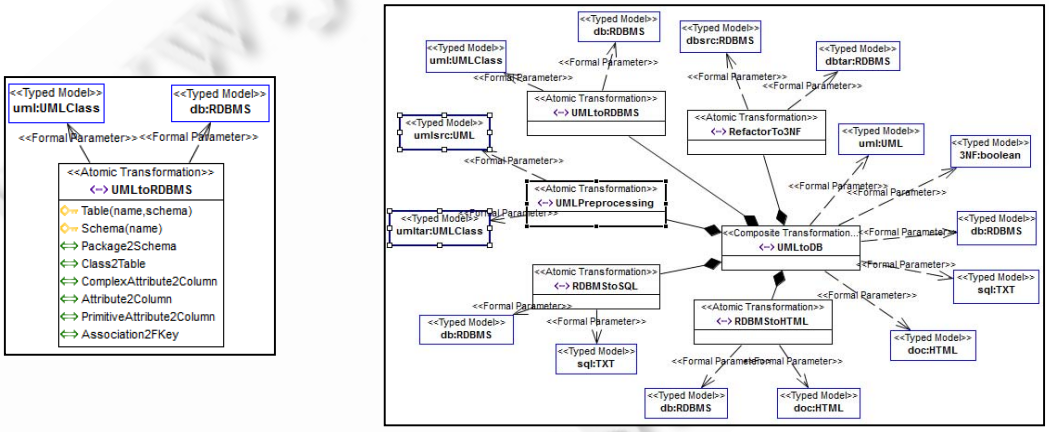
格式,用户只需要实现相应的数据格式转换器,并注册到数据格式管理模块中.

5 案例研究

下面将使用之前介绍的组合转换模型以及工具 MoTIF 实现第 2.1 节中介绍的组合转换.这个例子中存在第 1.2 节中讨论的几个问题:5 个子转换使用 4 种具体的转换技术实现(AGG,QVT,ATL,MOF M2T),它们有各自的规约描述格式;实现好的 5 个子转换分别部署在 4 个不同的执行引擎上运行,这些执行引擎的执行参数各不相同,而且它们分别依赖 3 种不同的数据存储格式;组合流程中又包括选择和并发结构,而并非简单的顺序结构.因此,本例具有一定的代表性.

假设这 5 个转换已经使用相应的技术实现(在文献[6,8,10,11]中可以找到这些转换或类似转换的实现方法),它们使用不同的格式定义转换规约.使用 MoTIF 中的转换规约库工具将它们导入进来之后,MoTIF 会利用公共转换规约模型对它们进行统一的封装.如图 12(a)所示,这是使用公共转换描述格式所表示的简单转换 UML2RDBMS.图形符号上的衍型标记(<< 和 >> 中的字符串)表示了符号的含义.图中用 <<Atomic Transformation>> 标记的矩形符号表示原子转换 UMLtoRDBMS,矩形中的字符串,如 Class2Table 和 Package2Schema,就是这个转换中所包含的规则.它有两个形式参数(图中所示的 Typed model 元素),名为 uml 和 db,其类型分别是 UMLClass 和 RDBMS.

之后,利用组合转换编辑器定义组合转换结构.图 12(b)是组合转换的静态结构,其中出现了 5 个子转换,即 UMLPreprocessing,UMLtoRDBMS,RefactorTo3NF,RDBMStoSQL 和 RDBMStoHTML.每个转换都具有类似图 12(a)那样的统一转换规约描述,受篇幅所限,这里省略了详细信息.组合转换的名字叫做 UMLtoDB,共有 5 个形式参数,分别是 uml,db,3NF,sql,doc.其中,uml 和 db 表示 UML 模型和 RDBMS 模型;sql 表示要输出的 SQL 语句;而 3NF 是布尔型参数,用来控制是否要将数据库模型转化成第三范式;doc 是要输出的 HTML 文档.



(a) 一个原子转换描述 (b) 组合转换的静态结构

Fig.12 Specifications of transformations

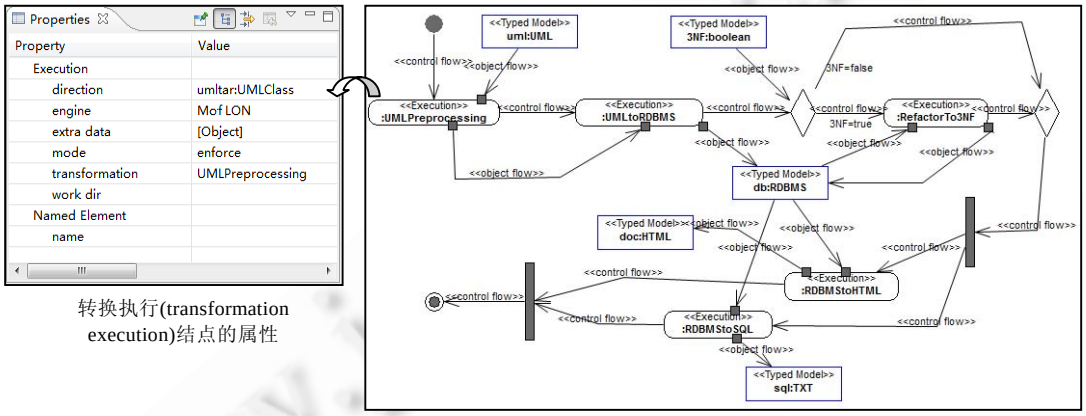
图 12 转换的规约

接下来定义组合转换流程,根据上文描述的逻辑,组合转换的流程定义如图 13 所示.图 13 参照 UML 活动图的语法,圆角矩形表示 TransformationExecution,其他符号和活动图含义相同.之后,再定义每个 TransformationExecution 所需要的执行参数(ExecutionParamater).例如,对于 UMLPreprocessing 这个 QVT 转换,需要指定其执行的方向为从 umlsrc 到 umltar,以及执行的模式为‘enforce’.

之后,利用 MoTIF 上的转换执行接口就可以执行这个组合转换.在执行过程中,首先,MoTIF 读取所有的实际参数.然后,组合转换执行引擎会根据图 13 所示的流程,调用相应的转换执行引擎,依次执行每个转换.其中,如

果参数 3NF 是 false,那么就不会执行 RefactorTo3NF.在这个例子中,不同的转换使用不同的转换引擎:UMLPreprocessing 和 RefactorTo3NF 使用 AGG 工具执行;UML2RDBMS 使用 MOFLON 工具执行[21];RDBMStoSQL 使用 Acceleo[22]执行;而 RDBMStoHTML 使用 ATL 执行.这些引擎在执行时不仅使用不同的执行参数,而且还使用了不同的文件格式.AGG 使用自己定义的图格式来存储模型类型与模型;MOFLON 使用标准的 XMI 格式存储数据;而 ATL 和 Acceleo 都使用 Ecore 格式[23](MOF 的一种变体).这些工具都无法直接读取其他格式的数据.

为了执行这个组合转换,MoTIF 首先按照组合流程模型中对每个转换所定义的执行参数,配置所需的执行引擎.执行过程中,由于每个转换所需要的模型格式各不相同,MoTIF 会调用相应的格式转换器,自动地将上一个转换的输出转换成公共模型表示,然后再转换成下一个转换的输入格式.MoTIF 组合流程执行模块依据图 13 所示的流程,依次执行每个转换,最终完成整个操作.



转换执行(transformation execution)结点的属性

Fig.13 Definition of composite transformation process

图 13 组合转换流程定义

从这个案例可以看出,本文提出的组合转换模型 CTM 以及支撑工具 MoTIF 能够很好地支持不同转换技术之间的组合.CTM 中的公共类型表示和公共模型表示能够有效解决不同转换技术之间数据格式不同的问题;公共转换描述能够很好地封装由不同转换技术实现的转换规约;组合转换定义语言则能够从静态和动态两个角度准确刻画组合转换.

在实践中,本文提出的组合转换方法和工具支持已经在我们研制的元建模工具[24]中得到应用,用于实现建模工具的自动生成.同时,在我们研制的模型驱动的服务建模工具[25]中,本文的方法也用于实现抽象的业务流程到 BPEL 代码的转换.因此,本文方法和工具的正确性在实践中已经得到一定的验证.

6 讨 论

6.1 复杂性

本文提出的组合转换模型用于支持不同转换技术的组合,使得用不同技术实现的模型转换可以被集成在同一个组合转换中,完成复杂的任务.这样能够提升模型转换的可复用程度,降低开发成本和复杂性.当然,使用本文的方法解决组合转换问题也存在一定的复杂性,但仍在可控制的范围内:第一,使用本文的方法需要编写不同模型存储格式到公共模型表示的格式转换器,但这一工作的复杂性是可以接受的.一方面,这些格式转换器本身是可复用的,一旦实现就可以重复使用;另一方面,如果有 N 种不同的模型存储格式,在不使用公共模型表示的情况下,需要实现 $N\times(N-1)$ 种格式转换器;而在使用了公共模型表示后,只需要实现 $2\times N$ 种格式转换器.因此,公共模型表示不仅没有增加额外的复杂性,还降低了实现组合转换的复杂性;第二,在某些情况下,使用本文的方法

还需要编写不同模型类型存储格式到公共类型表示的格式转换器.但与公共模型表示的情况类似,编写模型类型格式转换器的复杂性也是可以接受的;第三,本文方法要求将不同格式的转换规约统一封装成公共转换描述,但这一步骤可以自动完成,本文第 4 节介绍的工具已经能够提供相应的支持;第四,组合转换的定义需要使用本文提出的组合转换定义语言,该语言扩展自 UML 的活动图.这对于熟悉 UML 的开发人员来说,无需过多的学习成本.综上,本文方法对于解决组合转换中的问题来说,其复杂性是可以接受的.

6.2 完备性

本文提出了公共类型表示和公共模型表示两部分用来统一模型类型(即元模型)和模型的表示格式,本节讨论它们的完备性,即是否能够用它们来表示所有的元模型与模型.

首先,公共类型表示和公共模型表示中定义的概念并不是绝对完备的.例如,公共类型表示中就不包括 Package 的概念.对于目前的转换技术而言,要求绝对的完备性是没有必要的.因为元模型和模型中的某些信息并不会影响转换进行,所以在公共类型表示和公共模型表示中不需要定义相应的概念来记录它们.根据现有的几种典型转换技术和实现平台的特点,本文分析并总结了转换过程中必要的数据结构,并据此定义了公共类型表示和公共模型表示,因此它们都是相对完备的,即它们都能够表示元模型与模型中那些对转换有用的信息.例如,公共类型表示、类型图[2]与 MOF 中核心概念之间的对应关系可以用表 1 表示;公共模型表示、有类型的图及 MOF 实例模型中的核心概念之间的对应关系可以用表 2 表示.从这两个表中可以看出,公共类型表示与公共模型表示覆盖了所有主要的概念,是相对完备的.

Table 1 Correspondences of type graph, MOF, and common type representation

表 1 公共类型表示与类型图、MOF 中主要概念的对应关系

类型图	MOF	公共类型表示
Node	Class	MetaClass
Attribute node	Property	MetaProperty
Edge	Association	MetaRelationship
Composition edge		
Inheritance edge	Class 上的自关联 superClass	MetaClass 上的自关联 parent

Table 2 Correspondences of typed graph, MOF instance diagram, and common model representation

表 2 公共模型表示与有类型的图、MOF 实例图中主要概念的对应关系

有类型的图	MOF 实例图	公共模型表示
Node	ClassInstance	Class
Edge	AssociationInstance	Relationship
Attribute node	Slot	Property
映射 $type_N$,将 Node 映射到类型图中结点	ClassInstance 和 Classifier 之间的关联 classifier	Class 和 MetaClass 之间的关联 classType
映射 $type_E$,将 Edge 映射到类型图中边	AssociationInstance 和 Association 之间的关联 classifier	Relationship 和 MetaRelationship 之间的关联 relatinoshipType
映射 $type_{att}$,将 AttributeNode 映射到类型图属性结点	Slot 和 StructuralFeature 之间的关联 definingFeature	Property 和 MetaProperty 之间的关联 definition

根据实践经验,本文提出的公共类型表示和公共模型表示已经能够处理绝大部分的情况.如果所需要的信息无法转换为公共类型表示和公共模型表示,还可以使用 CTM 基础结构中定义的 Data 元素,它可以表示任意的一个数据项.

6.3 适用性

本文提出的组合转换方法解决了不同转换技术与工具的组合问题.本文方法考虑了不同技术与工具之间可能存在的异构性问题,而现有的关于组合转换的工作都没有考虑这些问题.本文提出的组合转换模型能够屏蔽不同转换技术之间的异构性:公共类型表示和公共模型表示处理了数据格式的差异性,而公共转换描述则统一了转换规约格式的不同.因此,与现有工作相比,本文方法具有更为广泛的适用性,能够支持异构技术和工具

的组合.

例如,对于本文所举的例子,现有的工作均不能很好地实现.其中,TCF 只能组合由它提供的转换语言所定义的子转换,Wires*和 ATL Flow 则只能支持 ATL 转换的组合,因此它们都无法将例子中用 4 种不同技术实现的 5 个转换组合起来;UniTI 中使用 UTR 定义组合转换,虽然考虑了不同转换工具之间转换规约格式的不同(支持 ATL,MTF),但并未考虑不同技术之间数据格式的不同,因此只能支持使用相同数据存储格式的转换技术之间的组合.所以,UniTI 无法将使用 AGG 实现和部署在 MOFLON 上的转换组合起来;此外,UniTI 不支持组合结构中的分支结构,因此不能描述本文案例中的组合流程;MCC 虽然能够实现本例中的复杂转换,但要求开发人员将组合转换中的每个子转换手工封装成 Eclipse 插件,并在每个插件里实现数据格式的转换.显然,这种方式不仅开销较大,而且不够灵活.

7 结 论

本文的主要贡献在于:1) 对组合转换进行了分析,并给出了实现组合转换的 4 个条件;2) 针对基于图的转换技术和基于 MOF 的转换技术,提出了一个组合转换模型 CTM,CTM 中包括了公共模型表示、公共类型表示、公共转换描述、组合转换定义语言等部分,分别用来实现组合转换中的 4 个条件;3) 提出了一种基于 CTM 的组合转换工具,并用一个案例说明本文提出的方法及工具的可行性.本文提出的 CTM 虽然只能解决基于图和基于 MOF 这两类转换技术的组合,但是这两个类别覆盖了大部分的转换技术,因此具有普遍性.与现有工作相比,本文主要关注并解决了不同转换技术与执行引擎之间的异构性问题,从而能够支持异构转换技术之间的组合,这提升了现有转换的可复用机会.同时,它能够将不同的转换技术结合起来,发挥不同转换技术的特点和优势,解决复杂的转换问题,提升模型驱动开发的能力.

进一步的工作首先将集中在如何使本文的方法兼容更多的转换技术,本文主要关注了基于图和基于 MOF 的两种转换技术,今后将继续研究如何支持其他转换技术;其次,研究在 CTM 的支持下,如何进行组合转换的可追踪性问题(traceability),即如何跟踪组合转换的输入到输出的转换过程,这也是一个值得研究的问题,对其的研究有利于提高增量转换的执行效率,也有利于进行逆向转换;最后,对本文提出的方法与工具进行更多的应用,并在实践中继续验证和改进本文的方法.

References:

[1] Miller J, Mukerji J. MDA guide. Formal/2007-11-02. 2003. http://www.omg.org/docs/omg/03-06-01.pdf

[2] Ehrig H, Engels G, Kreowski HJ, Rozenberg G. Handbook of graph grammars and computing by graph transformation: Applications, languages and tools. Singapore: World Scientific Publishing Company, 1999.

[3] Giese H, Wagner R. Incremental model synchronization with triple graph grammars. Lecture Notes in Computer Science, 2006, 4199:543−557. [doi: 10.1007/11880240_38]

[4] Taentzer G. AGG: A graph transformation environment for modeling and validation of software. Lecture Notes in Computer Science, 2004,3062:446−453. [doi: 10.1007/978-3-540-25959-6_35]

[5] OMG. Meta object facility (MOF) core specification 2.0. Formal/06-01-01. 2006. http://www.omg.org/spec/MOF/2.0/PDF/

[6] OMG. Meta object facility (MOF) 2.0 query/view/transformation specification. Formal/2008-04-03. 2008. http://www.omg.org/spec/QVT/1.0/PDF/

[7] Jouault F, Kurtev I. Transforming models with ATL. Lecture Notes in Computer Science, 2006,3844:128−138. [doi: 10.1007/11663430_14]

[8] OMG. MOF model to text transformation language, Version 1.0. Formal/2008-01-16. 2008. http://www.omg.org/spec/MOFM2T/1.0/

[9] Czarnecki K, Helsen S. Classification of model transformation approaches. In: Proc. of the OOPSLA 2003 Workshop on Generative Techniques in the Context of MDA. Anaheim: ACM, 2003. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.122.8124

[10] Liu H, Ma ZY, Shao WZ. Graph transformation based description language for model refactorings. Journal of Software, 2009,20(8): 2087−2101 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3469.htm [doi: 10.3724/SP.J.1001.2009.03469]

[11] ATL Project. ATL transformation zoo. http://www.eclipse.org/m2m/atl/atlTransformations/

[12] OMG. Object constraint language. Formal/06-05-01. 2006. http://www.omg.org/spec/OCL/2.0/PDF

[13] Steel J, Jézéquel JM. On model typing. Software and System Modeling, 2007,6(4):401−413. [doi: 10.1007/s10270-006-0036-6]

[14] IBM. http://www.ibm.com/developerworks/rational/library/05/503_sebas/, 2012-3-7

[15] Kleppe A. MCC: A model transformation environment. Lecture Notes in Computer Science, 2006,4066:173−187. [doi: 10.1007/11787044_14]

[16] Marvie R. A transformation composition framework for model driven engineering. LIFL-2004-10, LIFL, 2004.

[17] Vanhooff B, Ayed D, Baelen SV, Joosen W, Berbers Y. UniTI: A unified transformation infrastructure. Lecture Notes in Computer Science, 2007,4735:31−45. [doi: 10.1007/978-3-540-75209-7_3]

[18] ATL Flow project. http://opensource.urszeidler.de/ATLflow/

[19] Rivera JE, Ruiz-González D, López-Romero F, Bautista J, Vallecillo A. Orchestrating ATL model transformations. In: Proc. of the 1st Int'l Workshop on Model Transformation with ATL. 2009. 34−46. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.160.9313

[20] OMG. OMG unified modeling language (OMG UML), superstructure, V2.1.2. Formal/2007-11-02. 2007. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/

[21] Weisemöller I, Klar F, Schürr A. 16 development of tool extensions with MOFLON. Lecture Notes in Computer Science, 2011,6100:337−343. [doi: 10.1007/978-3-642-16277-0_16]

[22] Acceleo project. http://www.eclipse.org/acceleo/

[23] EMF project. http://www.eclipse.org/modeling/emf/

[24] Ma ZY, Liu H, He X, Zhang L, Ji Z, Ge M. Research and design of a meta-modeling platform for model-driven development. Chinese Journal of Electronics, 2008,36(4):731−736 (in Chinese with English abstract).

[25] Ma ZY, He X, Kang LH. A model driven development platform for service-oriented applications. In: Proc. of the 2009 World Conf. on Services—II. Los Alamitos: IEEE Computer Society, 2009. 17−24. [doi: 10.1109/SERVICES-2.2009.14]

附中文参考文献:

[10] 刘辉,麻志毅,邵维忠.一种基于图转换的模型重构描述语言.软件学报,2009,20(8):2087−2101. http://www.jos.org.cn/1000-9825/3469.htm [doi: 10.3724/SP.J.1001.2009.03469]

[24] 麻志毅,刘辉,何啸,张乐,吉喆,戈牧.一个支持模型驱动开发的元建模平台的研制.电子学报,2008,36(4):731−736.

何啸(1983－),男,北京人,博士生,主要研究领域为元建模,模型转换.

冯超(1985－),男,硕士,主要研究领域为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,CCF 高级会员,主要研究领域为软件工程,软件工程环境,构件技术,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,软件工程环境,软件复用,面向对象技术.